Diss. ETH No. 16318

Computer-Aided Management of Commodity Parts-Based Supercomputers

A dissertation submitted to the
Swiss Federal Institute of Technology Zurich

For the degree of
Doctor of Technical Sciences

Presented by Josef Nemecek
dipl. Informatik-Ingenieur ETH
Born March 31, 1973
Citizen of Wädenswil (Switzerland)
and Czech Republic

Accepted on the recommendation of
Prof. Dr. Bernhard Plattner, examiner
Prof. Dr. Anton Gunzinger, co-examiner

2005

Abstract

Supercomputers are used to solve big problems – they are «nutcrackers» that support scientists, researchers and developers in decoding the human genome, simulating the weather and climate, creating virtual wind tunnels for planes and cars, and designing effective medicaments – the so-called «grand challenge problems». Smaller supercomputers are used for tasks that require more performance than a single computer can deliver: Web-servers, data centers and calculation systems for in-house research.

One buzzword is currently changing the supercomputing market dramatically: Commodity. More and more systems are based on «commodity off-the-shelf parts»: CPUs, memories, storage devices, networking technologies and software that are used in ordinary workstations and servers are also used in these «commodity supercomputers». In the most extreme cases, the supercomputer basically consists of commercially available computers – specialized custom technologies are only used where absolutely necessary. These «clusters of workstations» (or «superclusters») are very likely to replace the conventional supercomputers installed in supercomputing centers, because many of these centers are actively investigating their potential and supporting the research in «commodity supercomputing».

However, one issue inhibits replacing supercomputers with superclusters today: The lack of a comprehensive integrated system management. Manual management using shell commands, scripts and independent tools only works for small systems with a few compute nodes. But for superclusters that consist of some thousand or million different components, computer-aided system management is mandatory. To provide high supercluster availability, this management must be fast, scalable, reliable, and secure.

This dissertation is the first complete research in this area that presents a concept for integrated and comprehensive system management of superclusters. This concept understands management as a *lifecycle* with the three phases: Design and simulation, installation, and operation – although only the operational phase is analyzed in-depth.

The expectations, fears and requirements of the people working with superclusters have been analyzed and compiled into the *eight bottlenecks and central requirements* of supercluster management: Scalability, availability, management integration, reliability, security, (low) overhead, compatibility, and cost. Also the *seven elements* of supercluster management are analyzed and presented: Control, configuration, monitoring, fault detection, trap handling, accounting, and planning. With these two tools in hand (bottlenecks and elements) it is possible to evaluate new and existing management architectures for their suitability for supercluster management.

This thesis presents some management architectures and offers a guide that allows the selection of the optimal architecture depending on system size and user requirements. One of the presented architectures has been implemented for the first Swiss-based supercluster «Swiss-T1», installed at the supercomputing center CAPA of the EPFL. This first implementation (called «COSMOS») is presented in detail in this dissertation, together with the project «Swiss-Tx» that this thesis was part of.

Of the *eight bottlenecks and central requirements*, scalability and availability have been analyzed on a theoretical level and the results have been turned into the presented management architectures – with clusters for availability and proxies for scalability. Integration and overhead are additionally covered in practice with the implementation of COSMOS. The remaining issues (reliability, security, compatibility, and cost) were taken out of the research focus and therefore neglected.

This dissertation proves that comprehensive and integrated supercluster management is an equal partner for enabling commodity supercomputing – together with networking technologies, communication libraries and distributed storage systems. The achievements of the research are blueprints of management architectures, with clustered managers for high availability, layers of proxies for scalability, tightly integrated application management for reliability, and modularity for integration. The implementation proves the concept to be correct, allowing efficient, stable, effective, and comprehensive management of the Swiss-T1 supercluster.

Zusammenfassung

Supercomputer werden gebaut, um grosse Probleme zu lösen – es sind «Nussknacker», um Wissenschaftlern, Forschern und Entwicklern bei der Entschlüsselung des menschlichen Genoms, der Simulation von Wetter und Klimaveränderungen, der Erstellung virtueller Windkanäle für Flugzeuge und Autos sowie der Entwicklung wirksamer Medikamente zu helfen – den so genannten «Grand Challenge Problems». Kleinere Systeme werden für Aufgaben verwendet, welche nicht durch einen einzelnen Computer gelöst werden können: Internet-Server, Datenserver und Rechensysteme für firmeninterne Forschung.

Der Supercomputermarkt wird momentan mit einem Schlagwort umgekrempelt: Commodity. Immer mehr Systeme basieren auf handelsüblichen Komponenten «ab der Stange»: Prozessoren, Speicher, Massenspeicher, Netzwerktechnologien und Software, welche in üblichen Workstations und Servern eingesetzt werden, werden auch auf diesen «Commodity Supercomputers» eingesetzt. Im Extremfall bestehen diese Supercomputer aus kommerziell erhältlichen Computern – spezialisierte Technologien werden nur dort eingesetzt, wo es absolut notwendig ist. Es ist sehr wahrscheinlich, dass die «Clusters of Workstations» (oder «Superclusters») die konventionellen Supercomputer ersetzen werden, welche in Supercomputing Centers installiert sind. Diese Rechenzentren betreiben aktiv Forschung im Bereich «Commodity Supercomputing», um dessen Potential abzuklären.

Supercomputer werden jedoch aus einem Grund noch nicht durch Supercluster ersetzt: Es fehlt ein umfassendes System Management. Das manuelle Management mittels Shell-Kommandi, Skripts und voneinander unabhängigen Tools funktioniert nur bei kleinen Systemen mit wenigen Rechenknoten. Für Supercluster mit einigen Tausend oder Millionen Komponenten ist jedoch ein computergestütztes Systemmanagement notwendig. Um eine hohe Verfügbarkeit des Superclusters zu erlauben, muss es schnell, skalierbar, zuverlässig und sicher sein.

Diese Dissertation ist die erste vollständige Forschungsarbeit in diesem Gebiet und präsentiert ein Konzept für ein integriertes und umfassendes Systemmanagement von Superclustern. Dieses Konzept sieht Systemmanagement als einen Lebenszyklus mit den drei Phasen Entwurf und Simulation, Installation und Verwaltung – wobei nur die Verwaltungsphase detailliert analysiert wird.

Die Erwartungen, Ängste und Anforderungen der Menschen, welche mit Superclustern arbeiten, wurden analysiert und zu den *acht Flaschenhälsen und zentrale Anforderungen* zusammengefasst: Skalierbarkeit, Verfügbarkeit, Integration der Verwaltung, Zuverlässigkeit, Sicherheit, (geringer) Zusatzaufwand, Kompatibilität und Kosten. Auch die *sie-*

ben Elemente des Supercluster Managements werden analysiert und präsentiert: Kontrolle, Konfiguration, Überwachung, Fehlererkennung, Warnsignale, Verrechnung und Planung. Mit diesen zwei Werkzeugen (Flaschenhälse und Elemente) ist es möglich, bestehende und neue Architekturen für Management Software zu evaluieren, ob sie für die Verwaltung von Superclustern geeignet sind.

Diese Dissertation präsentiert einige Management-Architekturen und bietet einen Führer an, welcher die Wahl der optimalen Architektur gestattet, abhängig von der Systemgrösse und den Anforderungen der Benutzer. Eine der präsentierten Architekturen wurde für den ersten Schweizer Supercluster «Swiss-T1» des Rechenzentrums CAPA der EPFL implementiert. Diese erste Implementation (mit dem Namen «COSMOS») wird in dieser Dissertation detailliert präsentiert, zusammen mit dem Projekt «Swiss-Tx», deren Teil diese Arbeit war.

Von den *acht Flaschenhälsen und zentralen Anforderungen* wurden die Skalierbarkeit und die Verfügbarkeit auf einem theoretischen Niveau analysiert und führen zu den präsentierten Architekturen – mit Clustern für die Verfügbarkeit und Proxies für die Skalierbarkeit. Integration und Zusatzaufwand werden in der Praxis durch die Implementation von COSMOS abgedeckt. Die verbleibenden Punkte (Zuverlässigkeit, Sicherheit, Kompatibilität und Kosten) waren nicht Bestandteil der Forschung.

Diese Dissertation beweist, dass umfassendes und integriertes Supercluster Management ein gleichberechtigter Partner ist für die Ermöglichung des «Commodity Supercomputing» – zusammen mit den Netzwerktechnologien, Kommunikationsbibliotheken und verteilten Festplattenspeichersystemen. Das Ergebnis dieser Forschung sind Baupläne für Managementarchitekturen, mit geclusterten Managern für hohe Verfügbarkeit, Schichten von Proxies für die Skalierbarkeit, straffe Integration des Applikationsmanagements für die Zuverlässigkeit sowie Modularität für die Integration. Die Implementation beweist die Korrektheit des Konzepts, welches eine effiziente, stabile, effektive und umfassende Verwaltung des Swiss-T1 superclusters ermöglichte.

Preface

The title of this thesis, *Computer-Aided Management of Commodity Parts-Based Supercomputers*, is very bold. It is big enough to cover a standards book, a universal description or a research manual. Since the specific research area is new and the author tends to be both precise and complete, there was quite a risk that this dissertation would turn into such a shelf-bending monster. To be honest, the first drafts were rather like that.

During the long editing time and the many reviews with the examiners, this book turned into that which dissertations are initially meant to be: A reporting guide. It is a guide for those readers, who will travel to similar destinations. And it is a report for those readers, who want to see the beauty of the paths followed by the author himself. As an expedition report and travel guide to yet undiscovered countries, this dissertation will not disappoint both types of readers.

As the title suggests, there are two main topics:

- *Supercomputers*, particularly those that are based on *commodity off-the-shelf parts*
- *Management*, especially the *computer-aided* type

The combination of these topics creates some questions that are answered in this dissertation:

Question	Chapter
What are «supercomputers» and what are they used for?	1
What are «commodity parts-based» supercomputers?	
Why are they so interesting?	
What does «management» mean and what does it include?	2
How can this be «computer-aided»?	
Where are the problems?	
What are the possible software designs for this management?	3
Do those designs solve the problems?	
What did others do to solve the problems?	4
Why is there still a need for more research?	
What did the author do to solve the problems he faced?	5
What did the author's project look like?	
Did his design solve the problems?	6
What would he do differently next time?	
What else needs to be done?	

Table 1: Arising questions and where they are answered in this thesis

This dissertation thesis describes how modern supercomputers will be managed using specialized software (guide) and presents the author's management software design that was created for the «Swiss-Tx» project's supercomputers (report).

The structure of this dissertation is quite classical, as the chapter numbers next to the questions suggest. In order to keep the dissertation fluent and interesting, sections considered too detailed or potentially boring were moved to the appendices.

In contrast to the classical structure of the dissertation, the project structure was quite unorthodox: The research work (occupying the first chapters) happened *after* the software was designed within the project (occupying the last chapters). This inverted order allowed for some interesting observations and procedures:

- The presented theoretical software architectures are better because of the practical experience made with the prototype.
- The selected design of the author is better analyzed, since all possible paths were discovered *after* the goal had been reached.
- The difficulties experienced opened the eyes to further potential problems.
- After being a «management guy» too, it was easier to discuss with administrators and other «management guys» the advantages and problems of creating management tools.
- Surprisingly, the author's intuition selected the optimal path for the project.
- The author was at first disappointed by his design, since the research work discovered that there were more beautiful paths to travel – on the other hand, these paths would have remained undiscovered if the less beautiful path had not previously been taken by the author.

The project structure suggested a second system to the author, but it was not created, because the succeeding project step was not taken: A larger supercomputer requiring this second system was not ordered, leaving the author unhappy. After time had healed the wounds created by the project's termination, this dissertation allowed a clearer view of the results, making the author happy and proud of his created thesis.

Now, about 3 years of editing later, this reporting guide can be published with pride, and also with some hope, that the guiding part will guide the developers who are following the paths, and the reporting part will demonstrate that also rapidly created prototypes have their beauty.

Table of Contents

1 Introduction

> *The cost of CPU chips is [...] leading to systems containing a large number of processors. Connecting all these processors using standard technology [...] is easy. The hard part is designing and implementing software to manage and use all of this computing power in a convenient way.*
>
> *Andrew S. Tanenbaum [TKR+91]*

Since their introduction in the last century, computers have been used to solve computational problems or problems that are based on large amounts of data. Computer technologies have enabled mankind to create stable buildings, to conquer the cosmos, and to manage complicated machineries.

Supercomputing has industrial targets: Building safer cars, creating more efficient airplanes, designing more effective medications – in short: Allowing a convenient life for everyone. Because supercomputers have a high computing performance (about 1 000 to 10 000 times higher than common computers), they allow problems to be solved within hours that would take years on ordinary computers.

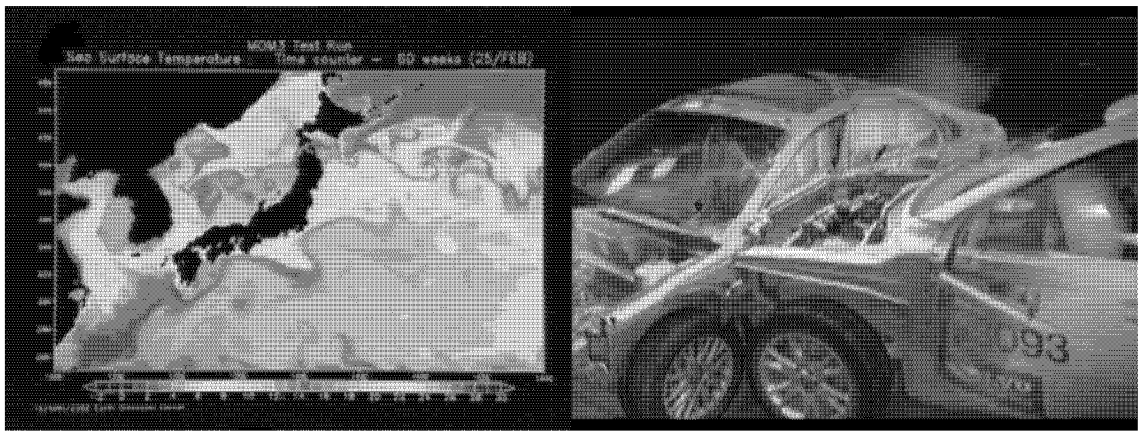


Figure 1.1: Two common supercomputer tasks: Climate simulation and virtual crash test

Achieving this computing performance requires high-tech and creativity. Many new technologies are introduced and increase the supercomputing performance every year. And it is necessary to continue these efforts, because there are many applications that are capable of keeping systems busy for decades, no matter how fast they are [Jaf84]:

- Analysis of the human genome and molecular structure of proteins, simulation of the cellular response to designed drugs. This enables effective and efficient drugs to be found without tests on animals needing to be done.
- Simulation of meteorological effects for reliable long-term weather and climate forecasts. Precise and reliable forecasts prevent casualties and costly damages.
- Virtual wind/water tunnels to simulate pressure distributions on surfaces. This enables planes, ships and cars to be created that use fuel more efficiently.

This chapter introduces supercomputing, shows what supercomputers look like, how they work and how they will look in a few years if the current trends continue.

1.1 Supercomputing in a Nutshell

As in all other niches in computing, supercomputing has its own vocabulary and logic. Although the glossary (see appendix F) covers most of the commonly used terms, some of them are worth presenting briefly individually, together with some supercomputing basics.

1.1.1 Supercomputing Newspeak

The most commonly used performance abbreviation in supercomputing is **FLOPS** (floating point operations per second). It explains how fast a processor or supercomputer is. One million (10^6) floating point operations per second are one **MFLOPS**, one billion (10^9) are one **GFLOPS**, and one trillion (10^{12}) are one **TFLOPS**, etc.

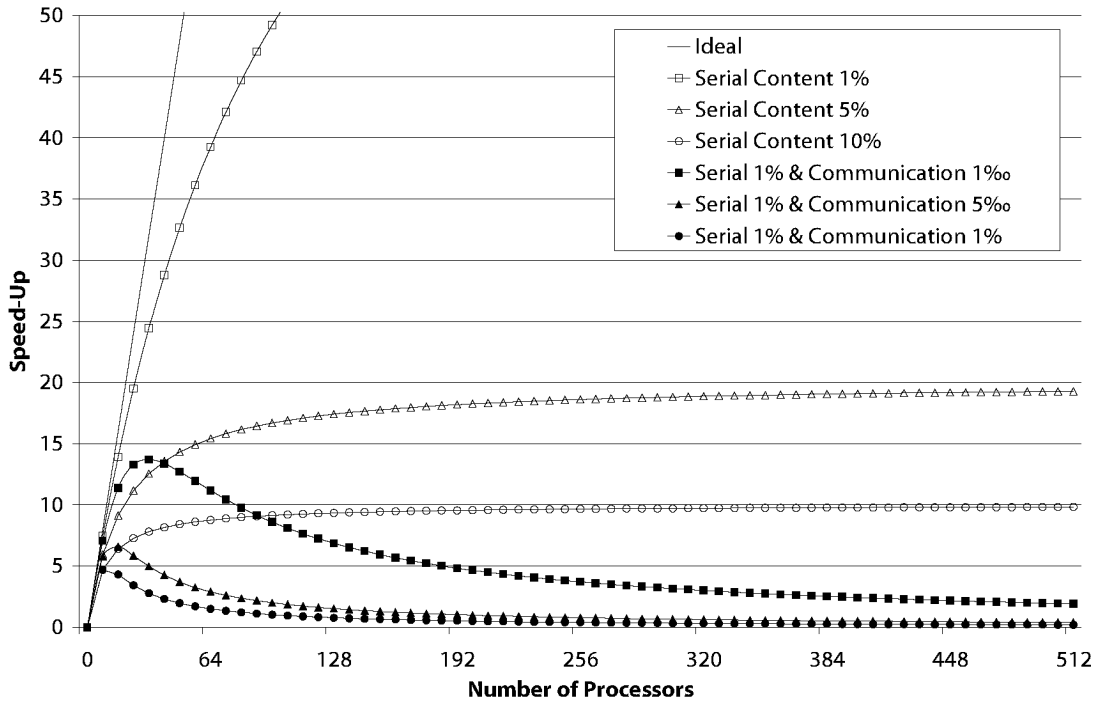


Figure 1.2: Amdahl's law: Speed-up in relation to processor count

Processor or supercomputer manufacturers present their products with **peak performance**. The manufacturer guarantees that the system cannot exceed this performance. It is usually driven by the clock rate and the operations started per clock. Four execution units with 2.5 GHz give 10 GFLOPS of peak performance.

The **sustained performance** is the performance, at which the processor or supercomputer is able to execute real-life applications. System-internal and application-specific bottlenecks reduce the peak performance by 5–85%. The fastest supercomputers have an efficiency rate of 50–90%.

The magic word in supercomputing is **speed-up**, which describes the performance relation between two systems. Since all supercomputers are multi-processor systems, the speed-up today relates the performance of systems to the different processor counts.

If the speed-up grows almost linearly with the processor count, the speed-up **scales**. In some cases there is a performance limit, in other cases the performance decreases with additional processors. **Amdahl's law** describes this behavior (see Appendix A).

1.1.2 Supercomputer Classification

The goal of every supercomputer architect is to create a computing system with superior performance. Since computers execute instructions on data, the possibilities for increasing the performance are limited:

- Increase the processor clock rate
- Increase the amount of data being handled by one instruction
- Increase the amount of processors
- Distribute the memory to the processors and interconnect these nodes

All these formulas can be put into categories, leading to the computer classifications as described by the computer architects Michael Flynn [Fly72] and Gordon Bell [Bel92].

1.1.2.1 Classification according to Michael Flynn

Computer architect Michael Flynn analyzed the instruction and data streams through the computer [Fly72]. Each stream type can be either alone (single stream) or there can be concurrent streams (multiple streams). This leads to a map with four areas:

		Data Streams	
		Single	**Multiple**
Instruction Streams	**Single**	SISD	SIMD
	Multiple	MISD	MIMD

Table 1.1: Architecture classification according to Michael Flynn [Fly72]

The simplest architecture is **SISD** (Single Instruction, Single Data). There is one processor in the computer, executing one application, operating on one data item at a time. This category includes conventional single-CPU computers such as PCs and worksta-

tions. The performance is increased by increasing the clock rate and other such tricks (e.g. pipelining). Signal propagation delays, heat generation and other physical effects limit the clock rate to some GHz.

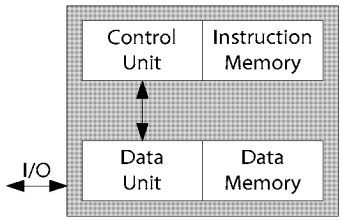


Figure 1.3: SISD computer architecture: One control unit manages one data unit

Allowing one single instruction to manipulate multiple data items at the same time leads to the **SIMD** architecture (Single Instruction, Multiple Data). These multiple data items can be vector elements that are added or multiplied. This category includes vector supercomputers and modern PC processors equipped with data-parallel processing technologies such as MMX [INTEL] or AltiVec [IBM]. A famous supercomputer of this category is the CRAY 1 with 160 MFLOPS and 8 MBytes of RAM, first installed in 1976 at Los Alamos National Laboratory for US$ 8.8 million. The performance is increased by increasing the number of data items manipulated by one instruction (and increasing the clock rate, enhancing the memory interface etc.).

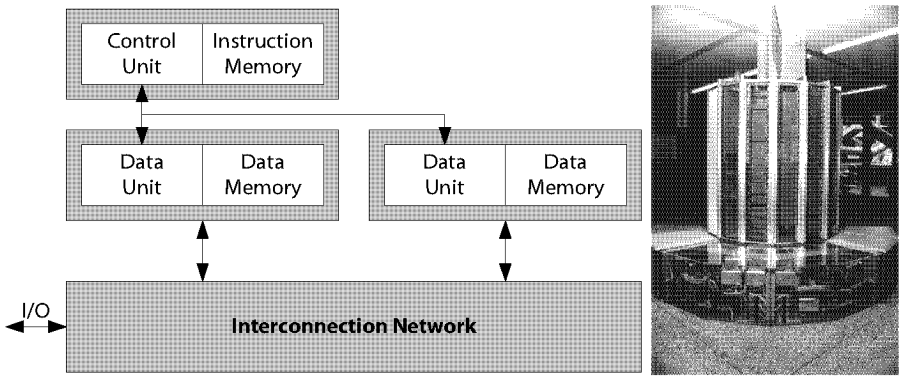


Figure 1.4: SIMD computer architecture (left) and CRAY 1 (right): One control unit manages multiple data units

The **MISD** architecture (Multiple Instruction, Single Data) is without supercomputing relevance today. It is used for specialized processors, where multiple applications operate on the same data, such as DSPs, audio processors or graphic pipelines.

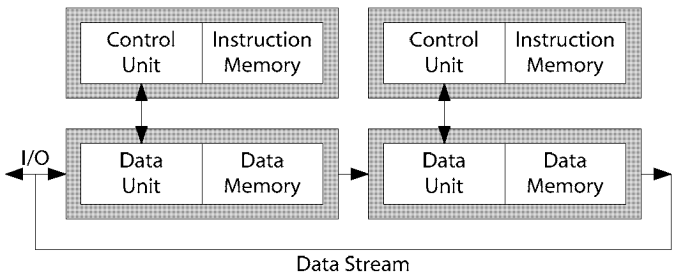


Figure 1.5: MISD computer architecture

«*Divide and conquer*» is the philosophy behind **MIMD** (Multiple Instruction, Multiple Data). If adding data units increases the complexity so much that the clock rate is compromised, additional control units with their own instruction streams are used, each managing one or more data units. These nodes (control units with associated data units) are independent and operate autonomously, but they require a system-area network for data exchange and an inter-node synchronization mechanism. This category includes multi-CPU computers and all modern supercomputers. System performance is increased by adding nodes (and increasing node and network performance).

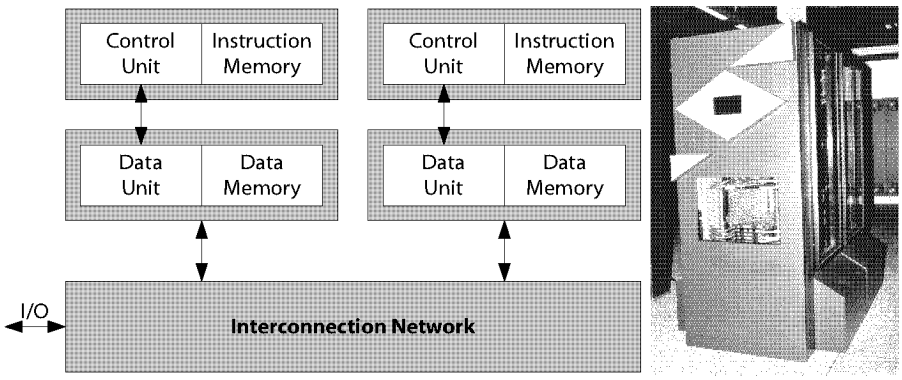


Figure 1.6: MIMD computer architecture (left) and CRAY T3D (right): Interconnected nodes (control unit with associated data unit)

Since all modern supercomputers are MIMD systems, and because there are big architectural differences between these systems, an additional classification for MIMD systems is necessary.

1.1.2.2 Classification according to Gordon Bell

The classification according to Gordon Bell [Bel92] analyses the memory model of the MIMD systems and divides them into two basic types: Multiprocessors and multicomputers.

Categories			Examples
MIMD	**Multiprocessors** • Single Address Space • Shared Memory	**Distributed Memory (NUMA)** (scalable)	KSR, Alliant, BBN, Cedar, Cray T3, NUMA Systems
		Centralized Memory (SMP) (not scalable)	Cray Vector, Fujitsu, IBM, NEC, Tera, SMP Systems
	Multicomputers • Multiple Address Spaces • Message Passing	**Distributed Multicomputers** (scalable)	Intel, NCUBE, Tandem, Compaq, NOW/COW
		Centralized Multicomputers (fixed network, not scalable)	SCS GigaBooster

Table 1.2: Classification of MIMD systems according to Gordon Bell [Bel92]

Multiprocessor systems are computers with many processors that have a common, shared memory space. They are mainframes with either centralized memory (SMP = symmetrical multiprocessing) or distributed memory (NUMA = non-uniform memory access). All processors can access each memory cell. In SMP systems, memory access is identical for all nodes. In NUMA systems, foreign memory access is slower than local

memory access. Most multi-CPU computers have this architecture. For supercomputing, the performance is limited too quickly, as the memory access is the system bottleneck (bandwidth, synchronization).

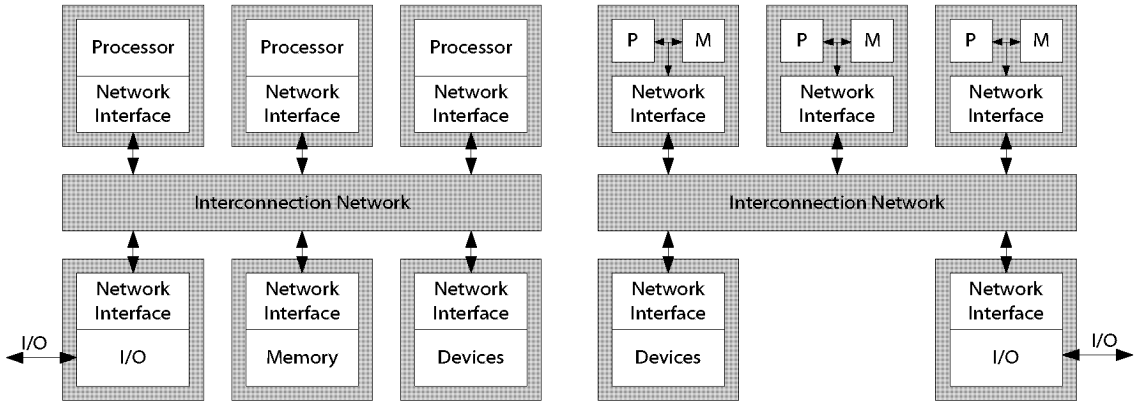


Figure 1.7: Multiprocessor MIMD architecture (SMP left and NUMA right)

Multicomputer systems are interconnected independent computers. Every computer contains one or more processors with its own memory and a network interface. Each processor can only access its own memory; data of foreign memories is sent and received over the network. The fastest supercomputers have this architecture, designed as modular mainframes (as the IBM/SP) or as clusters of workstations (COW).

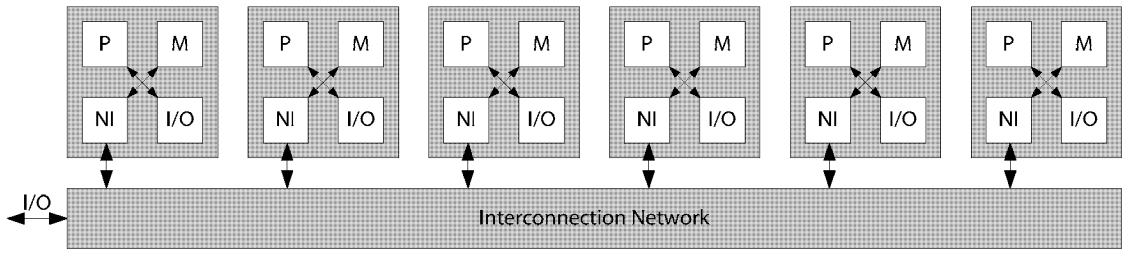


Figure 1.8: Multicomputer MIMD architecture

1.1.3 Supercomputer Utilization

Supercomputers are used for high-performance calculations. These calculations are done in applications. The users submit applications as a job into a queue. A management application – the scheduler – selects a job from the queue as soon as the system is available. There can be multiple queues (e.g. daytime, night, and weekend) that are activated by the scheduler. This job selection is based on a scheduling algorithm that uses user/department/project priorities, estimated run time and other information. The job is then started on the supercomputer, and the application fetches input data from storage and saves output data to storage. Once the job is finished, the supercomputer is cleaned up and the next job is started from the queue.

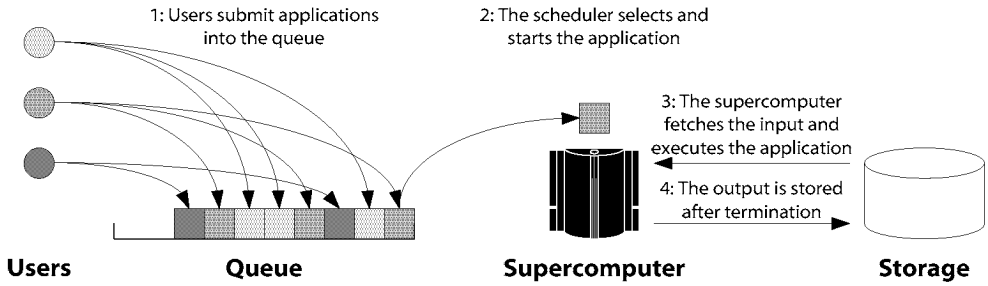


Figure 1.9: Basic supercomputer utilization – batch jobs in a queue

Applications can either terminate successfully, or they can be aborted. This abortion can either be caused by a programming error or by the scheduler: If an application consumes too many resources (e.g. too much time), the application receives a signal and aborts immediately.

The utilization is measured and accounted to the users, departments and projects.

1.1.3.1 Utilization of SISD and SIMD Systems

The utilization is very simple in SISD and SIMD systems, since only one application is executed by the processor within the supercomputer at a time. Each application «owns» the whole system during its execution.

1.1.3.2 Utilization of MIMD Systems

The utilization of MIMD systems is more complicated. The presence of multiple processors that can run in parallel and autonomously allows three basic utilization models:

In the first model, the MIMD supercomputer is used as **multiple SISD/SIMD computers**. Each processor executes an application with its own portion of main memory. The scheduler for this model is very simple, since it distributes the jobs to each processor separately, handling the parallel supercomputer as independent SISD/SIMD supercomputers.

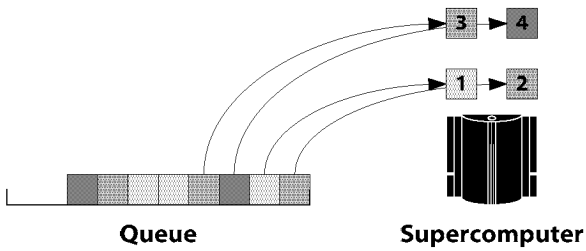


Figure 1.10: Utilization of a 4-processor MIMD system as 4 parallel SISD/SIMD systems

The applications do not know anything about the fact that they are running on a parallel supercomputer. The developer does not need to include any parallel logic or link any parallel library. He can use the MIMD system as a plain old SISD/SIMD system.

In the second model, the MIMD supercomputer is used as a **processor/co-processor system**. There is one application running on all processors – the application uses one processor as the main processor and all other processors as co-processors. The scheduler for this model is also simple, since it handles the parallel supercomputer as one SISD/SIMD system, where only one application is running at any one time.

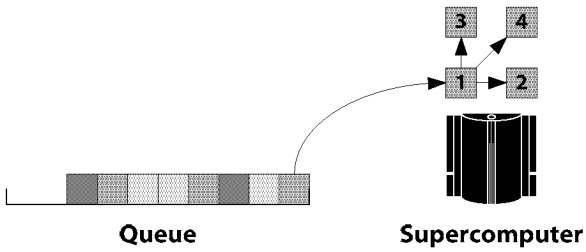


Figure 1.11: Utilization of a 4-processor MIMD system as a processor/co-processor system

The application must handle the arising complexity of parallelism (synchronization, mutual exclusion etc.) itself. It distributes parts of the work to the co-processors which then execute the code in parallel. As in multithreaded applications, communication between the processors (or threads) happens through signals and shared memory. This functionality is usually covered by multiprocessing libraries (e.g. OpenMP [OPENMP]).

In the third model, the supercomputer is used as a **parallel system** where each application can use between one and all available processors. The user submits the job together with the requirements (processor count range, estimated job duration). The scheduler for this model requires some kind of artificial intelligence, because each job needs to be entered as a rectangle in a matrix with the two dimensions being processors and time. As in a hotel's rooms rack (or the famous game Tetris), it is the scheduler's job to place the jobs in this matrix with maximal utilization and least wasted space.

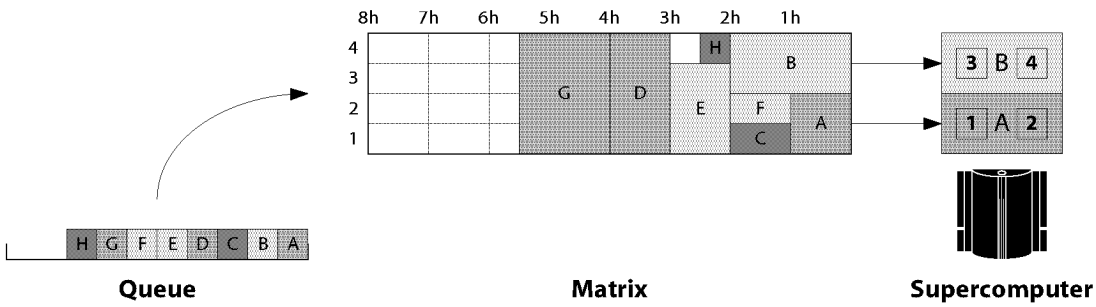


Figure 1.12: Utilization of a 4-processor MIMD system as a parallel system

The scheduler starts the application as processes on the allocated processors and informs each process of the application size (number of processes) and its rank within the application (from one to the number of processes). There are multiple independent applications running on the system. The communication between the application processes happens through messages (using a messaging library such as MPI [MPI95] [MPI97]).

1.2 Supercomputing History

The computer market is divided into segments, which are usually defined by the users' needs, such as performance, feature quality and count, available options, and price. During the last two decades, this segmentation has changed dramatically, this having basically been caused by the unification and improvement of computing technologies.

1.2.1 Computing in the Past

In the past, the computing universe was separated into six segments, the home computers at the bottom with the lowest price and least performance, and the supercomputers at the top providing the greatest performance at the highest price. Most manufacturers were present in one segment only, but bigger manufacturers were present in several segments.

Category	Selling Price	Typical Manufacturers
Supercomputer	10–50 M$	CDC, Convex, Cray
Super-Mini	1–5 M$	IBM
Minicomputer	500 k$	DEC
Workstation	50 k$	HP, SGI, SUN
Personal Computer	5 k$	IBM, Apple, Compaq
Home Computer	1 k$	Commodore, Atari

Table 1.3: Computer segmentation in the past (early 1980's)

The borders between the segments were basically technological borders, combined with system features and expandability provided. At the bottom, high-volume low-cost integrated components were used, such as microprocessors and ordinary memories. At the top, more specialized components were used to provide higher performance and more features, using customized processors, buses and network technologies.

For supercomputers, only leading-edge technologies were used since the computation performance was the main driver. The processors were based on discrete parts using the latest and fastest semiconductor technologies:

- Special wiring techniques allowed transportation of data very quickly internally and between boards containing processors, memories and storage.
- Exotic cooling was necessary to push the clock rates to the physical limit.
- Custom cabinets and interconnects were invented to package the boards as closely as possible to each other to reduce wire lengths and storage area.

The supercomputers of the past were full-custom SISD or SIMD designs, most often vector computers. Processors, memories, bus technologies, cabinets, storage – everything was developed, designed, produced and assembled by the manufacturer himself. The software was specifically tailored for the supercomputer: Operating system, tools, compilers and libraries were designed and tuned to allow the applications to approach peak performance as closely as possible. All this required great effort in research and development, but also led to high costs and long development cycles.

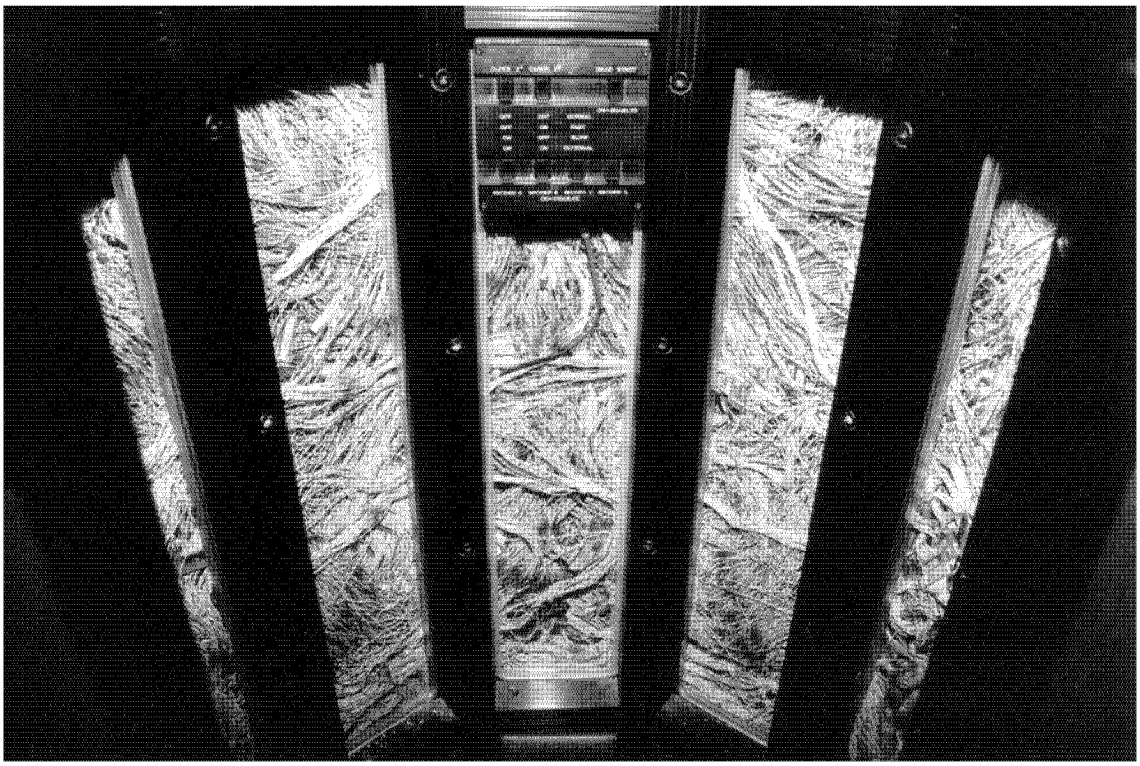


Figure 1.13: Internal wiring of the Cray Y-MP supercomputer

In short, supercomputers of the past provided superior computation performance using leading-edge technology and experimental architecture. This technology created serious implementation problems (wire length, clock rates, packaging, and heat) that made smart solutions necessary by building intelligent supporting technology. Huge development effort with a long time-to-market period was the result, together with high costs and selling prices.

1.2.2 Computing in the Present

Today, the computing universe is still divided into segments, but into different and a smaller number of segments than before. Most computer manufacturers produce computers for multiple segments now, because there are no fundamental technological differences between the computer market segments.

Category	Selling Price	Typical Manufacturers
Supercomputer	20 M$	HP/Convex, NEC, SGI, Cray
Super-Server	2 M$	Compaq/HP, IBM, SGI, SUN
Server	200 k$	Compaq/HP, IBM, SGI, SUN
Workstation	20 k$	Compaq/HP, IBM, SGI, SUN
Personal Computer	2 k$	Intel, Apple

Table 1.4: Computer segmentation today (late 1990's)

Apart from the supercomputers produced in Japan, the computers of all segments basically use the same technology: Highly integrated CMOS semiconductor parts, and often

commodity off-the-shelf components. The only difference is the quality and number of components used within a computer. Multiprocessor computers with multiple memory banks and multiple I/O buses can be found in every segment today.

The microprocessor has developed into a supercomputer killer – according to Moore's law [Moo65], the number of transistors on a chip doubles every two years, leading to a performance increase of microprocessors by 50% every year. Full-custom supercomputers in contrast increase their performance only by 25% every year [CSG98]. It is also less expensive to couple multiple microprocessors than to push the performance of a single processor to achieve the same performance. It was therefore only a question of time until a massive-parallel supercomputer using microprocessors would provide the same performance as full-custom supercomputers.

Most supercomputer manufacturers use custom components for their system design, but the processors in use are often commercial microprocessors, sometimes slightly modified to match the needs of the supercomputer users. To fit maximal performance into a minimal space, the experiences previously gained are still in use: Special racks with sophisticated interconnections of the boards and modern cooling mechanisms allowed for dense systems with high processing power. The development time has decreased through using microprocessors, but the custom packaging and supporting technology still need a lot of development effort.

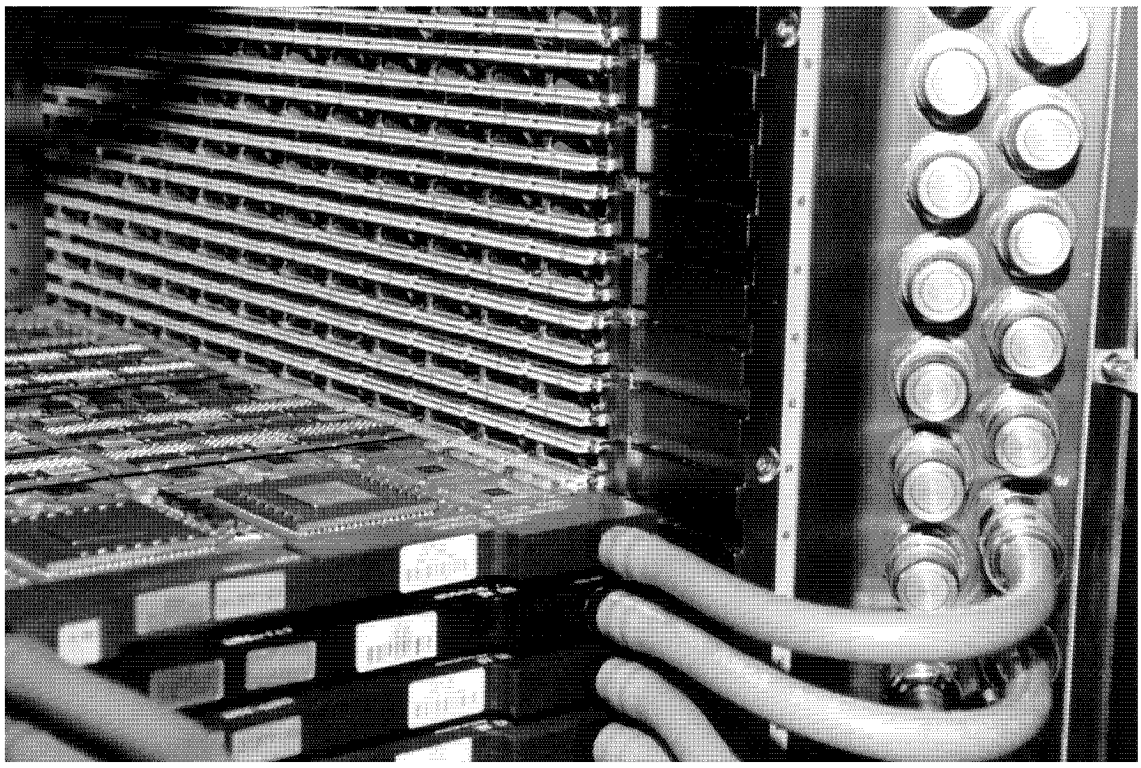


Figure 1.14: Rails with integrated connectors for the Cray T3D processor boards

In short, the supercomputers of today use commodity parts where they have outperformed the previously superior technology, according to the slogan «if you can't beat them, join them». Some specialized technology is still in use to maximize system performance and density.

1.3 The Supercomputer Top500 List

Supercomputers have a computation performance and this performance can be measured and compared. Although every supercomputing application has its own system requirements and scalability behavior, the supercomputing community selected a benchmark suite to compare the installed supercomputers worldwide: The LINPACK[1] benchmark, introduced by Jack Dongarra, solving a system of linear equations [Don94].

The Supercomputer Top500 List [Top500] contains the 500 fastest supercomputers worldwide and is updated twice a year (in June and November). The archive of the website allows analysis of the data back to June 1993, when the list was first published.

1.3.1 The Current List (June 2005)

According to the list of June 2005, the fastest supercomputer available is the BlueGene/L, an MPP supercomputer installed at LLNL (USA).

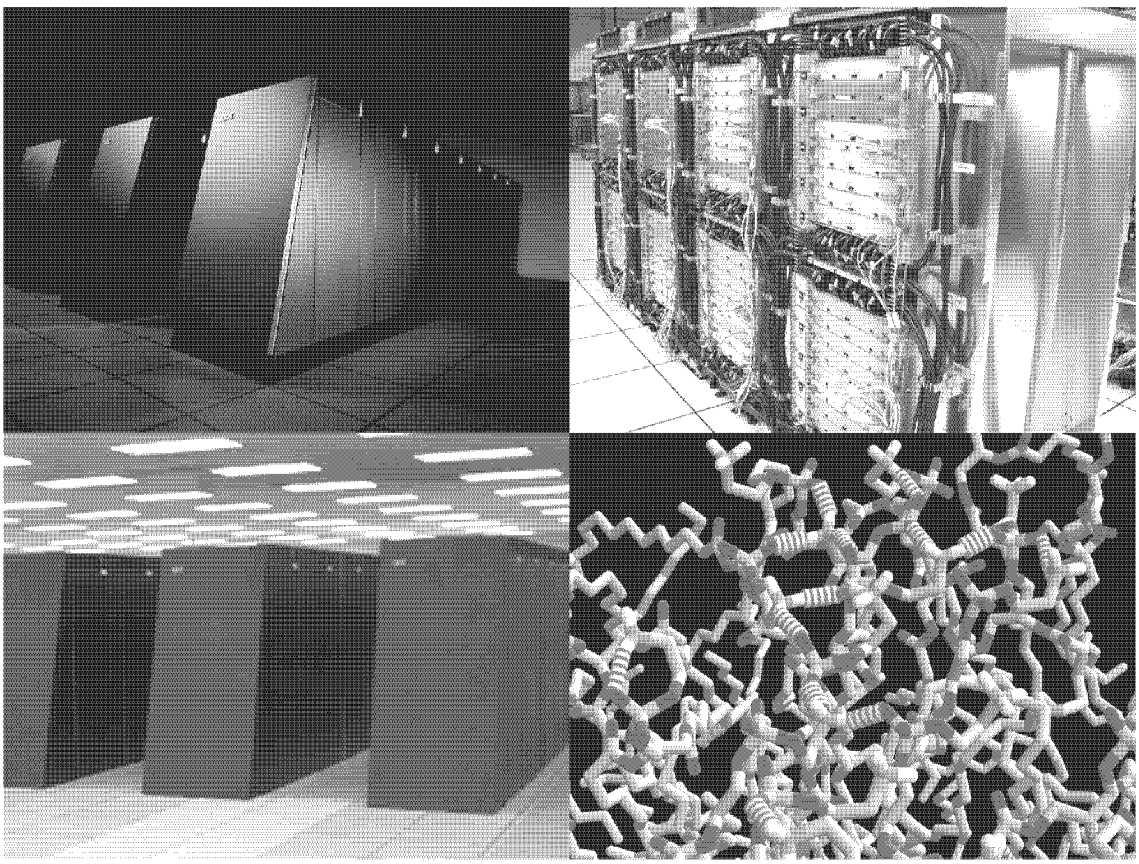


Figure 1.15: BlueGene/L – The fastest supercomputer today: Preview (top left), installation (top right), real system (bottom left) and sample application (bottom right)

[1] LINPACK is a standardized library for linear algebra

It is a supercomputer using off-the-shelf components, with custom modular system architecture and design. This design allows supercomputers of various sizes to be created, which makes the BlueGene not a unique supercomputer, but part of a supercomputer product family that will be installed many times. The BlueGene/L at LLNL (Lawrence Livermore National Laboratories) has 65 536 processors (IBM PowerPC 440 with 700 MHz) and delivers 136.8 TFLOPS sustained performance.

Some of the following systems are the supercomputers that have been built within the ASCI project of the US government. The ASCI project has the principal goal of building supercomputers that are fast enough to simulate the effectiveness and efficiency of nuclear weapons. Of course, the computers are also open to civil applications. The ASCI machines are installed at the national laboratories in Albuquerque (Sandia), Livermore and Los Alamos. They have been built by major computer (not supercomputer!) manufacturers using interconnected workstations (Compaq), super-servers (IBM and SGI) or using custom boards with commodity parts (Intel). The smallest ASCI machine contains 5808 microprocessors, the largest 9632 microprocessors – commodity off-the-shelf CPUs that are also found in workstations and servers all over the world.

1.3.2 Supercomputer Performance

When the list was first published in June 1993, the fastest supercomputer was a Connection Machines CM-5 with 1024 processors and a sustained performance of 60 GFLOPS (peak 131 GFLOPS). The total performance of the top 10 supercomputers was 280 GFLOPS and the top 500 supercomputers provided totally 1.2 TFLOPS.

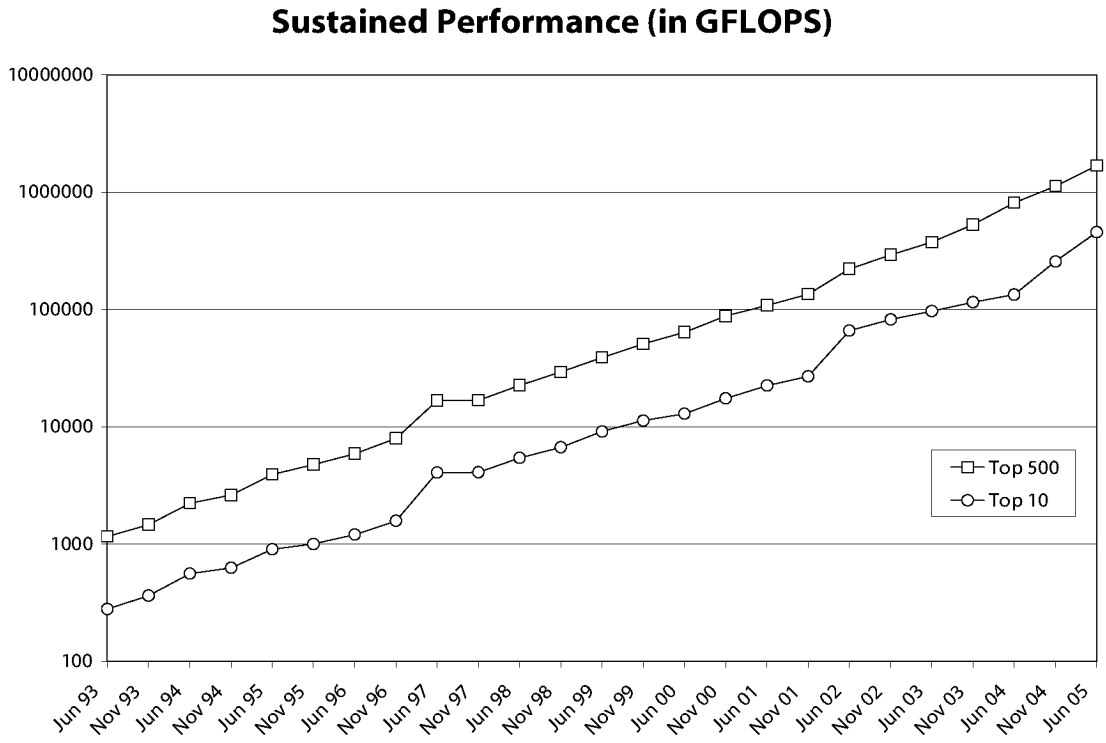


Figure 1.16: Sustained performance (in GFLOPS) of the top 10 and 500 supercomputers; Please note the logarithmic scale of performance!

Today, the fastest supercomputer is the IBM BlueGene/L supercomputer with 65 536 processors and 136.8 TFLOPS of sustained performance (peak 183.5 TFLOPS). The total performance of the top 10 supercomputers is 457 TFLOPS and the top 500 supercomputers provide a total of 1 687 TFLOPS.

The performance of the top 10 and top 500 supercomputers doubles every year. The performance of all supercomputers in the top 500 is reached by the top 10 supercomputers 2½ years later.

1.3.3 Supercomputer Architectures

When the list was first published, about half of the supercomputers were SMP machines (MIMD multiprocessors), one quarter was SISD or SIMD machines and the remaining quarter was MPP (massive parallel processing) systems (large MIMD systems).

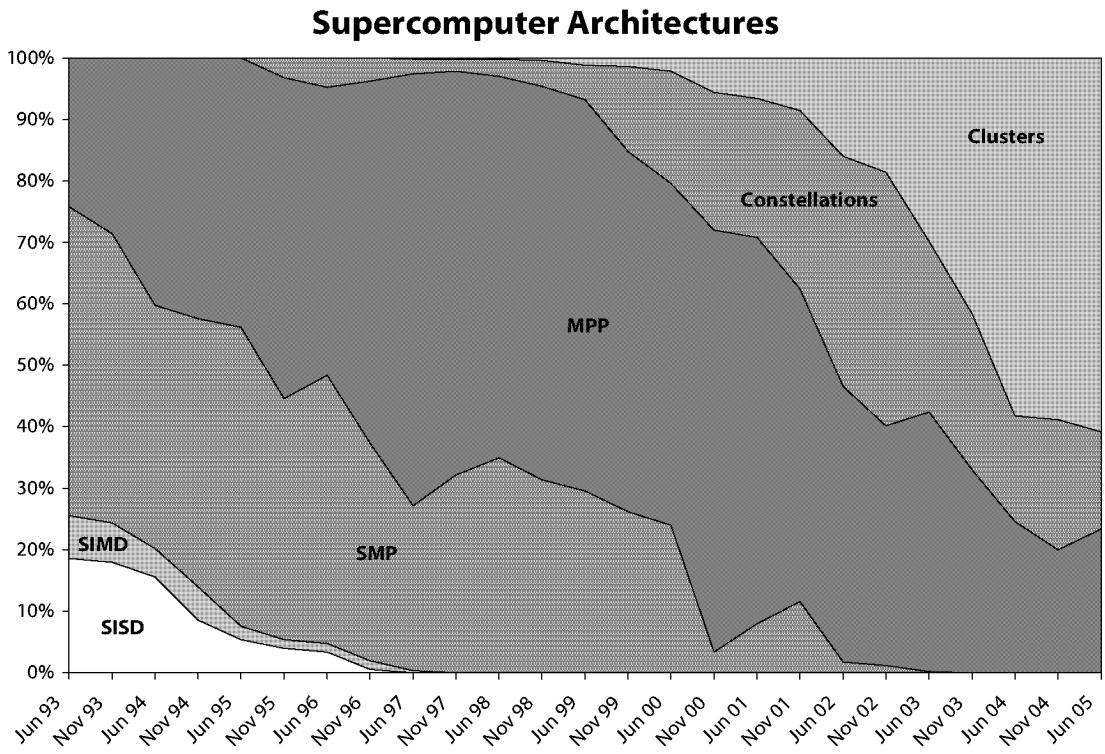


Figure 1.17: Supercomputer type classification in the Top500 lists

The SISD and SIMD systems have completely disappeared since the November 1997 list, and the SMP systems disappeared in November 2003. The reason for this is the design overhead for computer-internal communication technology between processors and memories, which is much higher than that for designing network interfaces which are interconnecting computers.

For a long time, most of the supercomputers were MPP systems – interconnected independent computing units (called processing elements (PE) or nodes). The MPP systems became more important as massive-parallel algorithms became available and the parallel applications performed better than the applications on SISD, SIMD or SMP systems.

Starting with the November 1995 list, «constellations» became more important. Constellations are clustered powerful MIMD multiprocessor mainframes. Clusters first appeared on the November 1998 list – these are interconnected off-the-shelf computers that use either commodity parts only (NOW) or commodity and custom parts (COW), where the custom parts are usually high-speed communication hardware. Clusters and constellations make up three quarters of the list entries today.

1.3.4 Processor Types

When the list was first published, SIMD and scalar (SISD) processors were the minor players while vector processors were found in the majority of supercomputers.

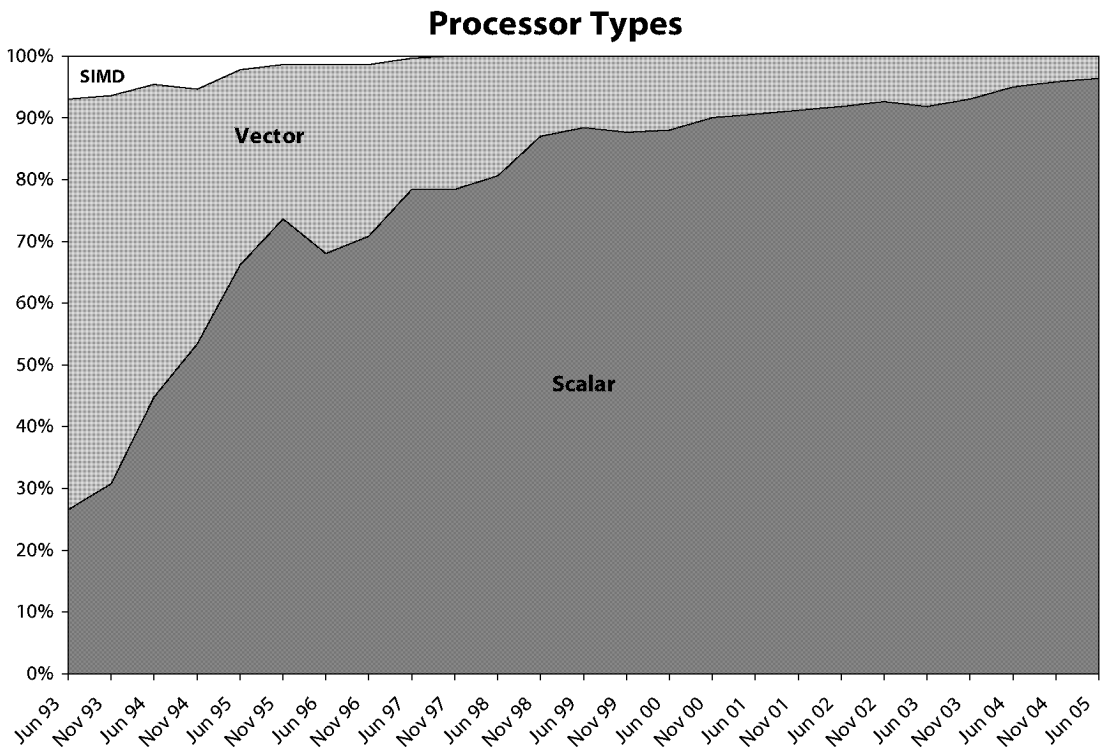


Figure 1.18: Processor type classification in the Top500 lists

Today, the SIMD processors have disappeared and the vector processors can be found in only 5% of supercomputers, which mainly come from Japan. The rest of the supercomputers on the list – about 95% – are based on scalar processors, which are commodity off-the-shelf microprocessors such as IBM Power, Sun SPARC, SGI/MIPS, Compaq/HP Alpha, and Intel x86.

1.3.5 Processor Count

Since the first published list, the number of processors in the top 10 supercomputers has increased from 3 912 to 164 344, and the total number of processors in the whole list has increased from 71 707 to 580 336. This increase in the number of processors is very likely to continue.

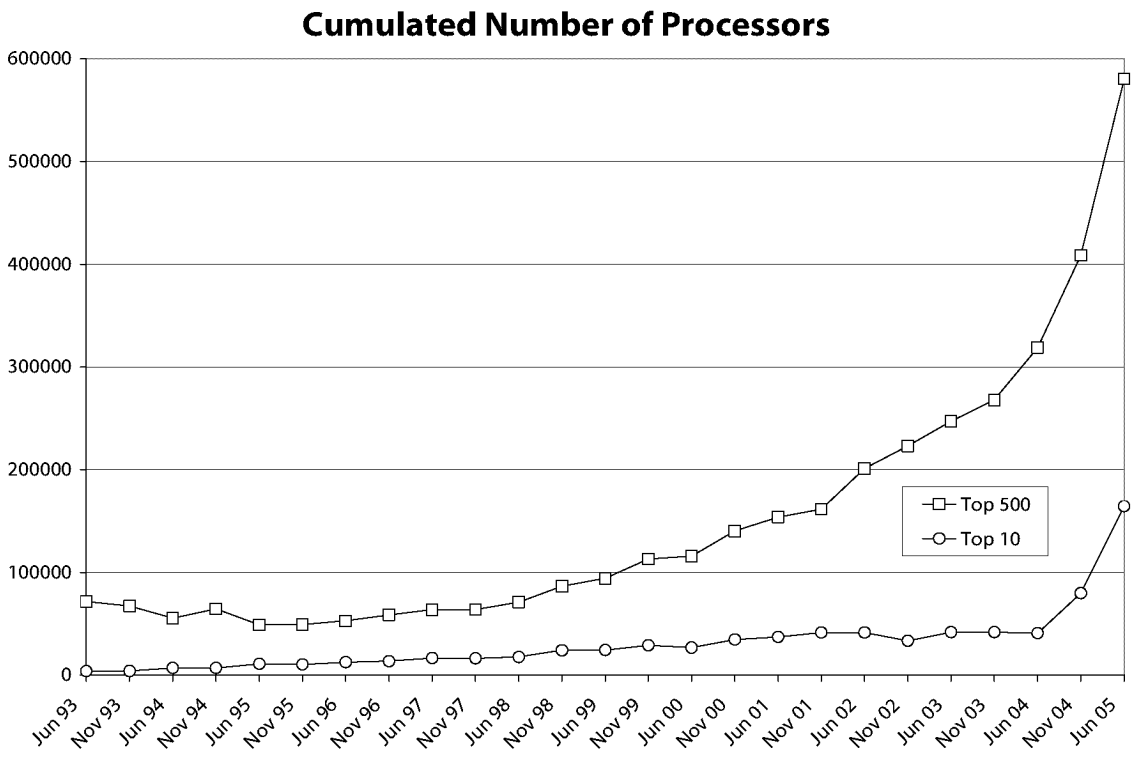


Figure 1.19: Cumulated number of processors in the top 10 and top 500 supercomputers

The current fastest supercomputer contains more than 65 000 processors. This will be topped by future systems. This high amount of processors, storage devices and connections poses many problems concerning scalability, reliability and manageability – and of course also in usability, since debugging or monitoring parallel applications with many thousand processes requires a set of very sophisticated tools.

1.3.6 Summary

What can be extracted from the given information about the Top500 list?

- Computation performance doubles every year.
- The number of clusters is increasing.
- More clusters will be found in the top 10 in the future.
- These clusters will integrate commodity off-the-shelf computers and components.
- PC manufacturers will build most of the supercomputer clusters.
- The number of processors will grow further by about 25% per year.

The good news is that supercomputers will get faster and bigger over the next few years, and that there is an open market for supercomputer manufacturers and customers. The bad news is that these supercomputers will integrate commodity off-the-shelf (COTS) components and there is not enough experience so far in creating such systems with many thousands of processors.

1.4 Commodity Supercomputing in a Nutshell

Every institution that owns several hundred computers wastes processing power to the value of a supercomputer: The idle cycles of the workstations standing on everybody's desk. In the late 1980's, new software allowed those wasted cycles to be caught, thus creating a parallel supercomputer. During the night, when the workstations are unused, large parallel applications were able to run. As long as the applications do not exchange too much data, these «network of workstations» (NOW) systems can achieve supercomputing performance as proven by various projects such as cryptographic attacks or event computations in nuclear accelerators [CERN].

Unfortunately, processes of parallel applications tend to communicate heavily. The users of the workstations also reduce the processing power, creating an unbalanced supercomputer that is only fully available during the night or at weekends. If dedicated computers are acquired for supercomputing only and a dedicated network for high-speed inter-process communication is used, such a system can achieve supercomputer performance easier and is always available. Such systems are called «clusters of workstations» (COW) systems and because of the similar structure, the same tools as in NOW systems can be used.

Using commodity off-the shelf computers, networks and many other parts is a strong trend in supercomputing today. Many supercomputers today are based mainly on commodity parts – custom technologies are only used, where a significant performance advantage can be detected.

Because the computers are put together in clusters and these clusters are used as a supercomputer, the community refers to such systems as **superclusters**.

1.4.1 Why Commodity Supercomputing?

There are several advantages to buying or building superclusters instead of supercomputers:

- Using many mass-production parts instead of some low-volume custom parts decreases the system price (economy of scale [Sei92]).
- Due to the standardization of commodity parts, the community using the same platform is larger and allows effort to be concentrated on increasing performance and decreasing cost.
- There are many ways to increase system performance (adding more nodes, replacing or upgrading the components etc.).
- The supercluster platform is similar or the same as the platform on the users' desks, allowing the users to reuse their utilization knowledge.
- The performance of such systems is high enough for the majority of applications.

Of course, there are also some problems and obstacles in creating superclusters:

- Commodity components were not created with supercomputing in mind, causing some nasty drawbacks such as lower reliability, scalability and manageability. In other words, superclusters are very hard to manage with currently existing management tools.
- Not every application scales with the processor count.
- Commodity products are not completely homogenous due to production tolerances, creating slightly slower and faster components and an unbalanced system.
- Superclusters are not as dense as supercomputers, thus requiring a huge amount of space.
- Installing a supercluster is a time-consuming task with many sources of mistakes.
- Commodity components have a limited time on the market and replacing a defective part could require an unscheduled upgrade of all like components in the whole supercluster.

Looking at the Top500 supercomputers list, it seems that the advantages outperform the disadvantages of building superclusters.

1.4.2 System Performance and System Size

In theory, superclusters can achieve infinite computing performance through an infinite number of processors. In practice, some limitations inhibit the building of such «towers of Babel»: Budget, technology, on-site resources, and software limit the system size today to about 10 000 nodes.

The current fastest ASCI supercomputer available (ASCI Q) offers a peak performance of 20.5 TFLOPS using 8192 processors. The LINPACK benchmark shows sustained performance of 13.9 TFLOPS. And the users desire even faster supercomputers to solve their computational problems: Hundreds or thousands of TFLOPS will be required over the next few decades – and technology must evolve quickly to provide the required performance.

Suppose a supercluster provides a peak performance of between 1 TFLOPS and 1 EFLOPS, and uses products available today or in the near future. How large will the whole supercluster be? What will its price, space and power requirements be?

The sample superclusters consist of the following two main components:

- **Node Computers:** Node computers contain one CPU providing a peak performance of 10 GFLOPS (4 pipelines with 2.5 GHz) for the price of US$ 2500. The CPU consumes 300 W of electrical power, 16 nodes are placed in one rack and it has an expected statistical lifetime of 5 years (MTBI).
- **System-Area Network:** The network switches build a 3D mesh with 10 nodes connected to one switch. The switch costs US$ 5 000, consumes 100 W of electrical power and 16 switches are placed in one rack.

The rack itself has a standing area of 1 m^2 with 150% additional overhead space required for access and cabling. Everything else (LAN, storage etc.) is not included.

Peak Performance	Number of Nodes	Space [m²]	Power [W]	Price [US$]	System MTBI
1 TFLOPS	100	20	31 k	300 k	18 days
10 TFLOPS	1 000	175	310 k	2 400 k	44 hrs
100 TFLOPS	10 000	1 720	3 100 k	19 M	263 min
1 PFLOPS	100 000	17 188	31 M	154 M	26 min
10 PFLOPS	1 000 000	171 875	310 M	1 229 M	158 sec
100 PFLOPS	10 000 000	1 718 750	3 100 M	10 B	16 sec
1 EFLOPS	100 000 000	17 187 500	31 G	79 B	1 sec

Table 1.5: Sample supercomputers of various sizes with current technology (rectangle)

The above table contains not only details of the system peak performance, but also the number of nodes, system price (US$), space ($m^2$), electrical power (W) and the expected mean time between interruptions (MTBI). The table adheres to the principles of «economy of scale», in that a ten times larger system is not ten times more expensive, but includes a 20% component price rebate.

The following observations can be made:

- The Swiss nuclear power plant Leibstadt produces 1.145 GW of electrical power [ATEL], which enables one 30 PFLOPS supercomputer to be powered. The distribution of this amount of energy is hard, but the evacuation of the same amount of heat energy is very difficult.
- One official soccer field (7 350 m^2) can accommodate a 500 TFLOPS supercomputer, one official basketball field (440 m^2) a 25 TFLOPS supercomputer. Distances play an important role in supercomputing, since every meter adds 3–5 ns latency.
- The budget of the Swiss Federation (2003: CHF 50 billion = US$ 33 billion) enables the acquisition of a supercomputer with a performance level of 350 PFLOPS.

The most disturbing part of this table is the mean time between interruptions (MTBI). A supercluster with 10 000 processors will face a broken node 9 times per day. A broken node usually requires a restart of the affected application. A huge application has a high risk of never finishing[2].

1.4.3 Supercluster Reference Architecture

A complete supercluster system contains some thousand or million components: Computers, network concentrators, data servers, disk arrays, workstations, racks, cables, power cords, operating system, application software, interfaces, and much more. The components are organized in subsystems on different layers:

[2] For the ASCI supercomputers, the Top500 benchmark often aborted because of hardware faults.

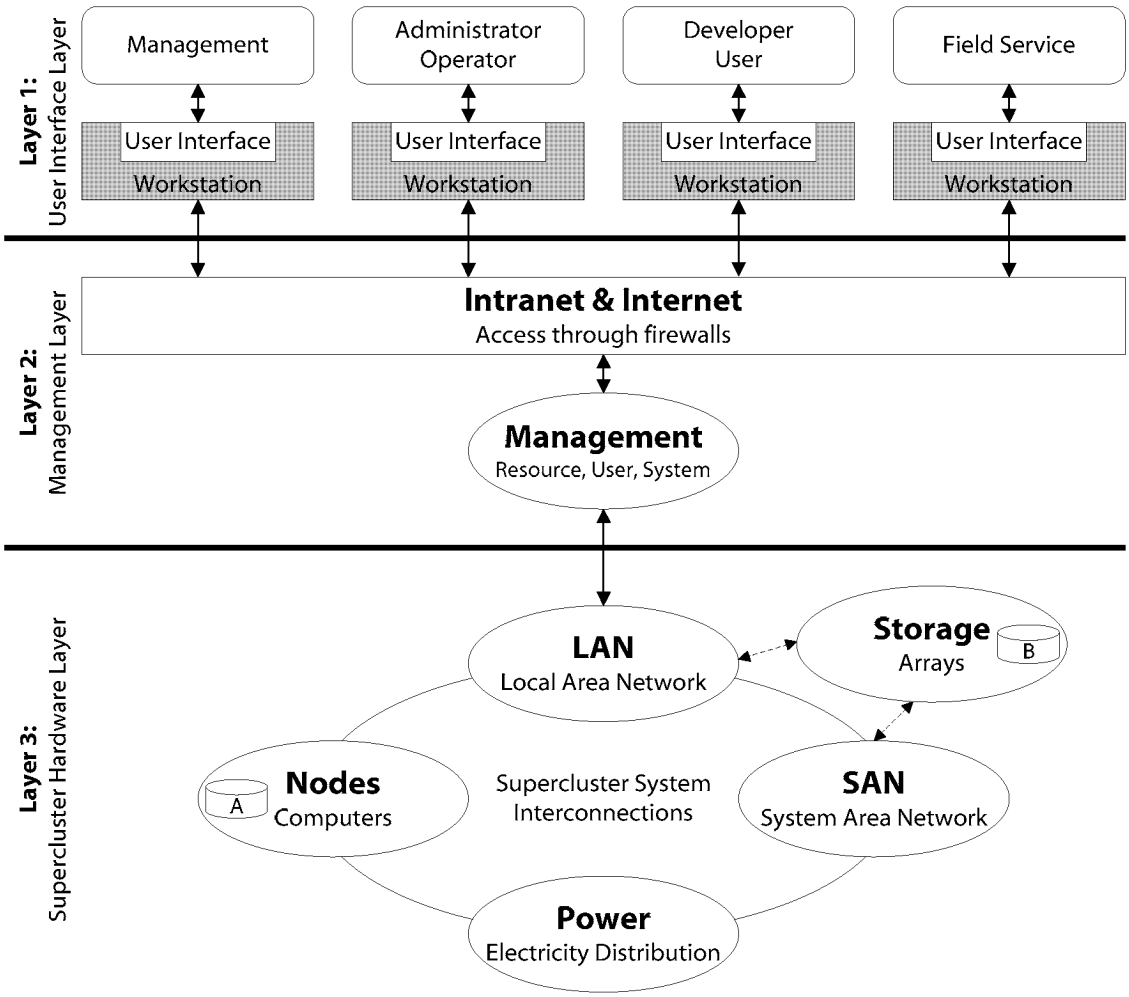


Figure 1.20: Reference architecture of a supercluster

1.4.3.1 Layer 1: User Interface Layer

The supercluster personnel access the supercluster through *User Interface Software* on their personal workstations or other agents such as mobile devices. The access is protected through security mechanisms (authenticity, integrity and encryption). This allows access to the supercluster almost from anywhere at maximum level security.

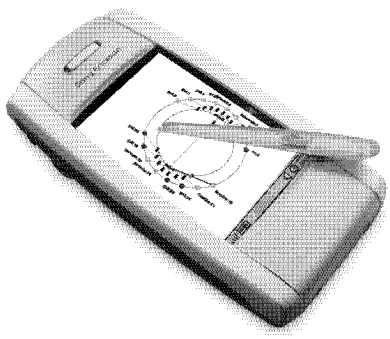


Figure 1.21: User interface application on a PDA system

1.4.3.2 Layer 2: Management and Access Layer

The supercluster is managed by one or more computers running the management software. They are connected through the *LAN Subsystem* to all components of the supercluster (it may also contain dedicated LAN components for management traffic only). Access to the data stored in the supercluster storage subsystem is also provided by these computers.

The management not only includes the mere hardware management, but also the management of the «soft» supercluster resources: Processing time, users, and applications.

1.4.3.3 Layer 3: Supercluster Hardware Layer with its Subsystems

The supercluster architecture is simplified when like components are placed in subsystems.

The **node subsystem** contains the node computers. These computers contain – besides the usual components – network interface adapters. They also contain multiple internal storage devices, used for the operating system, such as a scratch drive for temporary files, and additional drives if a distributed file system is used. Modern node computers for mounting in 19" racks are small – so-called «blade servers» allowing up to 280 nodes per 42U rack[3]. With this technology, a 1 PFLOPS system with 100 000 nodes can be stored in 4 000 racks.

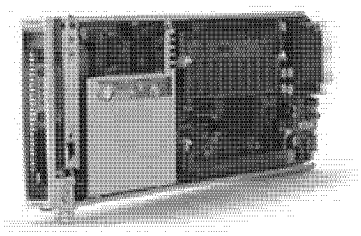


Figure 1.22: Blade server with small outline for high-density superclusters

The **SAN subsystem** interconnects all nodes and is used by parallel supercomputing applications. It consists of high-performance networking technology products such as switches, repeaters, protocol translators or partitioning enablers. The technology provides superior communication performance with high bandwidth and low latency.

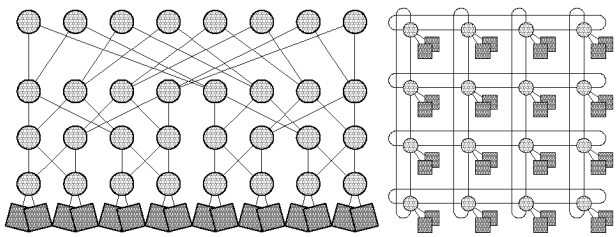


Figure 1.23: Examples of SAN topologies: Fat Tree on the left, 2D-Torus on the right; circles are SAN switches, boxes are nodes, lines are SAN cables

[3] 42U racks are 19" racks with 42 standardized height units (1U = 4.5 cm). Such a rack has the dimensions 60×100×200 cm.

The nodes and the SAN switches build a topology. The supercomputing center selects the topology with the best performance for the majority of the applications [Nem99].

The **LAN subsystem** consists of all network components that are used for standard socket- and OS-based communication systems such as FTP, NFS and others. It usually contains Ethernet hubs, switches, terminal servers and firewalls. It interconnects all nodes – and all other components connected to the LAN. To protect the system management traffic from usual LAN traffic, it is reasonable to use a separate network for management only. The LAN subsystem is not used for the communication of the parallel application, since the performance of current LAN technology (TCP/IP, MPICH) is too weak to achieve supercomputer performance.

The **power subsystem** is in charge of distributing electrical power to all components of the supercluster – no simple task because systems such as the ASCI White consume more electrical power (6.2 MW) than a small city.

The **storage subsystem** provides fast and reliable file and data storage for the users and supercluster nodes. There are two ways to provide this service (see Figure 1.20):

- The storage is provided by the nodes themselves (example A). Some of the node's storage is provided to a system-global file system in which each node participates.
- The storage is provided by dedicated servers with storage arrays (example B).

Because of the distributed nature of superclusters, the distributed storage approach (example A) seems to be the best match.

1.4.4 Superclusters Examples

The presented reference architecture requires illustration with existing superclusters. The two superclusters «Swiss-T1» (installed at the EPFL in Switzerland) and «ASCI Red» (installed at the SNL in the USA) show that this architecture holds.

1.4.4.1 «Swiss-T1» at EPFL

The «Swiss-T1» supercluster is installed at the center for parallel applications (CAPA) at the Swiss Federal Institute of Technology in Lausanne (EPFL), Switzerland [EPF00]. It is a small system consisting of 32 node computers with 2 processors each.

The following observations can be made when comparing the reference architecture with the «Swiss-T1»:

- The 32 node computers of the node subsystem are organized in 8 subclusters with 4 nodes and one SAN switch each.
- The eight SAN switches are interconnected in a 2-ring topology [KG99].
- The node computers are also interconnected using standard Ethernet technology.
- The storage subsystem with disk arrays is managed by the front-end computers.
- The management subsystem contains one computer which manages the SAN switches (management agent) and the front-end computers, which manage the nodes.

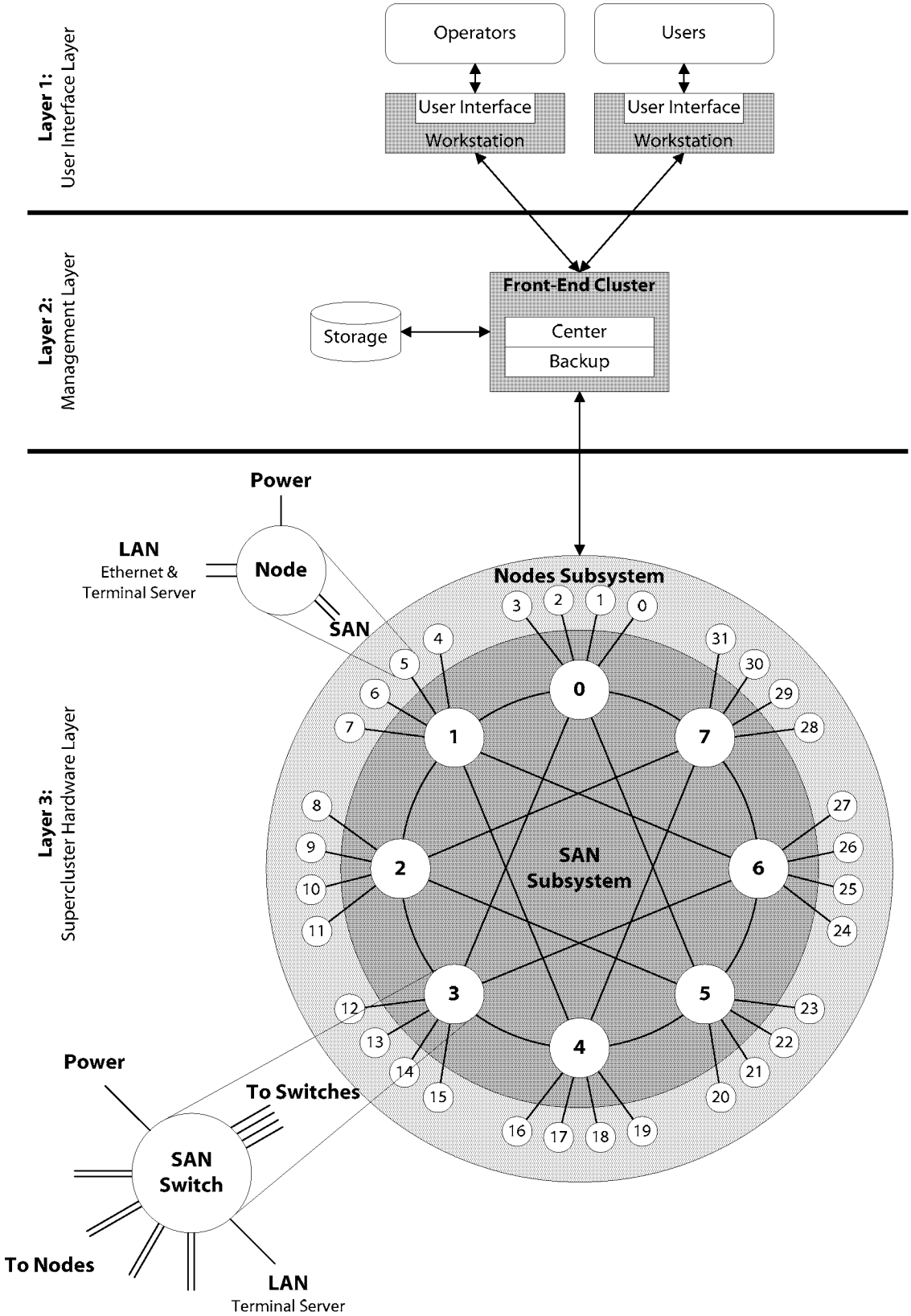


Figure 1.24: Structure of the Swiss-T1 supercluster

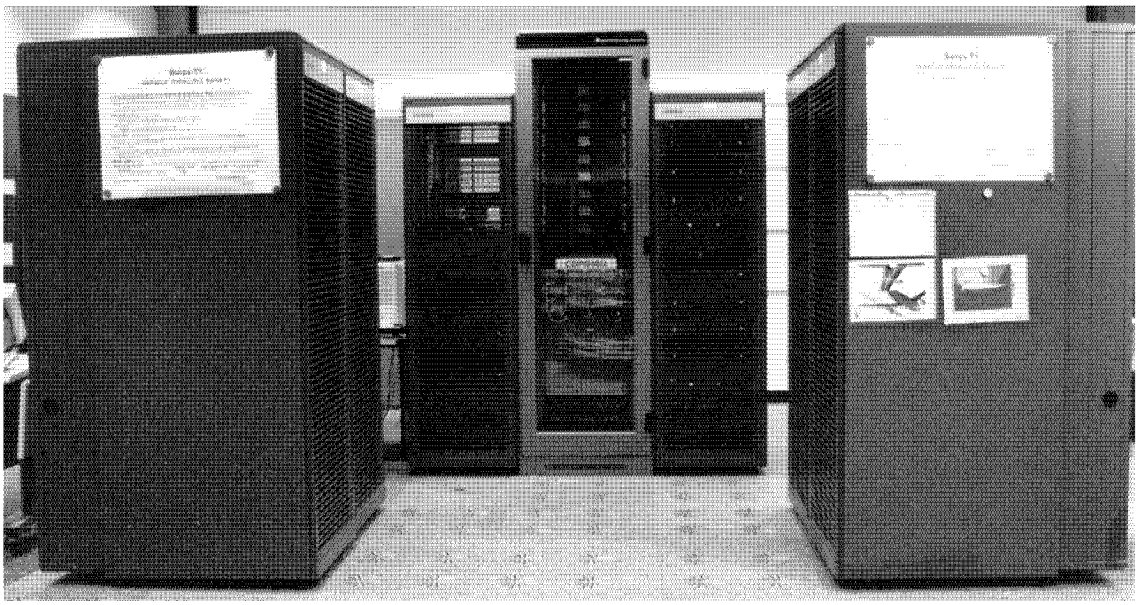


Figure 1.25: The Swiss-T1 supercluster installed at the EPFL in Switzerland

The presented supercluster reference architecture matches the Swiss-T1 architecture.

1.4.4.2 «ASCI Red» at SNL

The «ASCI Red» supercluster is installed at the Sandia National Laboratories in Albuquerque, USA [San96] [MH98]. It is a large system consisting of 9728 productive processors, organized in 4864 nodes. The computation nodes are organized in a 38×32×2 mesh network. There are additional nodes used for network management (12), disk I/O (73) and service tasks (52). Two nodes are used for system management.

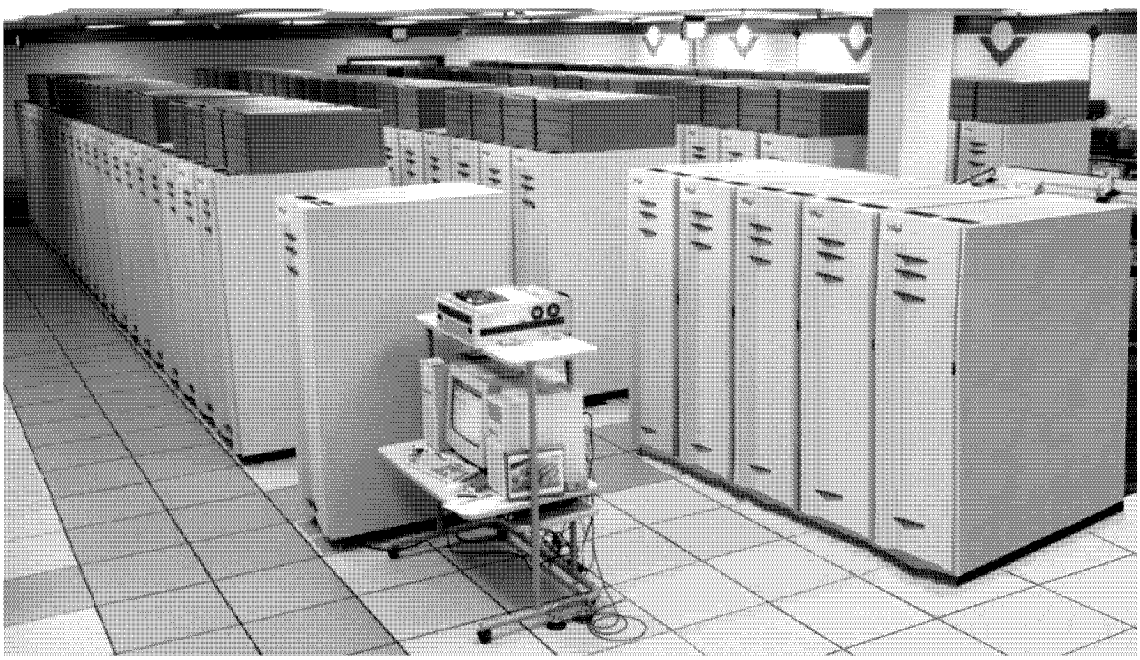


Figure 1.26: ASCI Red supercluster

Comparing the reference architecture with «ASCI Red», the following observations can be made:

- The 4864 nodes of the *Node Subsystem* are separated into 3 subclusters and the connections between the subclusters are plugged mechanically so that security personnel can check whether the red/black separation has been successful and no classified data can be seen outside.
- The *SAN Subsystem* contains 2432 switches of the high-performance network.
- The *LAN Subsystem* contains 12 nodes managing the Ethernet and ATM network.
- The *Storage Subsystem* contains 73 nodes, managing 20 cabinets with 12.5 TBytes disc capacity.
- The *Management Subsystem* contains two computers which manage the system. These computers are supported by management boards, each managing 16 nodes.

Although the *Management Layer* and *Supercluster Hardware Layer* are integrated into one layer, the tasks of the *Management Layer* are performed by dedicated computers. The reference architecture is simplified and holds, since the subsystems are present.

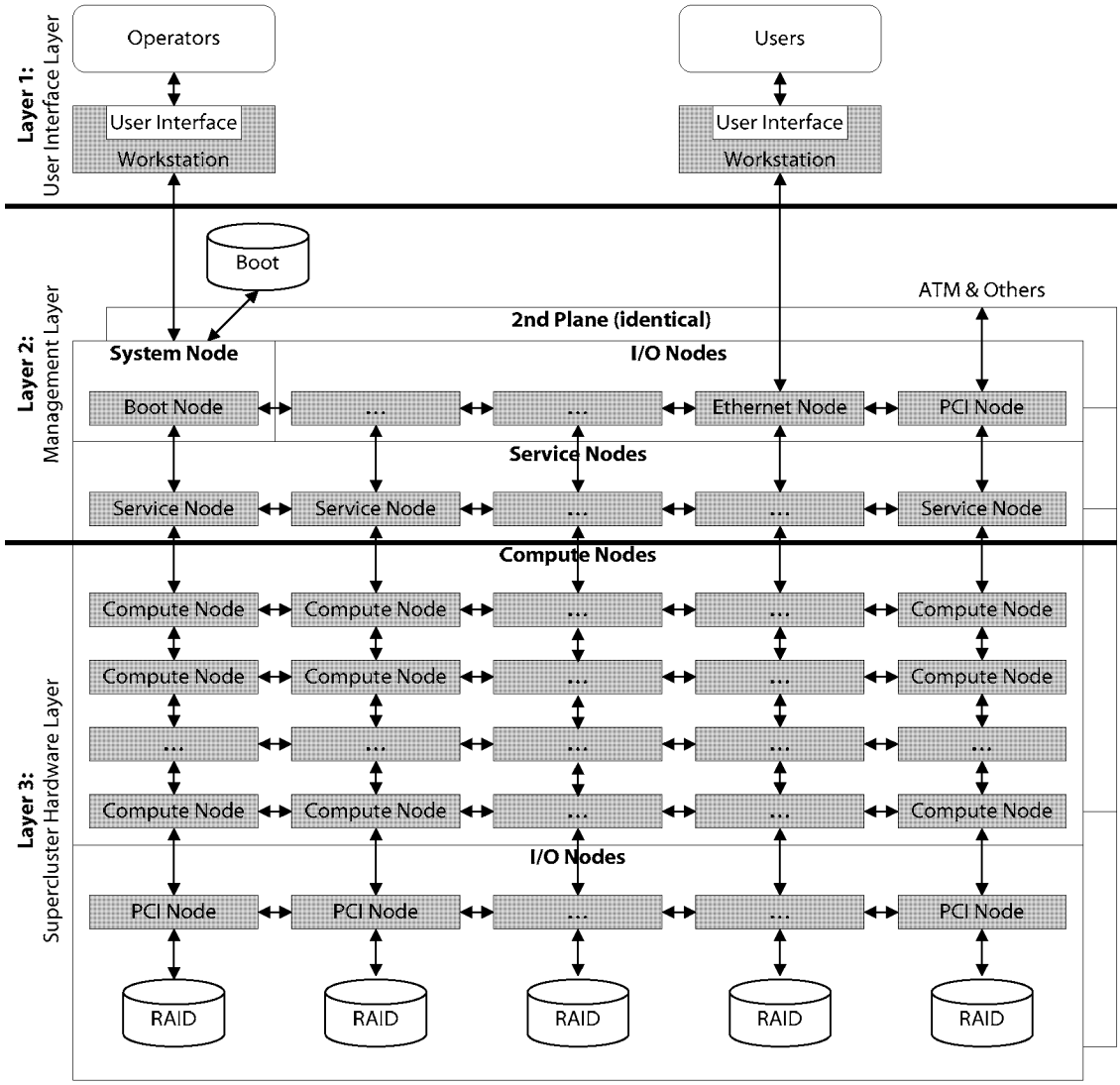


Figure 1.27: Structure of the ASCI Red supercluster

1.4.5 Four Basic Custom Supercluster Technologies

Not all commodity off-the-shelf parts are suited to supercomputing today. Whereas the performance of the nodes is high enough, the performance and skills of other products is too limited. Four basic non-commodity supercluster technologies are required:

- **High-speed System Area Network (SAN)** – because the available Ethernet-based communication is too weak for supercomputing.
- **Communication libraries using the SAN (e.g. MPI)** – because the available generic libraries do not take advantage of the high-speed network.
- **Distributed storage** – because centralized storage designs cannot scale.
- **Comprehensive system management** – because there are no satisfactory tools to control the whole supercluster.

These technologies are derived from the mainframe-type supercomputer technologies and are the subject of intensive research and development effort today.

1.4.5.1 High-Speed System Area Network (SAN)

The processes of a parallel application communicate with each other. After an iteration step, the results are broadcast to all other processes that need these results. This communication phase must be as short as possible, otherwise not only will the system performance be limited, but adding more processors will not increase, but decrease the performance (see Appendix A.1, especially Figure A.2 and A.3).

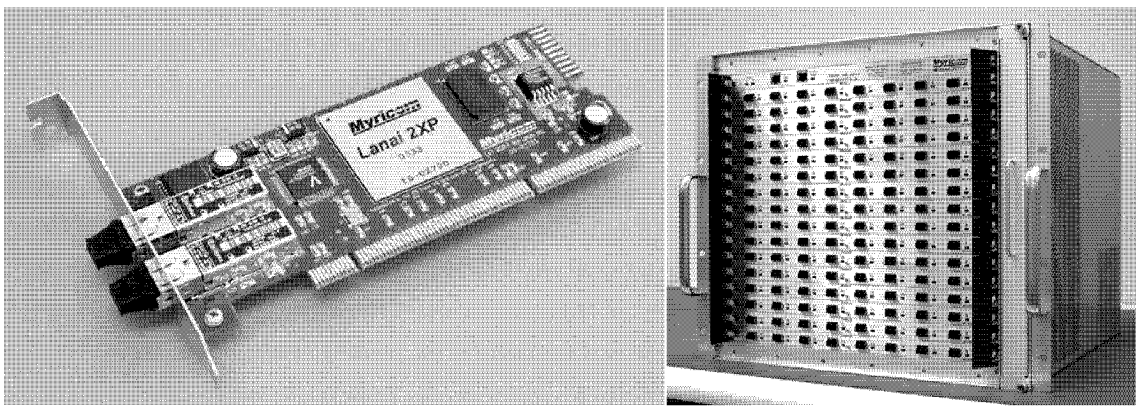


Figure 1.28: SAN hardware of Myricom (NIC left, switch right)

Besides simple data transfers, the hardware must also provide the following functionality that is necessary for efficient and scalable commodity supercomputing:

- **Multicast and Broadcast of Messages** – Processes of a parallel application can create groups and broadcast data within this group (and also broadcast data to the whole application). This is only efficiently possible, if the network supports multicasts. If it does not support multicasts, the message must be sent repeatedly by the sender to all recipients.
- **Barrier and Event Synchronization** – For the synchronization of all processes of an application, two models are used. The barrier synchronization is used if the application needs to wait until all processes have finished their tasks and are ready to

continue with the next task. The event synchronization is used if the one process wants to inform all the other processes that some condition has been reached. Especially barrier synchronization scales poorly without hardware support.

- **Key Management for Mutual Exclusion** – For transactions using shared resources and critical sections in applications, mutual exclusion of processes is needed. The resource can only be used or the critical section can only be entered, if the process owns the corresponding key. All other processes must wait until the process returns the key and a mechanism selects the next key owner. This functionality is hard to implement without hardware support.
- **Adaptive Routing** – If the message cannot be sent through the correct SAN link (broken or stalled), the message is sent through alternative paths. If there is only one link which may be used, the applications will stall and the whole system must be restarted.
- **Intelligent and efficient MPI Messaging** – MPI has various message sending mechanisms that require smart hardware support. To prevent costly application redesign (for using the most efficient mechanisms of the hardware), the most commonly used mechanisms must be supported.

Only a well designed SAN hardware (with appropriate MPI support) with high message bandwidth (of some GBytes/s), low latency (some µs overhead) and the mentioned features, allows the adding of nodes for increasing the system performance.

Some communication models and network technologies were designed in the author's project. The results are summarized in the dissertation of Martin Lienhard [Lie00].

1.4.5.2 Efficient Messaging Libraries (MPI)

The Message Passing Interface (MPI) is the standard for using parallel supercomputers [MPI95] [MPI97]. It contains calls for sending and receiving messages, creating groups, synchronization, storage access and data I/O.

The applications use MPI directly or indirectly, using mathematical libraries that are based on MPI. MPI needs to be ported on a particular platform, supporting the many features that are implemented in the hardware. It is an art to design and port an MPI implementation on a platform that takes advantage of the hardware features and optimizes those MPI calls that are used by the applications and libraries.

MPI was ported to the network platforms that were designed within the project. Hardware and software were closely coupled and highly efficient. The results of this work are summarized in the dissertation of Stephan Brauss [Bra00].

1.4.5.3 Distributed Storage

Supercomputers with many thousand nodes require large storage systems. Centralized storage servers with disk arrays may not match the performance requirements, since the network to these servers is the system bottleneck.

The nodes contain storage devices and can provide some space to the whole system. All they need is software that creates a virtual global file system. A system-wide RAID with

fail-over mechanisms is the goal of many developments at companies and universities, providing multi-terabyte storage space with an unmatched access and transfer performance [CPQPF].

1.4.5.4 Comprehensive System Management

To replace supercomputers with superclusters, the supercluster with all its components (from a few dozen up to several thousand) must be manageable. The existing systems use a combination of shell scripts and several tools, but this approach is not very efficient and effective for small systems – and big systems are almost impossible to manage without specialized software and dedicated hardware.

The author was responsible for designing the system management software for the created superclusters within his project. This dissertation summarizes the results.

1.5 Future of Supercomputing

Since there will be only one major technology for building computers, the computing universe will only have the segments of servers and workstations. The lowermost segment will be the appliance systems – downsized computers for specific usage such as Internet terminals and PDAs.

Category	Selling Price	Typical Manufacturers
«Servers»	20 k$	Compaq/HP, IBM, Intel, SUN
«Workstations»	2 k$	Compaq/HP, IBM, Intel, SUN
Appliance Systems	200 $	Many different manufacturers

Table 1.6: Computer segmentation in the near future

The separation into servers and workstations is artificial, since the basic technology is the same: Both systems have one or multiple CPUs connected to the memory in SMP or NUMA style and have basically the same in-system interconnections and components. The only difference is that servers move data more efficiently and are highly reliable – with redundant subsystems and modular components that can be replaced while the system is running – and workstations are highly interactive with powerful visualization subsystems and operating systems.

The supercomputer segment with its classical full-custom or partially-custom mainframe designs will have (almost) disappeared – the systems that will provide supercomputing performance are clusters of standard computers with off-the-shelf components.

In short, supercomputers of the future will contain nothing but interconnected, clustered standard single-CPU computers: Microprocessors, memories, storage, operating systems and everything else will be taken off the shelves of computer stores.

1.6 The Gap: System Management

In the past, supercomputers were huge boxes that were installed in a room. They had a power button and the management application ran on the system itself or on an associated box next to it. Managing a supercomputer was almost as easy as managing a workstation on the desk.

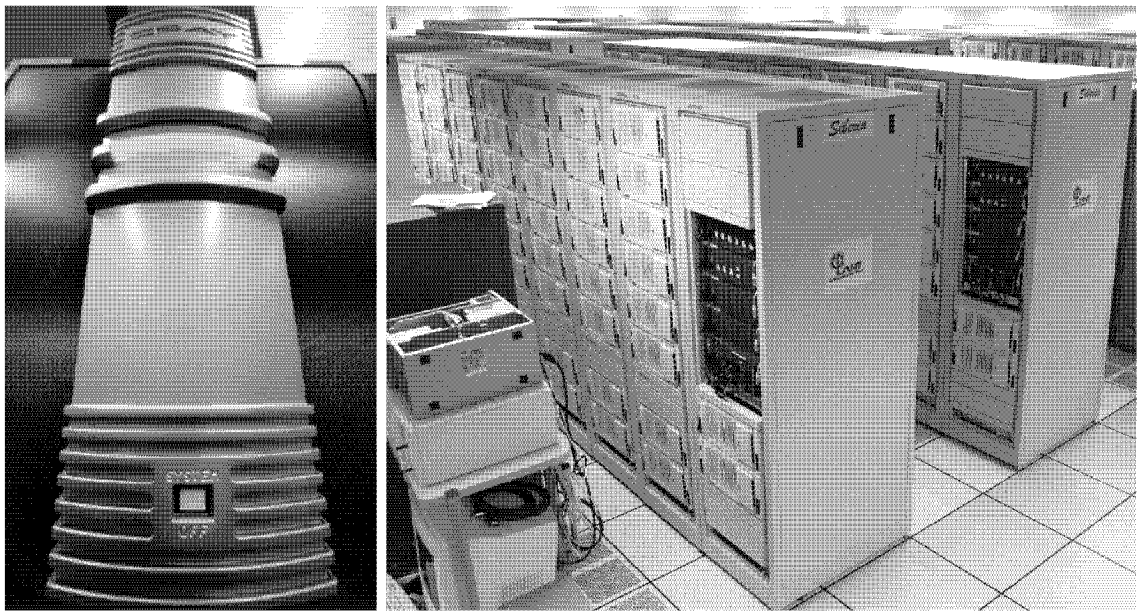


Figure 1.29: The power switch of the Cray J90 supercomputer (left) and the C-Plant supercluster with 1 800 power switches (right)

Today, supercomputers consist of many hundred racks installed in huge rooms. Managing a supercomputer has the same complexity as managing employees of a huge company.

Many research teams are investing a lot of time and money in high-speed network technology, efficient message passing interfaces, and parallel file systems. The performance of the products can be measured in MBytes per second and microseconds. The results are proudly presented in conferences, papers and dissertations, and compared to the results of others.

System management was considered to be consuming resources that would be better invested in increasing performance and functionality, and decreasing latency and cost. Supercomputer management today is a patchwork of small and usually self-made tools that try to fix a usability gap created by the integration of various products within the supercluster. It is a research orphan, considered to be boring, complex to implement, with goals that are hard to describe, and performance that cannot be measured. Management is fuzzy and therefore not beloved by engineers.

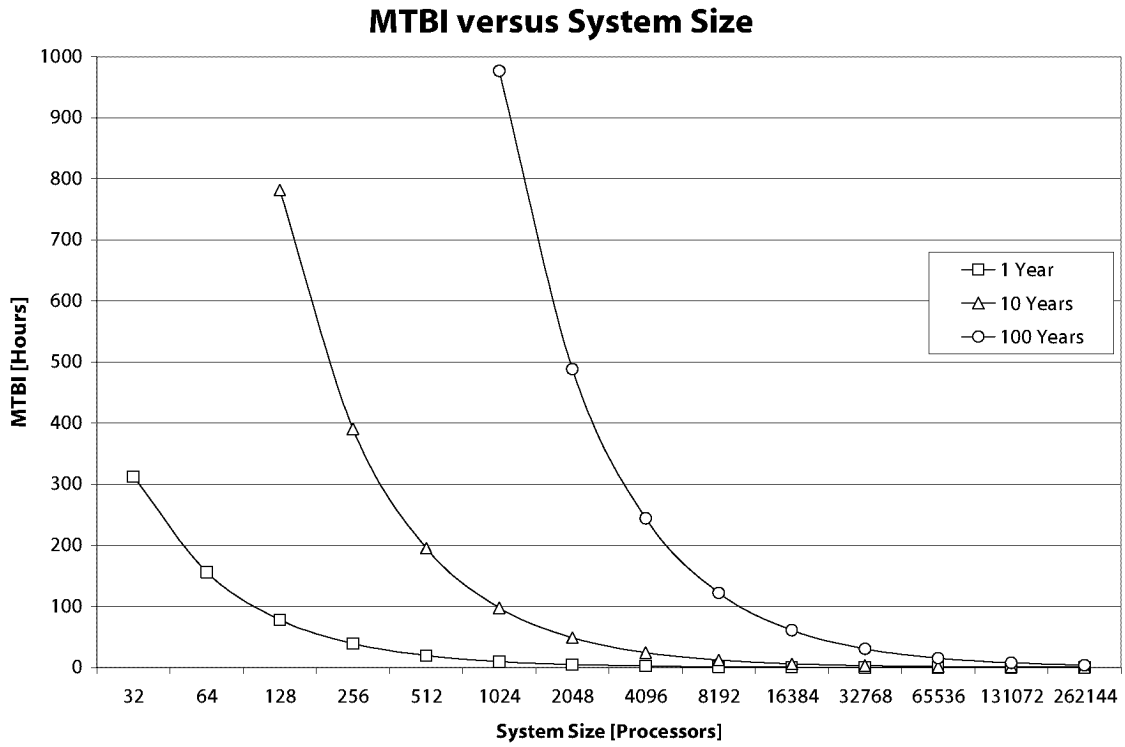


Figure 1.30: MTBI (Mean Time Between Interruptions) in hours versus system size (processors) for various component lifetimes (1, 10 and 100 years)

It is hard to understand why there is no integrated system management software available today – this thesis is the first work in this direction. Systems that deliver hundreds of TFLOPS or several PFLOPS will contain between 100 000 and one million CPUs that cannot be handled without computer-aided management, since one single minute downtime will cost US$ 58[4]. Ironically, managing top-edge supercomputers of the future will require a small supercomputer itself.

Reading the reports of the supercomputing centers that own the fastest supercomputers of the world, it can be clearly seen that the management facilities cannot keep up with the increase of system performance and size. Supercomputing management is an unsolved matter that urgently needs research. Many facts and reports show that action must be taken immediately:

- The management software that was created for the old supercomputers cannot be used anymore because the superclusters have different system architectures.
- Managing each node individually with the supplied management tools is a never-ending task if many computers must be managed. Using one additional tool for every additional component type makes system administration impossible.
- The ASCI Q supercomputer, installed at Los Alamos National Laboratories, has a MTBI of 8 hours and about 110 failures every month. Each failure blocks the system and the affected applications must be restarted [Mor03].

[4] The US$ 58 per minute is a calculated value under the assumption, that the supercomputer will depreciate within 5 years and has a price of US$ 154 million, not taking into account the cost of space, power and personnel.

- Global commands (start-up, shut-down, update with patches) take too much time and are unreliable, monitoring the system breaks bandwidth limits quickly, and administration is very hard for huge systems [Gon03].
- System management is identified as one of the important problems – if not in fact the most important problem of all – of commodity supercomputing. The ASCI laboratories want to build a team that searches for concepts of system management and develops «supercomputing enabling software».

Integrated system management software is therefore the most important issue both today and in the future. Its job is the following:

- Keep the supercluster available to the users as much as possible.
- Detect failures as soon as possible and handle them automatically.
- Provide the system status and log to the administrators.
- Support the developers in optimizing and debugging their applications.
- Protect the supercomputer from intruders, malicious codes and users.
- Allow accounting and usage analysis.

The goal of this dissertation was to take the first step toward integrated management of commodity parts-based supercomputers, and to search for an optimal computer-aided software solution for comprehensive, effective, efficient, scalable and reliable system management of commodity supercomputers of today and the near future.

2 Management

Power is nothing without control.

Slogan of Pirelli

Supercomputer users are creating applications, which are consuming resources. These resources are provided by the supercomputer hard- and software. All involved partners (users, applications, resources, hardware, and software) are part of a system. It is the management's job to keep this system running.

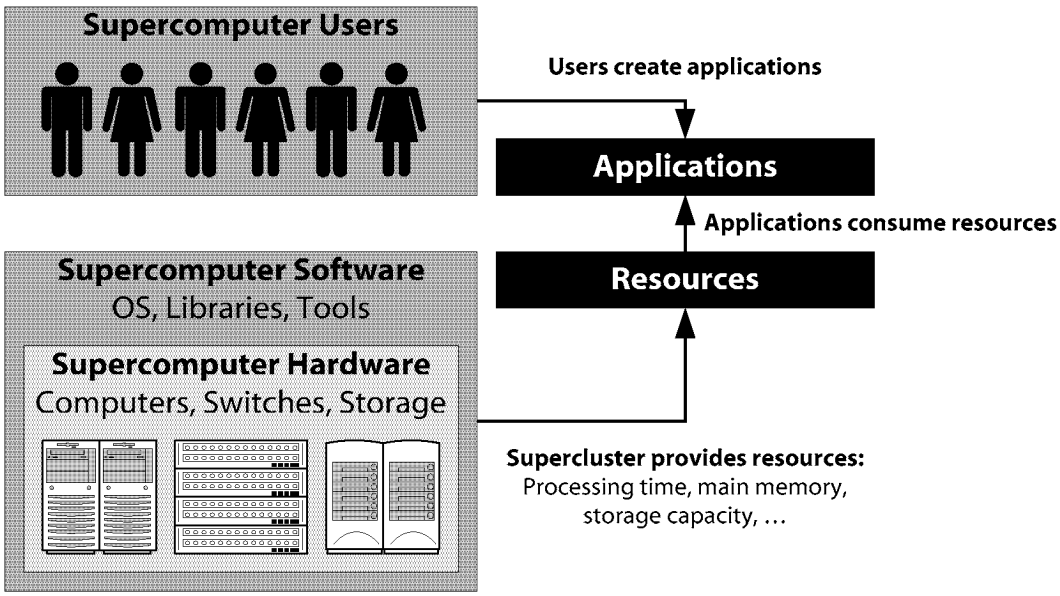


Figure 2.1: Supercomputer system – the management keeps it running

It can be seen that supercomputer system management includes:

- Management of the supercomputer hardware and software.
- Management of the supercomputer users.
- Management of the applications and resources.

This chapter is the «problem statement» of this dissertation and explains, what «computer-aided supercluster management» means. It introduces the users with their expectations, fears, and requirements. It shows that comprehensive system management also includes the planning and installation process. It lists the functionality that is required for system management. And, finally, it highlights the bottlenecks and problems that make system management difficult.

2.1 Supercluster People

There are four types of people around superclusters:

- Those who use superclusters: The users.
- Those who buy superclusters: The managers.
- Those who sell superclusters: The sales people.
- Those who keep the superclusters running: The administrators and field service.

The field service and the sales people are employees of manufacturers. All others work at the supercomputer center where the supercluster is installed.

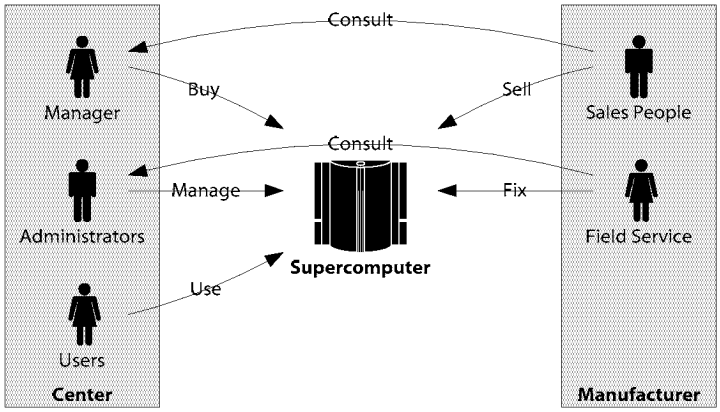


Figure 2.2: Persons involved in supercomputer usage

This section shows how they work, what they want, and what they are afraid of.

2.1.1 Supercomputer Center Manager

The manager is responsible for all financial matters. He buys the supercomputer and he wants to be sure that the selected system always operates at maximum performance. No matter what the management software looks like, he requires the following functionality:

- He wants to buy the fastest supercomputer he can get for his money. He needs software that helps him to find the fastest system for the applications his users will run on the supercomputer.
- During operation, he wants to check the utilization and performance and he wants to know why the system is not running at 100% and what he can do better when buying the next supercomputer or upgrading the current system. For this, he needs software support.
- If a new supercomputer is installed, the previous model is removed. During that time, the users cannot run their applications. The manager wants to keep the installation time as short as possible. With many thousand cables, this requires software support.

There are three things he is afraid of:

- He does not want to install a system that – because of the system architecture and the selected components – is not capable of delivering the performance he needs.
- He does not want to use a supercomputer without quantitative feedback about its performance and utilization. Management without reliable figures is impossible.
- He does not want to have a lot of technicians running around during installation (and later operation), searching for badly placed cables, wrong configurations and defective parts.

The manager's requirements suggest three main management applications:

- Design and simulation software that outputs the estimated performance of a supercluster design created by the manager itself.
- Installation software that shows whether the supercluster has been installed correctly in accordance with the previously created design or what needs to be changed.
- Operation software that permanently monitors the utilization and performance and summarizes it into a database and printable reports.

Since the manager is responsible for a multi-million dollar investment, he wants management software that proves to him that he has made correct decisions.

2.1.2 Supercomputer Users

The users develop applications and execute them on the supercomputer. They want to have the fastest system available for their applications so they can attain results very quickly. They have the following requirements:

- They want a supercomputer that executes their applications with the highest possible performance. They want to be able to tune the system and their applications to reach this goal. In academic environments, the users want to turn the results into published papers and reports.
- They want a fair scheduling system where every user, project, and department can be sure that their applications are run within a reasonable timeframe.
- They want the large applications using all processors for a long time to have a high probability of terminating successfully.
- They want debugging and profiling access to all processes of an application.
- They want a supercomputer that efficiently supports the most important functions of the main programming libraries (such as MPI or LINPACK).

There are four things they are afraid of:

- Applications are slow because of the supercomputer or application design.
- The applications are stuck in a queue and are not scheduled to start anytime soon, or the application is so large that hardware faults require permanent restarts.
- The system does not support debugging (debugger access, message tracing) which makes development and tuning of applications very slow.
- The applications need to be re-designed because the programming libraries are either incomplete or the most often used functions are inefficiently implemented.

The users' requirements suggest further features for main management applications:

- For the design and simulation of supercomputer architecture, the application (communication pattern between processes, instruction and MPI call mix) must be respected. This prevents buying supercomputers that are unable to provide the required performance.
- The libraries must be either complete and the functions efficiently implemented, or the speed-up using other functions or new libraries must be so high, that the users are motivated to re-design their applications.

The users work every day with the supercomputer. They require the management software to simplify their work instead of making it more complex.

2.1.3 Administrators

The administrators keep the supercomputer running. They want to keep utilization high and downtime low. They have the following requirements:

- They want the system to automatically detect all faults and resolve them autonomously without manual intervention. They also want the system to anticipate faults and show which components need replacement as soon as possible.
- They want the management software to provide a one-system view, where management actions are quickly broadcast and executed. They also want access to every component individually (or in groups) for management treatments.
- They want to be sure that the whole system is correctly configured, especially after installation or downtimes before the supercomputer is re-opened for the users.
- They want the system to protect users, applications, software and hardware.

There are four things they are afraid of:

- Applications, software or hardware faults must be detected and resolved manually. These faults block the whole system, and waste many processing hours.
- Every single component must be managed individually, leading to many hours or days of manual interactive work with many sources for tricky errors.
- It is impossible to detect whether one or more components has been badly configured or some cables have been wrongly placed. These mistakes leave the whole system blocked and unusable.
- Malicious users and applications, as well as interactions between users, applications, software and hardware cause unwanted side-effects with low performance and utilization.

The administrators' requirements suggest the following features for main management applications:

- Monitoring and fault detection is essential for the operation. Monitoring allows – besides measuring the performance – the detection of parts that will break soon. Fault detection reduces the time where parts block the system's operation. Both features increase utilization and decrease downtime frequency and duration.

- Management actions to single, multiple or all components of the supercluster must be transported reliably and efficiently.
- Monitoring also detects wrong configurations and badly placed connections when using smart diagnostic software that is part of the management.

The management software must make the supercomputer easy to handle. Administrative staff make operation expensive and bad management software requires many administrators.

2.1.4 Sales People

The sales people sell either the whole supercomputer (as «integrator») or parts of it (single components or whole subsystems). They are well informed about component performance, compatibility, price, reliability and availability. They want the center manager to be confident with the system, so they consult him regarding his decisions and evaluations, and they suggest upgrades as soon as they are available and reasonable. They also advise the supercomputer users about how they can improve the performance of their applications.

With regard to the management software, they have the following requirements:

- The management software either supports their product directly or makes it easy to integrate the product's management software into the management system.
- The management software supports the product's features allowing the product to operate with the highest possible performance. The management software must also support the applications in using those features.
- The sales people want to know why the products do not deliver maximum performance and the management software has to provide this information.

There are four things that they are afraid of:

- Their products are not supported by the management software or the product's management software cannot be integrated into the system's management software.
- The key features for the product's superior performance are not fully supported by the management software, resulting in poor performance.
- It is impossible to find out why products perform slowly or unreliably.
- The users, administrators and the center manager are not confident about the performance. This results in no further purchases, bad reviews and bad reputation.

The sales people's requirements suggest the following features for main management applications:

- The specifications of the products used must be included in the design and simulation process in order to find optimal products for the supercomputer.
- The supercluster management software must have an open interface that allows the integration of the products' management software or its management features.
- The product's management software must provide current and historical monitoring information in order to find out how the product could perform better.

The sales people want to sell their products. They want to be sure that they advised the managers correctly and sold the right product. The management software must prove to him that the products perform as promised and that he gave good and accurate advice.

2.1.5 Field Service Personnel

The field service personnel are on-site during installation and downtimes. They configure the products for optimal operation and fix problems. They regularly check the performance and suggest replacement during the next downtime. Their requirements are as follows:

- The management software allows them to access each component individually for configuration and monitoring. The log files store any event that might help in finding problems.
- The supercomputer design file allows them to generate configuration files for all products.
- The management software increases the system availability and the field service personnel are only on-site for «scheduled» downtimes (e.g. once a week).

There are three things they are afraid of:

- Configuration and monitoring access to each part is complicated. Additionally, the management application only allows access to one part at a time.
- There are no tools available that create configuration data automatically.
- «Unscheduled» downtimes, created by unexpected fatal faults require immediate on-site presence by the field service personnel. Either more personnel are acquired that make the product more expensive, or the reaction time is massively increased thus decreasing usage and performance of the supercomputer. Both make the customers angry.

The field service personnel's requirements suggest the following features for main management applications:

- The design files allow automated configuration generation for each product.
- The configuration data can be easily distributed and applied to all products.
- Special configuration data for diagnostic treatments can easily be applied to one single part or a group of parts.
- Reliability features of the management software reduce unscheduled downtimes.

The field service personnel prevent and fix problems. They expect the management software to prevent and fix problems autonomously, making his presence only necessary in complex cases.

2.2 Supercluster Lifecycle

The previously presented requirements of the supercluster people and the nature of the superclusters themselves show, that management is more than «a tool» that keeps an existing supercomputer «alive». There are some tasks that require software support before the supercomputer is ready to use. These tasks depend on each other, and lead to the following «supercluster lifecycle».

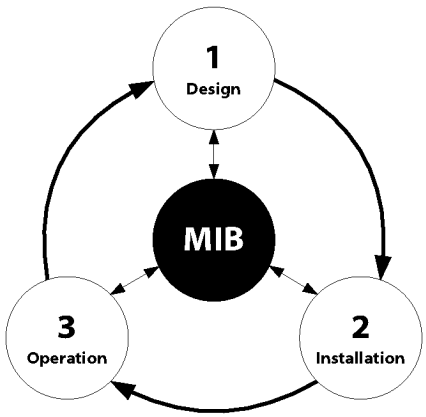


Figure 2.3: Supercluster Lifecycle

The supercluster lifecycle has three phases and a common database, the «Management Information Base» (MIB):

- The **design phase**, where various system architectures and components are evaluated. The performance of these systems is estimated and the best design is selected. The design description and estimated performance is stored in the MIB.
- The **installation phase**, where the virtual system design description is translated into physical construction plans and configuration data for all components. This basic configuration data is stored in the MIB.
- The **operation phase**, where the supercluster is managed.

The nature of superclusters allows three ways to increase performance:

- Extending the supercluster with the same components as already used.
- Upgrading one subsystem's components with other products.
- Creating a new supercluster with re-used components of previous superclusters.

This explains why there is a link between the operation and design phase: The supercluster is not a project with a start and an end, but is a cycle.

Describing the whole lifecycle with all its possible aspects would be enough work for a person's entire professional life or many dozen dissertations by smart researchers. This thesis therefore can only cover the aspects of one single phase: The operational phase. The other phases are described briefly for the completeness of this section.

2.2.1 Management Information Base (MIB)

The management information base (MIB) is the central part of the supercluster lifecycle. Each phase creates data which is stored in the MIB and data is consumed from this MIB as well as external sources. This MIB basically contains the following types of data:

- **Supercluster design data** – The supercluster design data contains the supercluster architecture. This includes system size, placement of components, and expected performance. This data is created in the design phase and remains static until the next design phase.
- **Performance simulation data** – For main applications, this is the estimated performance for different sizes. This allows comparison of the effective performance with the estimated data, which was used as a decision base for the system purchase.
- **Basic configuration data** – All components must be identified and require a basic configuration, which is downloaded to the components during start-up and reset.
- **Temporary configuration data** – Components receive temporary configuration changes caused by faults or application requests. These changes are undone after the defective component has been replaced or the application has finished.
- **Log file data** – The log file contains details of events that are worth being registered, such as triggered management tasks, failures or security-relevant data such as attacks, resource misuse or unexpected hardware changes.
- **Monitoring data** – The components are permanently observed and this data needs to be stored in a sensible way, since there is a huge amount of data that needs to be synchronized («global heartbeat») with context («which traffic is caused by which application»).
- **Accounting data** – The resource usage that is charged to the causative users.
- **User profile data** – The people allowed to use and manage the system are defined, together with their permissions and resource quotas.

A sample MIB is presented in Appendix B.

2.2.2 Design Phase

This phase has one goal: Finding the supercluster design that satisfies all needs.

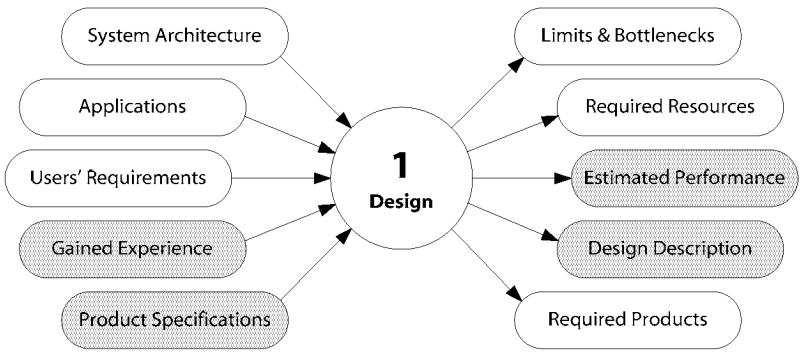


Figure 2.4: Design phase – requirements (left), outcome (right), MIB data (grey)

Designing a supercluster is creative work: All requirements are concentrated into a variety of possible supercluster designs. Simulations of the application on these designs

(as well as conditions such as available space and money) allow the best design to be selected for implementation.

The requirements are quickly listed:

- System architecture: System size, network topology.
- Applications: Instruction and messaging call mix, communication patterns.
- Requirements: System performance, available libraries, management capabilities.
- Gained experience: Which products, topology, size, libraries.
- Product specifications: Effective performance and capabilities of products.

The output of this phase is as follows:

- Limits & bottlenecks: Performance limits defined by architecture and products.
- Required resources: System cost, size, power consumption, etc.
- Estimated performance: Performance for each application in various sizes, based on the methods presented in Appendix A.
- Design description: Data that describes the supercluster design.
- Required products: Amount and type of products needed to build the supercluster.

The outcome of this phase can be tabled as follows:

	Criteria	System A	System B	System C
SAN	Topology Type	4D Torus	3D Mesh	Fat Tree
	Topology Size (Switches)	8 × 8 × 8 × 8	16 × 16 × 16	5 × 4 096
	Switch Count	4 096	4 096	20 480
	Nodes per Switch	8	8	8
	Link Count	49 152	44 288	163 840
	Bandwidth (bidirectional)	2 GBytes/s	2 GBytes/s	2 GBytes/s
Nodes	Node Performance	10 GFLOPS	10 GFLOPS	10 GFLOPS
	Node Count	32 768	32 768	32 768
	SAN NICs per Node	1	1	1
Limits & Bottlenecks	System Peak Performance	328 TFLOPS	328 TFLOPS	328 TFLOPS
	Bisectional Bandwidth	2 048 GBytes/s	512 GBytes/s	32 768 GBytes/s
	Average Bandwidth	125 MBytes/s	44 MBytes/s	1 000 MBytes/s
	System Cost	1 097 M$	1 097 M$	1 283 M$
	System Size	14 132 m^2	14 132 m^2	18 228 m^2
	Power Consumption	20 MW	20 MW	22 MW
Performance	Application A, 1024 Nodes	70%	70%	70%
	Application A, all Nodes	55%	50%	65%
	Application B, 1024 Nodes	55%	50%	60%
	Application B, all Nodes	50%	50%	50%
	Application C, 1024 Nodes	70%	70%	70%
	Application C, all Nodes	40%	40%	70%

Table 2.1: Sample design comparison table

After this phase, one design which is thoroughly known with its expected performance, all advantages and disadvantages can be selected. The design is stored in the MIB, together with details of its expected performance which can later be compared with its effective performance.

2.2.3 Installation Phase

This phase has one goal: Installing the supercluster according to the design previously selected.

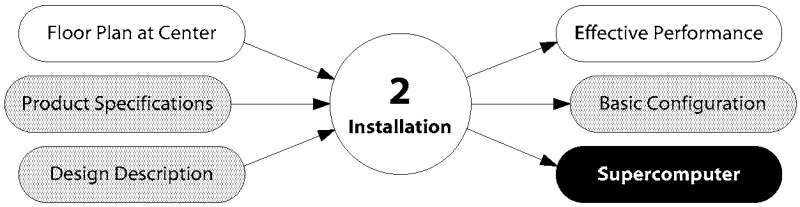


Figure 2.5: Installation phase – input (left), output (right), MIB data (grey)

The installation phase is the organizational and physical work: Out of the design files, a floor plan is drawn, a shopping list created, and configuration data generated. The parts are ordered, checked, assembled, configured and the whole system is tested and tuned [SGL03] [SSB+99]. At the end of the phase, benchmark applications are used to verify the performance estimations.

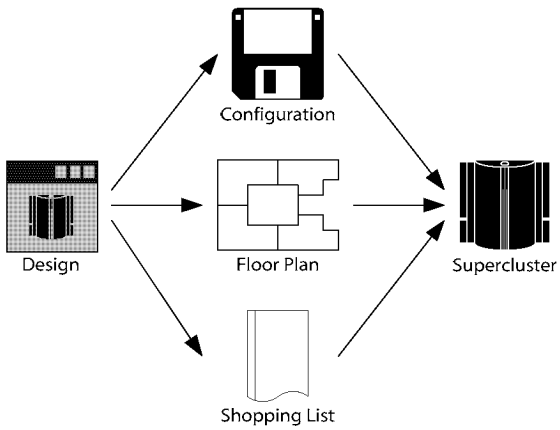


Figure 2.6: Installation phase – from the virtual design to the physical supercluster

The termination of the installation phase is normally celebrated with an inauguration day with supercomputing center staff, manufacturer representatives, users, developers, researchers and journalists. It is the official birthday of the supercluster.

2.2.4 Operational Phase

This phase has one goal: Maximize the usage of the installed supercluster.

Using management software, the administrators observe and control the system, the users their applications, the field service their products, and the center manager gets the data he needs.

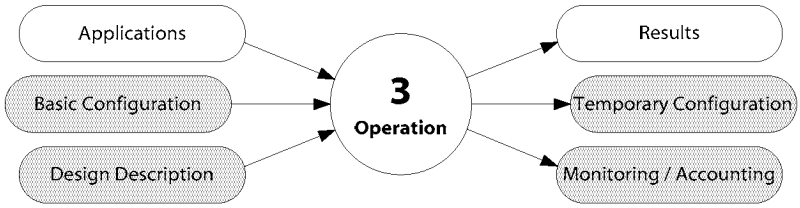


Figure 2.7: Operational phase – input (left), output (right), MIB data (grey)

This dissertation presents the software that is used in this phase.

2.3 Supercluster Management

Describing the functionality of the management software for the operational phase quickly becomes a bucket full of puzzle pieces. A strategy and some clues are needed to finish the puzzle. The strategy separates the functionality into **categories** of like tasks. The clue is that tasks depend on other tasks, similar to a **pyramid** structure, where upper layers depend on lower layers.

2.3.1 Management Functionality Categories

Core management software has four basic categories of functions. These are the four basic elements of supercluster management that cannot be replaced with other functionality and are required not only by the users and administrators, but also by the supercluster itself:

- **Control:** Perform actions that change something.
- **Configuration:** Generate and distribute configuration data.
- **Monitoring:** Observe the supercluster.
- **Fault Detection:** Detect fatal errors.

Three further categories are based on these four basic categories. These are the first-level derivatives of the elements which are not absolutely necessary, but simplify the management for administrators and managers:

- **Trap Handling:** Detect abnormal behavior using monitoring data.
- **Accounting:** Charge resource consumption using monitoring data.
- **Planning:** Perform actions and configuration changes based on a schedule.

The following figure illustrates these categories:

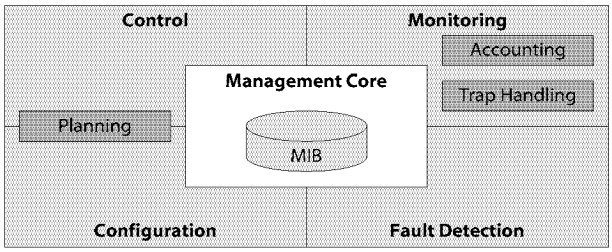


Figure 2.8: Management functionality categories

The core holds the MIB and the central intelligence detailing how the whole functionality may be integrated into a homogenous management application, how to use the functionality to achieve the main goals of the operational phase management software:

- Maximize the availability of the resources (computation time and storage).
- Minimize downtime duration and frequency.
- Protect everything and everyone from everything and everyone else.
- Gather as much information as needed from managers, administrators and users.

This section presents the categories briefly in general, using the cockpit of an airplane as a metaphor. A detailed sample functionality description is presented in Appendix C.

2.3.1.1 Control

The control category can be compared to the switches and levers that are available in a cockpit. When the pilot presses a button or turns a handle, something happens in the airplane: The engines start or stop, the plane turns left or right, etc.

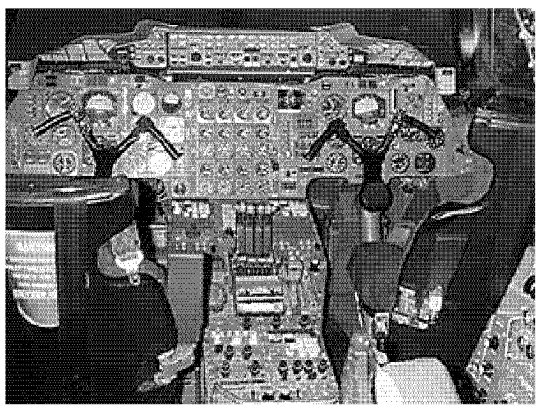
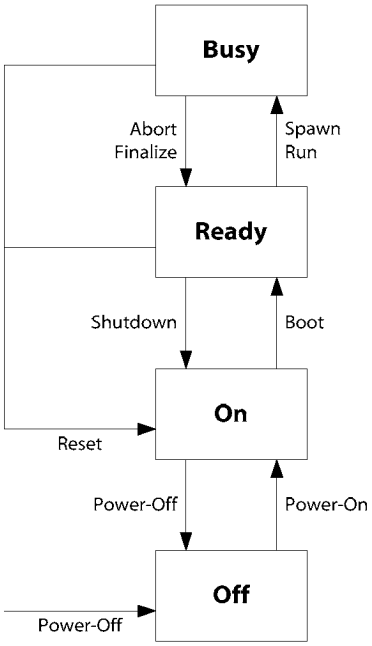


Figure 2.9: The Concorde cockpit, the workspace of the pilots for control tasks

The control functionality group has the same kind of tasks: Switching parts on and off, distribute applications, change configurations, etc.

The power status is basic for the supercluster hardware. The actions that can be taken by the management software (and the users and administrators) depend on the current status of every individual part or the status of the whole system.



The state «**Busy**» indicates that resources are being used by an application. The action *Spawn* starts a parallel application process; the action *Run* starts a local application (e.g. for management tasks). The application can either terminate successfully (*Finalize*) or be terminated by an error or abortion (*Abort*).

In the state «**Ready**», the component is booted and ready to be used. The management software provides the action *Boot* that boots the node (with the option, which boot image to select).

In the state «**On**», the component is shut down, ready to be booted or switched off. The management software provides the action *Power-On* that turns the power on and *Shutdown* that shuts down the operating system. *Reset* returns each component to this status immediately like pushing the reset button on a PC.

In the state «**Off**», the component is powered off. The management software provides the action *Power-Off* that turns the component off, using either hardware-internal power-switching features (e.g. console) or external remote power switches.

Figure 2.10: Power states of supercluster hardware

Using graphical user interface (GUI) software for system management, the functionality of this category is available in a context-sensitive menu accessed by clicking on a component with the mouse, as in the following illustration.

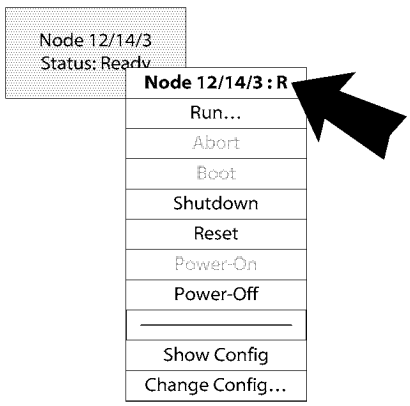


Figure 2.11: Sample context-sensitive menu for nodes with available control actions

Of course, not only single nodes can be selected in the GUI application, but also groups of nodes, groups of parts of every subsystem, subclusters including switches and storage devices etc.

2.3.1.2 Configuration

The configuration functionality category can be compared to plane setups, where correct settings allow the pilots to perform the actions they want: There is a setup for docking, a setup for taking-off, a setup for the flight itself, a setup for landing, etc. Each setup is described in a manual and achieved through actions previously described.

Besides these basic setups, there are setups for special cases, e.g. if turbulence requires a route change, balance changes if an engine is switched off, or another landing setup because a tire became flat during the start. These temporary configurations are also described in a manual, but they are only selected when special conditions are present. After the special condition has disappeared, the pilots select the basic setup again.

The supercluster also has a set of basic configurations as the following samples suggest:

- Whole supercluster is switched off.
- Whole supercluster is on, in use for management tasks, disabled for public use.
- Whole supercluster is on, open for public daytime use (small applications).
- Whole supercluster is on, open for public night or week-end use (large applications).
- Whole supercluster is on, parts are open for public use, and other parts are closed for management tasks.

The basic configurations can be altered in special cases as the following samples suggest:

- A link or switch is broken and the network traffic must be re-routed. The routing tables of the surrounding switches are recalculated and distributed. After the problem has been fixed, the routing tables are reset.
- A parallel application creates a MPI group which requires a routing table modification for fast broadcast and multicast of messages. The routing tables of the affected switches are recalculated and distributed. After application termination, the modifications are undone.
- An application generates a lot of traffic. The messages of other applications that would normally go through the same links are temporarily re-routed through links with less traffic.

The basic configurations are created (and selected) by the administrators. Temporary configurations are created on-the-fly and selected by the management software.

2.3.1.3 Monitoring

The monitoring functionality category can be compared to watching the gauges and meters in the cockpit, which describe the current status and performance. Certain values are archived for diagnostic reasons (black box) or management statistics.

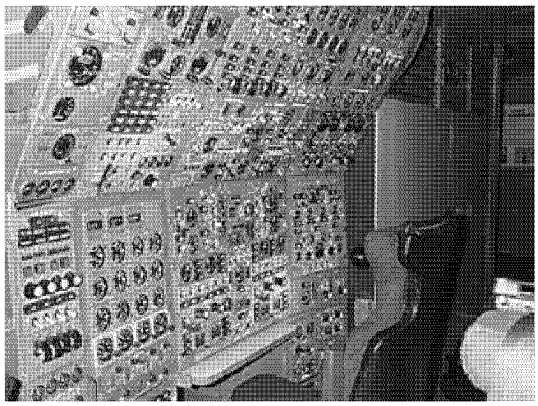


Figure 2.12: The Concorde cockpit, the workspace of the flight engineer

Monitoring functionality observes all components of the supercluster, measures various values and stores these in the MIB. These values include:

- The current power status and status changes.
- The current configuration and configuration changes.
- The current resource usage, performance and other vital values.
- All kind of events which are written in a log file.

For integrated system management, where monitored values of all subsystems are stored in the same MIB and accessed with the same user interface software, it makes sense to store the data with additional context data such as:

- Exact system time, using a global heartbeat.
- Resource usage with its cause[5], e.g. bandwidth and CPU load per application.

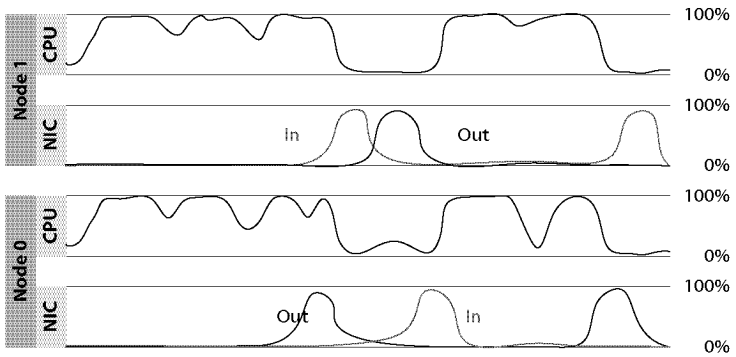


Figure 2.13: Monitoring data of two nodes (CPU load and NIC bandwidth)

The collected data is shown in graphical user interfaces for checks, and the data is also used for accounting and trap handling.

[5] The context is derived from business economics, where costs are separated into product lines, production lines and cost type. The figures allow the cost structure to be analyzed and are required for management decisions.

2.3.1.4 Fault Detection

The fault detection functionality category can be compared to the bells and flashlights that go on in the cockpit when something fatal happens and the pilots must take action quickly. If an engine is on fire, a door lock opens during flight, fuel pipes leak or gears become blocked, the pilots can perform actions that either allow the flight to continue normally or allow an emergency landing – if nothing is done, the plane is likely to crash during the flight or the landing.

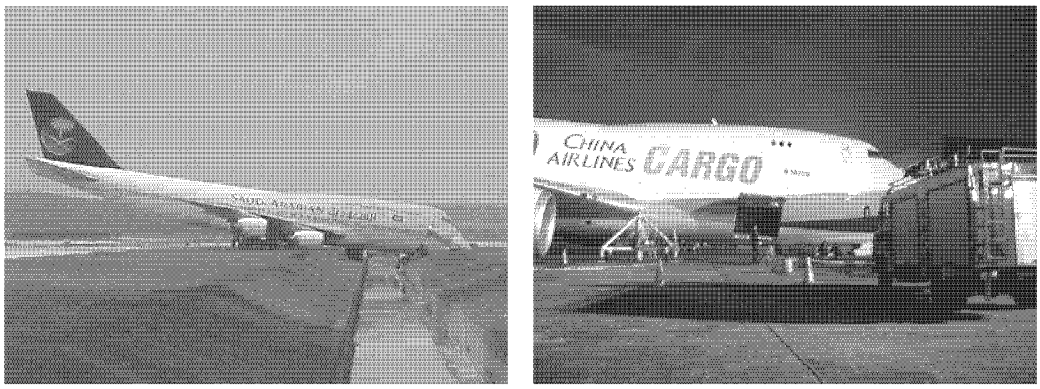


Figure 2.14: Defective brakes (left) or blocked gears (right) require precautions for landing

Supercluster faults are usually lethal for the applications, since broken nodes, switches, cables and other components usually block parts of the supercluster. If the fault remains undetected for 30 minutes, a 10 000 nodes supercluster loses 5 000 CPU hours.

Management software must not only detect the fault quickly, it must also take appropriate action to «heal» the system, thus minimizing the effect of the fault for the remaining supercluster and running applications, and maximizing the supercluster availability and performance. These «self-healing mechanisms» are stored (e.g. in a script-like language) in the MIB and usually create a temporary configuration for the supercluster.

Figure 2.15: Fault message indicating fault cause and required administrative action

Not every fault can be completely resolved automatically, but may also require manual or physical work by the administrators. The management software resolves as much as possible in order to minimize the fault effects, but it also informs the administrators about the fault and indicates its source. This allows the administrators to fix the fault quickly and minimizes the downtime duration.

2.3.1.5 Trap Handling

There are also bells and flashlights if monitored values break predefined limits. They warn the pilots about stalling conditions, ground contact without gears, collisions with other planes, too high engine temperature and many more. These warnings are necessary, as there are too many meters and gauges to be observed by the flight engineers and pilots. Exceeded limits do not automatically mean immediate danger, but they show that something might be wrong and a check is required. This allows parts that demonstrate strange behavior to be replaced or serviced – before the part breaks and causes a catastrophe.

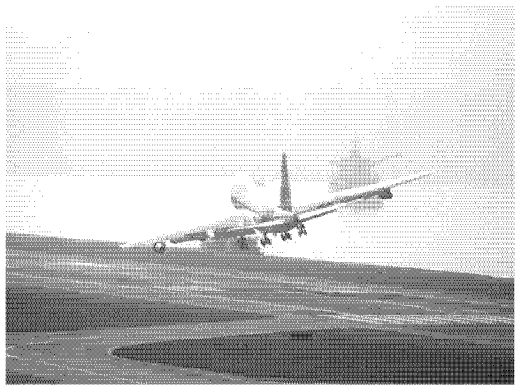


Figure 2.16: Warning bells prevent a landing where wings will touch the ground

There is no administrator watching all meters of the supercluster 24 hours per day. This task is easily performed by the management software that collects the monitoring data. Each monitoring data item is protected with a set of limits and associated actions that are triggered if a limit is exceeded (trap), in a similar way to the fault detection mechanism.

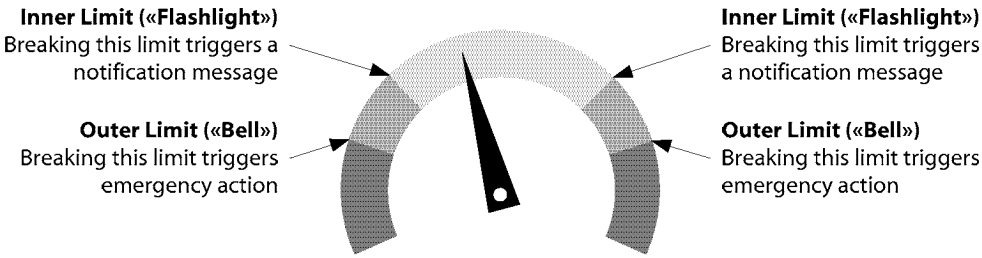


Figure 2.17: Monitored values protected with two limit pairs

The basic idea behind trap handling is the fact that components do not break from one moment to another. They usually start by having higher non-fatal error rates, higher temperatures and other values compared to other components. Trap handling prevents unexpected downtimes because «strange» components can be replaced during scheduled servicing downtimes.

2.3.1.6 Accounting

Accounting is nothing more than charging resource consumption to the consumer. The passengers are paying for the flight and the senders for the cargo. For supercomputing,

the consumption of resources (CPU time, memory, storage, network traffic) is accounted, alongside the following events:

- Application abortion due to programming errors.
- Quota violations (exceeding the «soft» limit on memory or storage consumption).
- Bad estimation of application execution time.

The managers require a tool to find out which users, projects and departments are consuming the resources. This allows the resources available for the next days and weeks to be planned. Accounting can also prove the need for a larger supercluster through the production of figures.

2.3.1.7 Scheduling

Scheduling in avionics has three tasks: Flight schedules for passengers and cargo (arrivals, departures), scheduling of the basic configurations during the flight, and scheduling of maintenance. The goal of scheduling is to optimize usage.

Figure 2.18: Scheduling of flights, configurations, and maintenance

For supercluster management, scheduling basically includes two tasks:

- Disposition – plan the available calculation time for efficient usage, e.g. using a time/node matrix, entering applications as boxes, trying to reduce unused space.
- Configuration scheduling – schedule configuration changes for the supercluster, e.g. shutdown after last application, restart with a different configuration.

Organizing the resource matrix is a complicated, NP-hard problem. At the same time, scheduling mechanisms must be simple enough that the users can check when the application will start or terminate and understand the scheduling criteria.

2.3.2 Management Pyramid

The Maslow's pyramid of self-actualization [Mas71] shows that human needs have a hierarchy: It only makes sense to reach for higher-level needs if all lower-level needs have been fully satisfied.

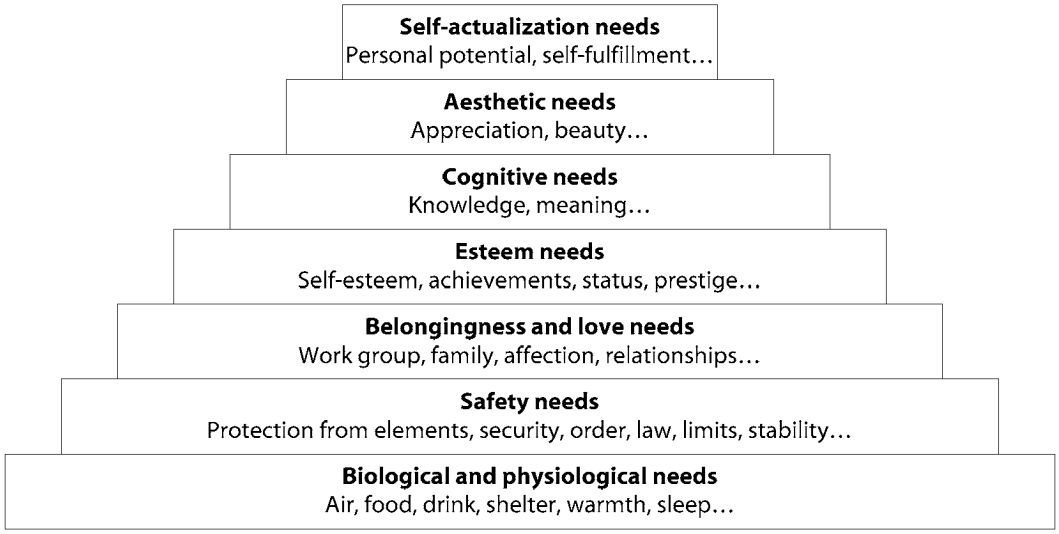


Figure 2.19: Maslow's pyramid of self-actualization [Mas71]

Supercluster management tasks also have a hierarchy: If lower-level capabilities or tasks are unavailable, it is impossible to provide higher-level tasks and capabilities. This leads to the pyramid of supercluster management.

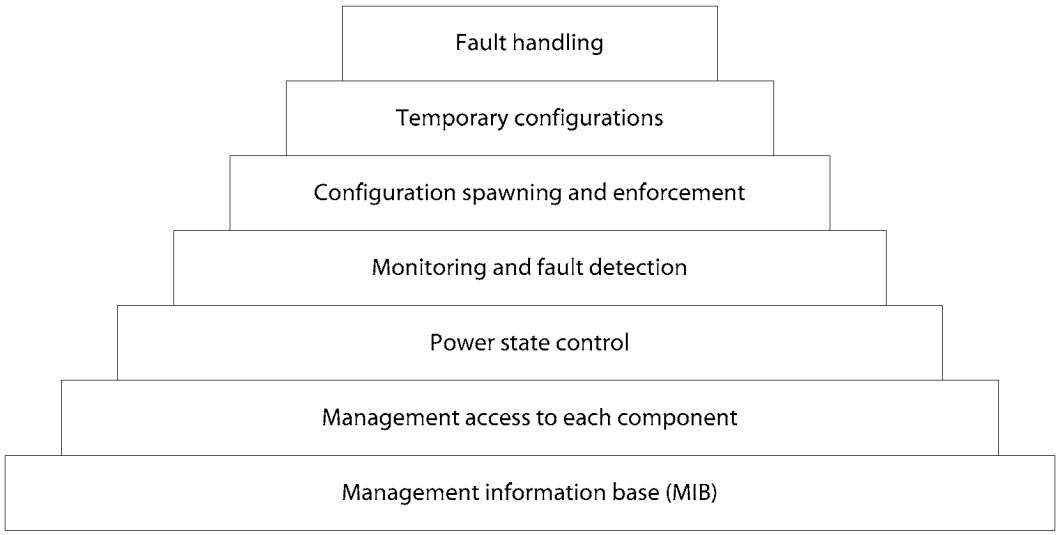


Figure 2.20: Supercluster management pyramid

The basis of supercluster management is the **management information base** (MIB). It provides all the data that the higher levels need, and it collects all the data of the higher layers.

The management software must have **management access to each component**, the ability to control it, to enforce configurations, to monitor it, and to detect faults. The MIB contains all the connections to each component plus the protocol for using each connection.

The basic management action is the **power status control**, the ability to switch all components on and off, globally as well as individually. This requires management access to each component.

The founders of SNMP (Simple Network Management Protocol) [Ros96] commented that «showing up» solves most of the management problems. **Monitoring and fault detection** is therefore the next layer in the pyramid. A detected fault may require switching on and off components.

The management software has sets of basic configurations. These configurations are selected by administrators or the management software. **Configuration spawning and enforcement** makes sure that the selected configuration is used by the supercluster. Checking the configuration requires monitoring (version check) and fault detection (blocking components).

In certain cases, the basic configuration is slightly altered: Faults, traps and applications require temporary changes which are removed afterwards. These **temporary configurations** are calculated by the management software and require mechanisms that distribute the configurations.

Automated **fault handling** is at the top of the pyramid. It requires the ability to detect faults, create temporary configurations and distribute them to all components.

This hierarchy has one important clue for management software developers: The management software's goals define the height of the pyramid. The higher the pyramid is built, the broader and stronger the lower levels must be. Adding an additional layer to existing software means that some effort will be necessary to improve the lower levels. The author assumes that every additional layer doubles the implementation efforts for the management software developers.

2.4 Bottlenecks and Central Requirements

The supercluster people and their expectations have been introduced, the supercluster lifecycle has been presented, and the management functionality has been described. So far, this chapter has created the illusion that creating supercluster management software is easy and straightforward.

So what are the problems that made this dissertation necessary? Why are the administrators and users of existing supersize-superclusters unhappy with the management software? Why is supercluster management currently a never-ending tragedy?

In mathematics, very large numbers obey different laws. In supercomputer management, the supercluster, with its many components, turns the simplest tasks into logistical nightmares. There are many bottlenecks, leading to central requirements for the management software design and implementation. This section illustrates these problems.

2.4.1 Scalability

The biggest problem in supercluster management is scalability, which appears at various places with various effects. Reliability drops exponentially and execution time of

management tasks grows linearly with the component count. It is the job of the management software designers to find solutions to overcome the problems caused by the large scale of the parts used together.

System Size [Nodes]	Task Duration					
	1 µs	1 ms «Ping»	1 s «Spawn»	10 s «Boot»	1 min «Update»	10 min «Upgrade»
100	100 µs	100 ms	100 s	17 min	100 min	17 hrs
1 000	1 ms	1 s	17 min	3 hrs	17 hrs	7 d
10 000	10 ms	10 s	3 hrs	28 hrs	7 d	69 d
100 000	100 ms	100 s	28 hrs	12 d	69 d	2 y
1 000 000	1 s	17 min	12 d	116 d	2 y	19 y
10 000 000	10 s	3 hrs	116 d	3 y	19 y	190 y
100 000 000	100 s	28 hrs	3 y	32 y	190 y	1901 y

Table 2.2: Task duration if performed serially

The above table shows the effect of simple-minded solutions for management tasks. Good scalability does not come for free and requires good ideas from management software developers. In some cases, these worst-case duration scenarios hold true (such as booting some superclusters effectively takes hours because of SAN-to-node side effects), but management software must scale much better, such as constant or logarithmically with size.

Good scalability is one of the main requirements of the supercluster users:

- Collection, transportation and storage of high-rate monitoring data are a problem, since moving gigabytes of monitoring data to the manager is not a good idea as the following table demonstrates. But high-speed monitoring is required by users and administrators.
- The users also expect the system to operate at maximum performance. Distribution of new routing tables, after a MPI group creation of application or temporary configuration data after a fault, must happen quickly, otherwise the performance and utilization will drop.
- Administrators and users expect permanent connections between user interface, manager and managed components for observation and debugging. Maintaining thousands of permanent connections between manager and all managed components is a problem.
- For high system performance and utilization, faults need to be detected quickly – ideally before they happen so the administrators can replace weakening parts before they break. The management requires a scalable fault detection mechanism to achieve this goal.

Monitoring is the most important management feature. If 64 Bytes monitoring data is collected per node and sample, the monitoring data overloads the management software:

System Size [Nodes]	Sampling Frequency					
	1 Hz	10 Hz	100 Hz	1 kHz	10 kHz	100 kHz
100	6400	64 k	640 k	6400 k	64 M	640 M
1 000	64 k	640 k	6400 k	64 M	640 M	6400 M
10 000	640 k	6400 k	64 M	640 M	6400 M	64 G
100 000	6400 k	64 M	640 M	6400 M	64 G	640 G
1 000 000	64 M	640 M	6400 M	64 G	640 G	6400 G
10 000 000	640 M	6400 M	64 G	640 G	6400 G	64 T
100 000 000	6400 M	64 G	640 G	6400 G	64 T	640 T

Table 2.3: Data bandwidth to main managers in Bytes/s (64 Bytes per node and sample) Current systems have 1 Hz, but several kHz are required

Current systems with 1 000 nodes and a sampling rate of 1 Hz create less than 100 kBytes of monitoring data per second (and 5 GBytes per day). This amount of data is easy to handle, but current processors execute some billion instructions and modern networks transfer TBytes of data within a second. This sampling rate is obviously too slow for monitoring tasks as required by administrators and users for profiling and optimizing applications and superclusters.

Monitoring systems with 100 000 nodes and 1 kHz sample rate is challenging, creating 6.4 GBytes of monitoring data per second and 540 TBytes per day. Without a smart idea about how to handle and store this amount of data, monitoring superclusters will require superclusters themselves.

Scalability is the most important issue for system management. It should be able to manage any system, regardless of size.

2.4.2 Availability

If the management software crashes, the supercluster is blocked: Faults are not detected and handled, application requests go unanswered. The processing time since the last checkpoint (or since the application's start) is lost. Management software crashes are expensive events, since one minute costs more than US$ 58 for a 100 000 nodes system. The management software must be designed to minimize crash probability, duration and effects.

The following examples assume that the nodes have an MTBF of 5 years. The system price covers the hardware cost only (US$ 2500 per node and US$ 5000 per SAN switch, both respecting the «economy of scale» with 20% discount for the 10 times larger system), without additional costs (space, people, energy). The price per minute is the system price divided by the expected usage of 5 years.

System Size [Nodes]	Performance [FLOPS]	System MTBI	Price [US$]	Price / min. [US$]
100	1 TFLOPS	18 days	300 k	0.11
1 000	10 TFLOPS	44 hrs	2 400 k	0.91
10 000	100 TFLOPS	263 min	19 M	7.30
100 000	1 PFLOPS	26 min	154 M	58.41
1 000 000	10 PFLOPS	158 sec	1 229 M	467.26
10 000 000	100 PFLOPS	16 sec	10 B	3 738.08
100 000 000	1 EFLOPS	1 sec	79 B	29 904.63

Table 2.4: Supercluster size, performance, MTBI, cost and cost per minute (5 years usage)

Not only management software crashes create expensive downtimes. Defective components or crashed software on the component may also block the whole system. The management software must therefore also prevent loss of time and money due to faults.

High availability of the management software and the supercluster resources managed by this software is a requirement of the users, administrators and center manager. Time is money, and low utilization and performance is expensive.

2.4.3 Management Integration

Although this dissertation expects that the management of all supercluster subsystems is to be integrated into one software toolkit, this is not mandatory – at least for the moment.

Management integration can happen at three different places:

- **User Level Integration:** The user can be the integrative element. Every subsystem is managed individually with specialized management software and the users have to juggle with the tools. This is of course very complex and the bottleneck of this design is the user.
- **Middleware Level Integration:** Management software applications of each subsystem still act independently, but special software («interfaces», «middleware») is used to make the applications interact. Creating such software is difficult and is the bottleneck of this design.
- **Application Level Integration:** Management of all subsystems is integrated into one single software toolkit. Creating such software is time-consuming (has to be created from scratch) and expensive, but has the most advantages since the bottlenecks are under the control of one software instead of many independent interfaces or the users.

From an academic point of view, full integration into one application toolkit would be the most beautiful approach, moving the bottlenecks (performance, scalability, reliability) away from the users and interfaces towards the software, but creating such a system consumes the most resources.

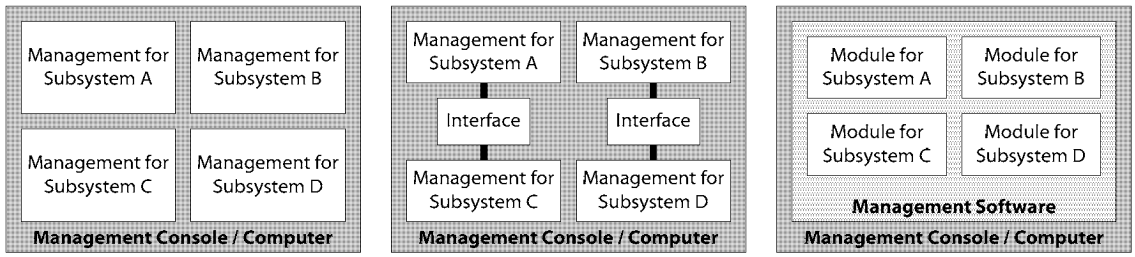


Figure 2.21: The three management integration levels (from left): User level integration, middleware level integration and application level integration

Integration is important when the users' requirements are analyzed:

- Monitoring data is only available in full context if the subsystem management is integrated (data bandwidth and CPU time per process and application, MPI groups per application). Debugging access for users and administrators also requires monitoring access to all parts.
- High availability and the reduction of fault effects are only available if the management software can access monitoring data of all components, if it can detect faults of every subsystem's component, and if it can create and apply temporary configurations to all parts of the supercluster.
- The «one system view», where the whole system can be observed centrally and the administrators and users can zoom into single cabinets, nodes and switches, requires integration.
- Effective integration requires not only that the management of the most important subsystems is integrated, but that their key features are fully supported and implemented.
- Only integration allows the creation of configuration data for all components out of one design file, created by the center manager during the design phase.

The grade of integration determines the usability of the management software.

2.4.4 Reliability

The management software must make sure that everything is reliably executed, monitored, and transferred. If all nodes but one are booted and this one blocks the supercluster, or a lost process on a node inhibits submitting further applications, or some routers have a bad routing table and messages are lost, the management software is not effective.

Reliability can be extracted from the users' requirements:

- The users expect fair scheduling of their applications. They want to be sure that their applications will finish successfully within a predefined timeframe. They do not want to be afraid that their applications will never be run because other applications are preferred.
- Users and administrators require faults to have no or low effect on the performance and utilization of the supercluster. If large jobs requiring almost all nodes cannot start because there are never enough free nodes because of faults, or if large jobs never terminate successfully because a node fault aborts the application, utilization and performance will drop.

- Faults need to be reliably detected and handled. Undetected faults will lead to low utilization and performance which is not accepted by users, administrators and center managers.
- Management tasks must be executed reliably on single parts, on groups of parts, or on all parts of the supercluster. Aborting an application for instance must wipe out all processes of the application on all nodes, since surviving processes will block these nodes for further applications, leading to low utilization and performance.

Where scalability is mandatory for efficiency, reliability is required for effectiveness.

2.4.5 Security

Security in supercluster management comes in many shapes and forms, since an unauthorized user («hacker») or application («virus», «Trojan horse», «worm») can be dangerous in various ways. The management software has the job of protecting the system and the users from malicious attacks coming from inside and outside the supercomputing center:

- The management software is the only authority that manages the supercluster. Any other managing access or action is prevented and inhibited by the management system. The management software and hardware build a «firewall» around the supercluster.
- Only authorized and authenticated users can access the supercluster and the applications of the management software. Any intruder is blocked from accessing any component. The management software is the gatekeeper and key master of the supercluster.
- Every application and all its data (files on storage, memory content, network traffic) must be protected from all other applications and users. No other application or user should be able to access this data, nor to consume resources reserved for this application.
- Even authorized personnel can trouble the supercluster operation, intentionally or unintentionally. Permission policies for every user – enforced by the management software – prevent downtimes caused by dangerous manipulations.

The weakest link defines the strength of the security chain, such as database systems for the MIB, network connections of the management software, holes in the nodes' operating system, weak authentication mechanisms, open doors in the supercomputer center, open VPN connections, or standard console passwords. The weakest link causes crashes, expensive downtimes, damage, lost secrets, stolen blueprints.

Security is a basic requirement and one of the biggest problems in every management system. Security must already be integrated in the design, since it is absolutely impossible to add security to a non-secure implementation – every attempt to prove the contrary has failed up until now.

2.4.6 Overhead

Every management system causes overheads, which are always part of discussions. Additional software on nodes consumes processing time, memory, storage, and network bandwidth. Additional computers managing the supercluster require space, money, and electricity.

The amount of additional computers for management should be low, at most in linear relationship with the system size. Higher quality of management functions (higher availability, better monitoring) may require more management computers.

The management overhead on the nodes (and the networks) must be minimal and homogenous. If some nodes have higher overheads than others, they will slow down the application. Because of the required homogeneity, the management software requires separate computers.

2.4.7 Compatibility

The supercomputing market is very heterogeneous: Ten node manufacturers, three operating systems, ten SAN manufacturers, some dozen programming libraries, technological evolutions and revolutions, and new product lines every few months. There are some 100 possible combinations. Because of the lack of management standards, integrating all products into a homogenous supercluster is hard, writing management software even harder.

Compatibility is one of the major users' requirements:

- All programming libraries must be available for the supercluster platform.
- These programming libraries must be fully implemented – partially implemented libraries are very painful for software developers, since experience has proven that those parts which are needed most in one's applications are usually those which are missing. Developers usually do not want to have to redesign their application for each platform, except if there is a huge performance gain.
- Users and administrators are used to certain tools and applications which must be available on the platform (parallel debuggers, parallel shell scripting, and message tracers).
- The management software should ideally integrate all components of all subsystems. This is only possible if the component is compatible to the management software. It is also important that it supports the key features of the component, making it possible to operate at maximum performance.

The operating system software of modern workstations supports many hardware products, and the hardware manufacturer can provide custom drivers if there is no standard driver available.

The supercluster management software must support a similar strategy, where a wide variety of supercomputing products is supported directly, and the manufacturers of supercomputing hardware and software can easily integrate their products using documented interfaces.

2.4.8 Time and Cost

The management functionality pyramid illustrated in Figure 2.20 shows that the design and implementation of management software requires a lot of work, depending on the pyramid level to be reached. The more functionality, the more supported platforms, the more work it takes.

If the management software is developed by the supercluster owner, the cost of development is low because only the components in use are supported and there is no need to create anything more than a simple tool. However the effort is almost completely lost when buying a new supercluster. The development of the management software for the Swiss-T1 supercluster as presented later took about 2 man-years. Fully integrated management software for this system would have taken double that. These 4 man-years would have cost about US$ 500 000 in an academic environment.

Turning this management software into a fully documented and tested product would have taken at least 3 as long [Bro75]. This would be the approach used by supercluster integrators that build multiple similar superclusters and sell them to supercomputing centers. The development cost for the 12 man-years project of US$ 1.5 million can be divided by the number of superclusters installed, but again, the efforts are almost lost if new product lines need to be supported.

If a company were to accept the challenge and develop general-purpose management software for many platforms, the development would take at least 10 times longer. These 40 man-years, which would cost about US$ 5 million, can be divided by the many superclusters that use the software.

These simple calculations show that developing management software is a multi-million dollar project. From an academic point of view, time and money is not a bottleneck, but in reality, it is. Additionally, the prices are based on salaries in academic environments (US$ 125 000 per year). IBM calculates internal salaries of US$ 250 000 per year.

2.5 Conclusions

What have we learned from this chapter?

There are **three phases** in the **supercluster lifecycle**:

- The **design phase**, where superclusters designs are created and evaluated.
- The **installation phases**, where the selected supercluster design is installed.
- The **operational phase**, where the supercluster is used.

All functionality is grouped in **categories** that depend on each other as in a **pyramid** and the height of the pyramid defines the required effort:

- The base is the **management information base (MIB).**
- The management software has **management access** to each component.
- Each component is **controlled** by management actions.
- **Monitoring** alone solves most of the problems and is the base for **trap handling** and **accounting** resource consumption to users and departments.
- **Fault detection** reduces downtime duration and saves money.
- The management software has **configuration setups** that can be spread and enforced.
- These basic configurations can be varied by the management software in certain cases (faults, traps, application requests) leading to **temporary configurations**.
- The top of the pyramid is the **automated fault handling** where the management software autonomously heals faults and keeps even damaged systems running.

For design and implementation of management software, there are **problems** and **bottlenecks,** caused by the nature of superclusters and by the requirements of the supercluster people.

- Some problems and bottlenecks have an effect on the management **efficiency**:
  - Scalability
  - Availability
  - Overheads
  - Management cost and implementation time
- Some problems and bottlenecks have an effect on the **effectiveness**:
  - Integration
  - Reliability
  - Security
  - Compatibility
- Inefficiency can have such fatal effects that it leads to ineffectiveness.

The next chapter will present some designs for management software of the operational phase that respect all of these issues.

2.6 Research Focus

It can be clearly seen, that research and development of integrated management software for large superclusters would be enough work for dozens of researchers and hundreds of developers. This amount of work cannot be covered in this single dissertation, but many more dissertations in this area will certainly follow, covering the previously unknown land of integrated scalable management software for large superclusters.

From the supercluster lifecycle presented in 2.2, this dissertation is only interested in the software required for the **operational phase** (presented in 2.2.4), where the supercluster is already installed on-site and ready to use by the administrators and users.

From the central requirements and bottlenecks presented in 2.4, this dissertation only covers the following issues:

- **Scalability** and **availability** is covered **in theory** in the next chapter, where architecture and design of supercluster management software are presented.
- **Integration** and **overhead** is covered **in practice** in chapter 5, where the author's implementation of supercluster management software is presented («proof of concept»).
- All other issues (reliability, time & cost, security, compatibility) are neglected in this work.

There are many activities in building supercomputers with 50 000 nodes and more. The owners of the current fastest supercomputers experience many lost processing hours due to inadequate system management features. The utilization and performance pressure from the owners will create research activities focused on creating efficient and effective management software for such number crunching monsters. Indeed, efforts were made to create DOE-based research by management software teams in the US, but the election of the current president and his budget priorities have stopped these efforts for the time being.

3 Architecture and Design

Everything should be made as simple as possible, but not simpler.

Albert Einstein

The previous chapter presented what supercluster management software has to do and which problems it has to solve. This chapter will now present the way that software must look so it can perform these tasks efficiently and effectively. This includes:

- The software pieces that build the management system.
- The relationship between these software pieces.
- The internal design of the software pieces.

One additional part is also presented: A measure for estimating how efficient and effective the architecture will be. These evaluations tables are not only used in this chapter, but also used for evaluating existing solutions and the author's implementation «COSMOS».

3.1 Management Components

Management systems generally consist of the following components:

- The **user interface**, the software used by the users for management.
- The **manager**, the central software deciding everything.
- The **agent**, the software managing the hardware and software.
- The **managed hardware** and **software** itself.

The following illustration presents the basic architecture of the components.

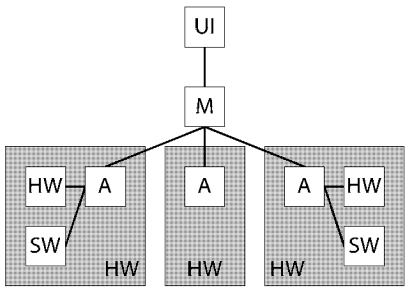


Figure 3.1: Basic management components: User interface (UI), manager (M), agents (A) and managed hardware (HW) and software (SW)

3.1.1 User Interfaces

Management software usually operates in the background, managing the supercluster automatically without user interactions. The user interface allows the management software to step out of this darkness, and allows users to interact with the supercluster.

There are three basic types of user interfaces for management software:

- The **graphical user interface (GUI)** is the most suitable strategy for users. Information is shown graphically or in tabular style in a window, and the users use their mouse and keyboard to execute their tasks. With current web-based technology, it is also possible to create GUI applications that run in web browsers.
- The **command line based interface (CLI)** is limited in presentation and usage to keyboard and text-based terminals. Users type their commands in a command line, and the response is written as a list of data.
- The **shell-based interface** consists of many commands used in shells or scripts.

State-of-the-art management software must include at least the shell-based user interface and the graphical user interface. The shell-based user interface is mandatory, since users and administrators are used to shells and shell scripts in UNIX and Linux. The GUI is mandatory, because large amounts of data can only be conveniently presented graphically.

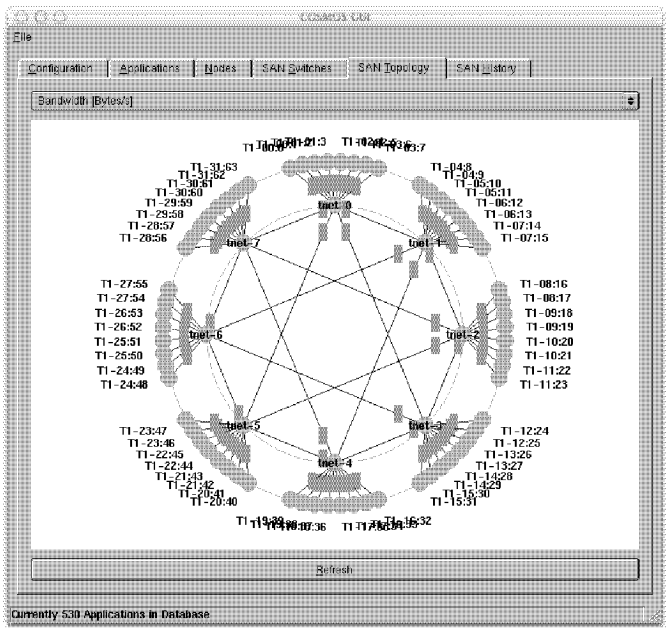


Figure 3.2: The GUI of COSMOS, the Swiss-T1 supercluster management software

In the presented architecture, the user interface connects to the manager only to exchange management data, it does not connect to any agent, managed hardware or software, or other user interfaces. There can be as many user interfaces connected to the manager as he can handle without compromising performance or reliability.

The user interfaces are the face of the management software. It is therefore important that the users like the face and that the promises made by the face are matched by the underlying management functionality.

3.1.2 Manager

The supercluster is managed centrally by the manager application. For non-integrated management architectures, there may be multiple managers, where each manager only manages the associated subsystems (e.g. nodes). For availability and performance reasons, multiple instances of the same manager application are possible, but they require internal synchronization.

The manager application is connected to the user interfaces and the agents described next. The tasks that are performed by the manager can be described as follows:

- The manager maintains a global database, where everything relevant for management is stored. This database is called MIB (Management Information Base) and usually has a hierarchical and relational structure (see Appendix B).
- The manager sends orders to the connected agents. These orders are either based on the commands sent by the administrators and users through the UIs, or they are automated orders based on scheduled services, detected faults or traps, or requests from applications (e.g. creation of multicast routing table extensions).
- The manager receives messages that are stored in the MIB from the agents and/or which trigger automated reactions as configured by the administrators or users.

Since the manager must always be available (for the GUI and automated reactions to faults and traps), it is reasonable that this application be implemented as background service («demon» [Ste98]). The following figure illustrates the basic structure of the manager application.

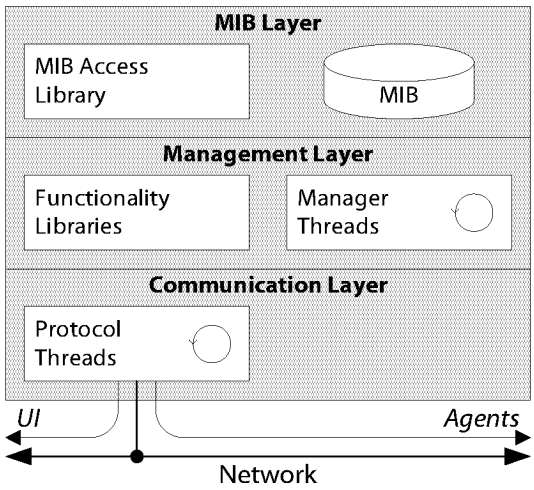


Figure 3.3: Basic structure of the manager application

This basic structure consists of three layers:

- The **Communication Layer** maintains the connections to all agents and user interfaces. It consists of one or more threads that handle connections and messages.
- The **Management Layer** contains the management logic. It consists of the functionality libraries (one per managed subsystem) that describe how the components are managed, and the manager threads (usually one per managed subsystem) that perform the management.

- The **MIB Layer** contains the MIB itself and information about how to access the MIB. The MIB can be implemented either in the manager's memory or as a persistent external database.

Every layer has problems that must be solved by the developers in the design:

- The communication layer usually maintains persistent connections to the user interfaces and agents. These persistent connections are usually socket-based TCP connections that create a lot of overhead, which in turn limits the amount of concurrent connections to about 200.
- The management layer contains one functionality library per managed subsystem. Since flexibility should be the goal of the design, the interface between library and manager should be flexible enough to support multiple platforms (although not at the same time).
- The MIB is filled with data, accessed by the center (read/write) and the user interfaces (read only, usually through the center). Storing the MIB in a database system (e.g. Oracle or MySQL) is a good idea, but these database systems have limited performance.

Within this document, the manager is referred to as «management center» or «center».

3.1.3 Agent

The agent application runs on or close to the associated managed hardware or software. It is connected to the center and performs the orders received from it. Since the agent must be always available (to receive the orders or send fault notifications), it is reasonable that this application be implemented as «demon» [Ste98].

The following figure illustrates the basic structure of the agent application.

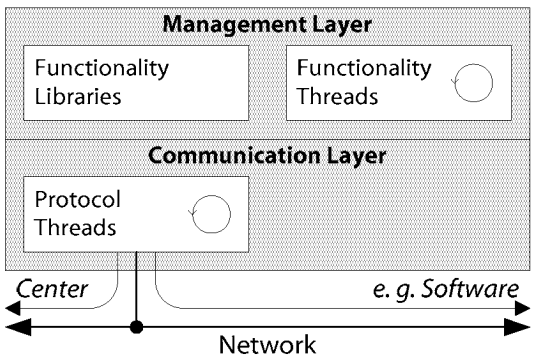


Figure 3.4: Basic structure of the agent application

This basic structure consists of two layers:

- The **Communication Layer** maintains the connections to the center and – if applicable – managed hardware or software. It consists of one or more threads that handle connections and messages.
- The **Management Layer** contains the management logic. It consists of the functionality libraries (one per subsystem that is managed by the agent) that describe

how the component is managed, and the functionality threads (usually one per managed subsystem) that perform the management.

Since the number of subsystems and different components in a supercluster is limited, there will be the following potential agents in such a system:

- The **node agent** runs on the node computers, managing their hardware and software. Depending on the management architecture and implementation of the features, it is possible that there will be multiple agents running on the nodes, each serving other subsystems or functionality, e.g. one separate replication agent for the spawning mechanism, one agent for the local SAN NIC, one agent for the LAN NIC, one for the node itself, one to manage the application etc.
- The **SAN agent** runs either in the SAN switches themselves, or on separate computers. If it runs on separate computers, it is possible for the same agent to manage multiple (or all) SAN switches of a supercluster. In this case, the agent connects to the switch-internal operating system that performs the management tasks. If the SAN agent runs in the SAN switch itself, the functionality library allows those tasks to be performed.
- The **LAN agent** usually runs on a separate computer, and manages multiple (or all) LAN components of the supercluster (Ethernet switches, routers, firewalls, console servers). Since these components are usually managed using SNMP, the LAN agent translates the management-internal protocol to SNMP, making it more reliable and effective.
- The **storage agent** usually runs on the computers that connect the storage subsystem to the supercluster. These computers have special hardware and software installed that transfer the data to the nodes over the SAN or LAN. The storage agent usually communicates with this manufacturer-supplied software for management purposes.
- The **power agent** runs on separate computers and is used to manage the power state of all supercluster components. Nodes are usually controlled directly via the console (through console servers). Other components may require remote power switches.

The node agent is very delicate, since it consumes resources on the computer that is used for calculation. Users are very suspicious about services running on the nodes, since even the slightest imbalances on the nodes can have a huge impact on performance:

- If one node has an additional service that consumes 1‰ of CPU time, there is only 99.9% CPU time left for the process of the parallel application. All other processes must wait for the process on this node. This is the reason why the nodes must have balanced load.
- If the service is required to send monitoring data once a minute to the center (taking 0.05 CPU seconds) and the iteration of the currently running parallel application takes 1 second, the iteration is slowed down to 1.05 seconds if the services are not globally synchronized, since there is always at least one node that has to send the data in this iteration. If the service is globally synchronized, only one single iteration step is slowed down per minute.

The first case slows the application down from 100% to 99.9% of the potential peak. In the second case, the application is slowed down from the potential peak of 99.2% to 95.2%. Considering shorter iteration steps, higher process switch penalties and potential cache memory trashing, performance can drop far below 80%.

For the other agents, resource consumption is not as important.

3.1.4 Managed Hardware

The goal of comprehensive system management is to manage all components of the supercluster. Therefore, all hardware must be manageable, and is usually managed by the associated agents:

- The nodes and all included hardware are managed by the node agents.
- The LAN components are managed by the LAN agents.
- The SAN switches are managed by the SAN agents.
- The storage is either managed by the storage agents (if the storage subsystem has dedicated hardware) or the node agents (if distributed file systems are used that use the node-internal storage devices).
- The power agents are used for supporting the centers in controlling the power status of all hardware components – if the centers do not control the power status by themselves.

If the managed hardware is able to include the agents as described earlier, the structure of the managed hardware and the agent are glued together as in the following figure.

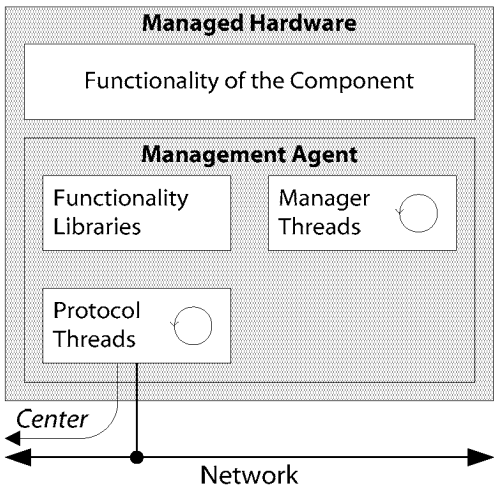


Figure 3.5: Managed hardware with integrated agent (lower box)

The functionality of the component is on top of the management agent, managed by the functionality library.

If the managed hardware cannot include the agent, but has an internal operating system responsible for management, the agent must translate the management protocol used by the center to the protocol used by the hardware component (such as SNMP).

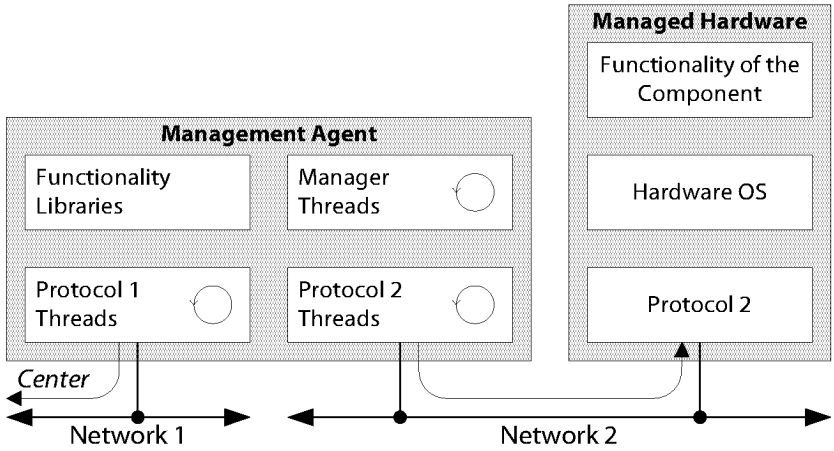


Figure 3.6: Managed hardware (right) and associated agent (left)

This kind of management is very likely to be used for storage, SAN and LAN management, where the subsystem is usually managed by custom software.

3.1.5 Managed Software

The «managed software» is the set of parallel applications with their processes, consuming the resources of the node computers. The other software subsystems (users and resources) are managed by the center alone, but the application processes must be managed by the node agents for the following reasons:

- It is easier for the process to connect with a local host than a distant center.
- Only the node agents can efficiently detect the resource usage of the processes.
- Only the node agents can reliably abort applications or detect faults.

The integration of the application processes into the management software requires a library that is linked to the application directly, or linked to the parallel environment.

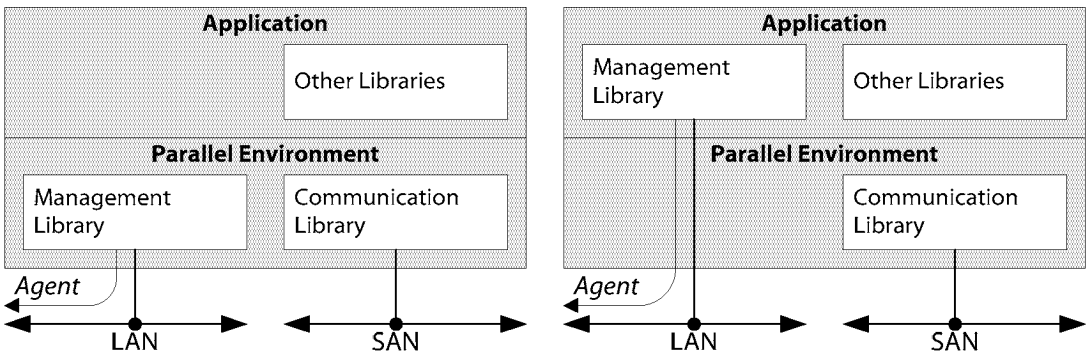


Figure 3.7: Managed software («application library») with integration in the parallel environment (left) or in the application (right)

If the library is linked to the parallel environments, the manufacturers of the library have to re-design their software. The advantage of this approach is that all applications using these parallel environments can be integrated into the system management. The disadvantage is that manufacturers are not very motivated to alter their source codes.

If the library is linked to the applications, the application developers have to re-design their applications. The advantage of this approach is that the parallel environments can be left as they are («commodity» off-the-shelf and unaltered). The disadvantage is that only applications using this library can be integrated into the system management software.

The parallel environment manufacturers can be motivated to change their software if they can benefit from the centralized and integrated system management – such as receiving SAN device information and control. The application developers can be motivated to change their applications, if the management library allows welcome features (such as checkpointing, enhanced monitoring, or process migration). Ideally, the management library should consist of both parts: The (mandatory) library for the parallel environment, and the (optional) library for the application.

For the Swiss-Tx project, the parallel environment FCI/MPI was created by the project team. The initial version that was created for the EasyNet hardware, had no management interface, and the parallel application managed itself using socket connections between all processes of the same application. Since 63 concurrent connections cause a lot of overhead and FCI/MPI was redesigned anyway for the T-Net hardware, this management part was taken over by COSMOS, and FCI/MPI depended on COSMOS. The second parallel environment used on the Swiss-T1 supercluster (MPICH, an open-source MPI implementation using sockets) was not altered and therefore not managed by COSMOS. It would have been possible to alter MPICH to use COSMOS, but since this software has regular patches and many parts would have had to have been changed, this path was not followed – also because the benefit would have been low for such a small system.

3.2 Architecture Evaluation Method

What is a good design and what is a bad design for supercluster management software? How can its performance be measured or estimated? Which architectures are suitable for large superclusters, which due to the limitations of their design are suitable for small systems only? This section presents a measure for comparing supercluster software architectures and designs, based on three different tables:

- The first table evaluates the design efficiency for systems between 100 and 100 000 nodes.
- The second table evaluates the bottlenecks and central requirements.
- The last table evaluates the support of typical management tasks.

The quantitative and qualitative results allow developers to select the right design for their supercluster management software.

3.2.1 Quantitative Evaluation Table

Bad decisions in software design quickly become evident if the size of the problem increases. Without smart engineering ideas, execution time and resource usage grow in

linear relationship (or even worse) to the problem size. For management software, bad design decisions are disastrous – everything seems to work fine in the confined testing environment of the designers' laboratory, but does not work reliably anymore when installed on huge productive systems.

The quantitative evaluation table contains values that can be measured, calculated or estimated. These values are characteristic of the software architecture and will not change in orders of magnitude between designs based on the same architecture.

The following table shows an example of the quantitative evaluation table.

Criteria	Value	System Size [Nodes] (+ 1 Switch per 10 Nodes)			
		100	1 000	10 000	100 000
Required bandwidth for monitoring (64 B/Comp.)	1 Hz	6.4 kB/s	64 kB/s	640 kB/s	6.4 MB/s
	1 kHz	6.4 MB/s	64 MB/s	640 MB/s	6.4 GB/s
Duration startup/shutdown	10 s/Comp.	18 min	3 hrs	31 hrs	13 days
Duration OS download	30 s/Node	50 min	8 hrs	83 hrs	35 days
Duration group creation	0.1 s/Comp.	11 sec	2 min	18 min	3 hrs
Duration process spawn	1 s/Node	2 min	17 min	3 hrs	28 hrs
Duration fault detection	0.5 s/Comp.	55 sec	9 min	92 min	15 hrs
Duration config change	1 s/Comp.	2 min	18 min	3 hrs	31 hrs

Table 3.1: Quantitative evaluation table sample for management software designs

The first two lines of the table show the required network bandwidth of the center for monitoring. These values are based on the assumption, that the agents send their monitoring data to the center (average of 64 Bytes per component and sample), and that the sampling rate is either 1 Hz (upper line) or 1 kHz (lower line). Since the bandwidth to the canter is limited by the LAN technology (10 to 100 MBytes/s), the monitoring coverage is limited as well.

The other lines show the execution time for the most important management tasks if executed sequentially for all components. This sequence is of course simplified and it is the task of the developers to find smarter solutions, but in some cases it might be mandatory.

- The duration of startup and shutdown is 10 seconds per node, because the center must use the console server connections, send a sequence of text-based commands and wait for text-based responses. It would be nice to be able to send one simple Ethernet packet to the LAN NIC for this task, but the current UNIX-based workstations and servers require such a protocol.
- Upgrading the node OS takes about 30 seconds per node, because downloading an image of about 150 MBytes from the center should be serialized for efficient storage access, and the bandwidth to and from the center is limited.
- Creating or deleting an MPI group includes the calculation and distribution of new routing tables to the SAN NICs and switches, taking about 0.1 seconds per affected component. In the worst case, all components require new routing tables.
- The start-up of parallel applications includes the creation of the processes on the nodes. This can be performed using a shell script (e.g. «mpirun») with serial re-

mote shell calls from the center, by forking away processes from the node agents, or other mechanisms. Center-based startup mechanisms using remote shells require one second per node.

- There are two ways of checking reliably whether a component is operating or not: Either the component is actively sending its current status to the center, or the center has to query the component. Querying each component may take 0.5 seconds per component, performed by a thread or process in the center.
- Distributing a new configuration (e.g. after a fault) may take longer than a simple routing table update, about one second per component, performed sequentially by the center.

The 64 Bytes monitoring data per component and sample is calculated as follows:

Object	Monitored Value	Unit	Size [B]	Rate
Node	Temperature	°C	1	Low
	CPU Load	%	1	Low
	Memory Usage	%	1	Low
	Storage Usage	%	1	Low
Node / Process	Memory Consumption	kBytes	4	High
	Storage Consumption	MBytes	4	High
	CPU load	%	1	High
	Message Data Sent	kBytes	4	High
	Message Data Received	kBytes	4	High
Node / SAN NIC / Port	Bandwidth In / Out	Bytes/s	8	High
	Packet Rate In / Out	Packet/s	8	High
	Error Rate In / Out	%	2	High
SAN Switch	Temperature	°C	1	Low
	CPU load	%	1	Low
	Memory Usage (2 types)	%	2	Low
SAN Switch / Port	Bandwidth In / Out	Bytes/s	8	High
	Packet Rate In / Out	Packet/s	8	High
	Error Rate In / Out	%	2	High

Table 3.2: Sample monitoring data for node and SAN switch

For this sample, each node contains one single-port SAN NIC and one process slot. This leads to 39 Bytes of monitoring data per node. The SAN switch has 16 ports, leading to 292 Bytes per switch. The data amount per 10 nodes is 682 Bytes, and with smart compression (high and low sampling rates) and additional overhead, these 68.2 Bytes per node can be reduced to 64 Bytes.

The effective time or required bandwidth differs dramatically between architectures. Smart designs will massively reduce these values and make management more scalable.

3.2.2 Qualitative Evaluation Table

«Not everything that counts can be counted – and not everything that can be counted counts» – this succinct saying of Albert Einstein is also valid for system management design, because not everything important can be precisely measured. The qualitative evaluation table includes eight points that cannot be measured, but are important for efficiency and effectiveness.

Criteria	Quality	Comments
Monitoring Synchronization	– / 0 / +	How difficult is it to implement a global heartbeat mechanism in the design?
Monitoring Context	– / 0 / +	How well can the monitoring data be integrated into other subsystems' data?
Development Time & Cost	– / 0 / +	How complex is the design, how much time and money will implementation take?
Management Overhead	– / 0 / +	How many additional resources need to be added for system management?
Reliability & Availability	– / 0 / +	How reliable is the distribution and execution of administrative tasks in the system?
Scalability	– / 0 / +	How well does the management scale? What is the estimated maximum system size?
Security & Vulnerability	– / 0 / +	How well are the applications, users and system components protected?
Flexibility	– / 0 / +	How flexible is the management if requirements, size or components are to be changed?

Table 3.3: Qualitative evaluation table sample for management software designs

These eight points include:

- Effective system monitoring requires a global heartbeat, a regularly repeated moment during which all monitoring data is sampled, collected and sent to the center. This point analyses how well this synchronization has been or could be implemented.
- Effective system monitoring also requires a context between monitoring data (e.g. bandwidth consumed by process x of application y). This point analyses whether the context of the monitoring data is also collected for precise statistics.
- Most software projects fail because of a lack of calendar time and money. This point analyses the implementation cost and time for the design, as well as whether invested time is protected if the design is improved.
- All management creates overheads: Additional computers to run the management software must be installed and processor cycles on the nodes are used for administration. This point analyses the overheads created by the system management and its scalability.
- The management software availability is important, since crashed management software leads to a blocked system, thereby losing much processing time. This point analyses the mechanisms in the design that should permit high availability and reliability.

- Scalability is the key to supercomputing. This point analyses whether the design scales with the supercluster size, based on the quantitative evaluation table previously presented.
- Security and vulnerability become important as soon as applications of secret projects are executed on the supercluster. This point analyses the protection from malicious use of users, applications, supercluster hardware and software as well as the management software itself.
- Some management software is designed for use with only one system, and some for use with only one platform. If the size is changed or the main product of a subsystem is changed, the management software can only operate with difficulties. This point analyses the flexibility of the design.

This qualitative evaluation table describes the design on a «management summary» level.

3.2.3 Typical Tasks Evaluation Table

This table evaluates how well the management software supports the typical tasks. It also describes the quality on a «management summary» level as the qualitative evaluation table, selecting only the most important typical tasks that have a direct effect on the system manageability.

Criteria	Quality	Comments
Design, Installation & Configuration	– / 0 / +	How well are the design, installation and configuration processes supported?
Power Management Implementation	– / 0 / +	How reliable and efficiently are start-up, boot and shut-down implemented?
Monitoring Transport to User Interfaces	– / 0 / +	How good is the monitoring data transport from components to user interfaces?
Process Groups / Routing Tables Creation	– / 0 / +	How well are process groups supported by SAN and system management?
Application & Process Control	– / 0 / +	How completely are applications and their processes managed and observed?
Detection & Handling of Faults	– / 0 / +	How efficiently and reliably are faults found and corrected?

Table 3.4: Typical tasks evaluation table sample for management software designs

The following tasks are analyzed:

- Design, simulation, installation and configuration are the most time-consuming tasks, since they are usually not supported by the system management software. If these tasks are computer aided, the system design quality will improve since changes are quickly simulated, distributed, checked – and undone.
- Power control is the second step of the management pyramid (see Figure 2.20). If changing the power state does not scale or is unreliable, effective management is impossible.
- Transporting (and storing) monitoring data is one of the most important tasks of the system management software because they are of global interest. Efficient

transport (between component and user interface) and smart storage (for later retrieval) are mandatory.

- Process groups are frequently created and group-internal communication often used in MPI applications. Some SAN technologies support this multicast-type communication, but the management software must quickly calculate and distribute the new routing tables for the NICs and switches. If unavailable, the processes must use sequential unicast communication and this slows down the performance.
- Parallel applications and their processes must be tightly controlled by the management software. Lost processes and forgotten applications block the system, leading to abortions of succeeding applications, and low usage and efficiency.
- In systems of several thousand components, faults occur regularly and potentially block the system completely. This is the reason why faults must be detected immediately, their effects must be reduced and affected areas and applications must be restarted or taken off-line. All this requires quick calculation of temporarily used configurations and reliable transport to all affected components – and later removal if the fault is corrected.

This table, together with the qualitative and quantitative evaluation tables previously presented, covers the system management software completely, evaluating design proposals in this chapter, the existing designs in chapter 4 and the COSMOS design in chapter 5.

3.3 Management Architectures

There are three main grades of management integration, leading to three main architectures:

- If there is no integration between the management software of each subsystem, the software design follows the **«non-integrated software architecture»**. The user or administrator has to manage each component and subsystem individually. This architecture is presented in subsection 3.3.1.
- If only some supercluster subsystems are integrated in one management application, the software design follows the **«glueware software architecture»**. The management simplifies the utilization and administration of the supercluster, because tasks that require interactions between subsystems that are glued together can perform automatically. This architecture is presented in subsection 3.3.2.
- If the management of all subsystems is integrated into one application, the software design follows the **«integrated software architecture»**. All tasks that require interactions between subsystems can be performed automatically, since all subsystems are managed by the same application. This architecture is presented in subsection 3.3.3.

The integrated architecture is presented with further capabilities, such as availability-enhancing features in subsection 3.3.4 and scalability-enabling features in subsection 3.3.5.

3.3.1 Non-Integrated Architecture

In the non-integrated management architecture, all components of each subsystem are managed by the specifically designed management software toolsets of the manufacturers. There are no interfaces between the toolsets (such as between SAN and nodes or applications). Some of the management functionality can be provided by using shell scripts (e.g. remote shell scripts for power control or process management). These tasks are usually executed sequentially.

This architecture requires that users and administrators use the user interface applications of each subsystem in parallel. There might be user interface applications that use those user interfaces to create something that looks like an integrated user interface (such as web-based solutions), but they are no match when compared with the integrated management applications.

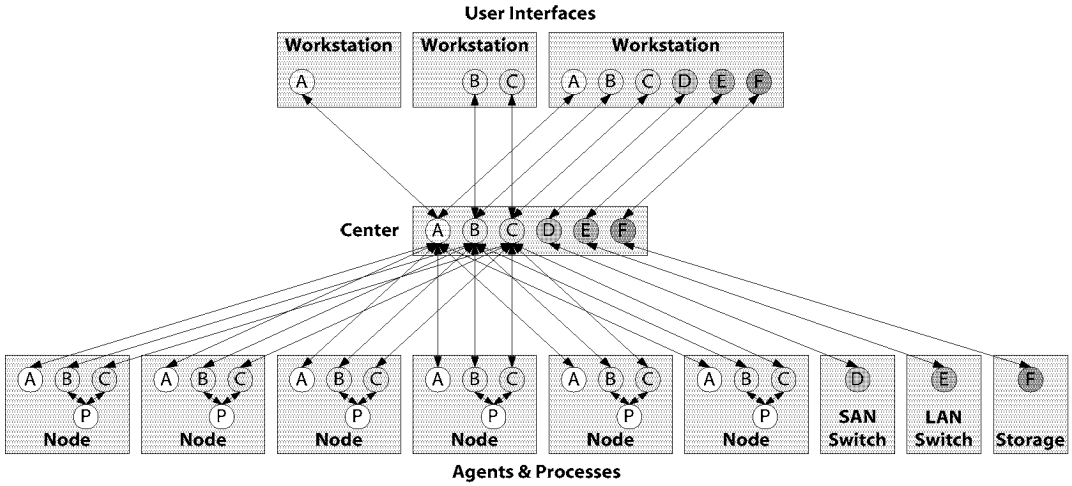


Figure 3.8: Non-Integrated Management Architecture Sample

Figure 3.8 illustrates a simplified sample supercluster that is managed by six independent management applications («A» to «F»). Additional management components (scripts, MIB etc.) have been omitted in this simplified figure.

- The six center applications («A» to «F») run on the center computer.
- There are three workstations that have user interfaces connected to the centers:
  - The left hand workstation runs only the user interface of management application «A», e.g. for monitoring the SAN NICs.
  - The middle workstation runs the user interfaces of management applications «B» and «C», e.g. for monitoring the user's applications and their resource usage.
  - The right hand workstation runs the user interfaces of all management applications. This workstation is very likely to be the computer of the administrator that wants to be fully informed of the supercluster's status and performance.
- There are three agents running on the nodes («A» to «C»), where two of them observe the processes of the parallel application («P»).
- One agent controls the SAN switch («D»), one the LAN switch («E») and one additional agent the storage subsystem («F»)

Since there are no interfaces between subsystem management applications, there is no possibility for symbiosis, such as:

- Crashed nodes are not reported by the node management to the SAN management.
- Defective SAN NICs are not reported by the SAN management to the resource management to prevent further applications on the affected node.

The missing integration is compensated for by the manual work of the administrators and users. Up to a certain size (and quality level expectation), this type of management works well. But this approach is not very professional, since permanent administrative presence is required.

The evaluation table of this architecture is as follows:

Criteria	Value	System Size [Nodes] (+ 1 Switch per 10 Nodes)			
		100	1 000	10 000	100 000
Required bandwidth for monitoring (64 B/Comp.)	1 Hz	6.4 kB/s	64 kB/s	640 kB/s	6.4 MB/s
	1 kHz	6.4 MB/s	64 MB/s	640 MB/s	6.4 GB/s
Duration startup/shutdown	10 s/Comp.	18 min	3 hrs	1 day	13 days
Duration OS download	30 s/Node	1 hrs	8 hrs	3 days	35 days
Duration group creation	0.1 s/Comp.	11 sec	2 min	18 min	3 hrs
Duration process spawn	1 s/Comp.	2 min	18 min	3 hrs	1 day
Duration fault detection	2 s/Comp.	4 min	37 min	6 hrs	3 days
Duration config change	1 s/Comp.	2 min	18 min	3 hrs	1 day

Table 3.5: Quantitative evaluation table for non-integrated architecture

The above figures allow the following observations:

- The fault detection duration (that grows with size if implemented as sequentially performed checks) competes with the MTBI (that decreases with size).
- The sequentially executed tasks for startup/shutdown or process spawning show the system limits concerning scalability and expected reliability.

The bare numbers show that only small systems of some hundred nodes can be reliably and effectively managed with this management architecture.

Criteria	Quality	Comments
Monitoring Synchronization	–	No global heartbeat, no synchronization between subsystems possible
Monitoring Context	–	No context possible to monitoring data of the other subsystems
Development Time & Cost	+ / –	Quickly developed and installed, but hard to debug, maintain and extend
Management Overhead	+ / –	No additional HW, many node agents
Reliability & Availability	–	Each tool acts independently of all others
Scalability	–	Limited by the least scalable subsystem management software and shell scripts
Security & Vulnerability	–	The weakest security mechanism of all subsystems defines its vulnerability
Flexibility	+	Management quickly adapted to changes

Table 3.6: Qualitative evaluation table for non-integrated architecture

Except for the flexibility and the low cost of development and usage, there are many disadvantages to this architecture. The most problematic disadvantages are the bad scalability and low availability and reliability, together with the many agents running on the nodes.

Criteria	Quality	Comments
Design, Installation & Configuration	–	Manual hard work with no computer-aided support, many sources of mistakes
Power Management Implementation	–	Script-based, slow and unreliable
Monitoring Transport to User Interfaces	–	Each tool transports data independently to centers and user interfaces
Process Groups / Routing Tables Creation	–	Not possible, since node, SAN and application management are not integrated
Application & Process Control	–	Separate from other subsystems, no current system information (unavailable nodes due to non-application based faults)
Detection & Handling of Faults	–	Simple faults are handled, complex multi-subsystem faults require the administrator to be on-line and on-site

Table 3.7: Typical tasks evaluation table for non-integrated architecture

Except for the fact that the non-integrated management system is inexpensive, there are too many problems making it hard to use efficiently and manage the supercluster effectively. This is the reason why only small systems should be managed with non-integrated management software – or systems where service availability and reliability is unimportant.

The non-integrated management software approach is indeed the most frequently used architecture, since the applications can be used, updated, and altered (almost) independently from each other. The high number of concurrent socket connections between centers and agents (and the overheads created by these connections in the center) can be addressed by use of separate computers for each of the management center applications.

This architecture leads to independent MIB content, and whenever the MIB content is inconsistent, problems occur that are hard to find and to fix: Crashed nodes are usually still available in the resource management for some minutes – and in some cases, applications are still scheduled with processes on unavailable nodes for hours. MIB inconsistencies are the most annoying effects of non-integrated management architectures, and make the administrators very busy.

3.3.2 Glueware Architecture

In the glueware management architecture, some subsystems (usually the major subsystems[6]) are managed by integrated management software, while the others are still managed by the specific management toolset of the manufacturer. Between the subsystem management applications, interface software (often shell scripts) may be used, which is often developed on-site.

[6] The major subsystems are the nodes, SAN, resource and application management.

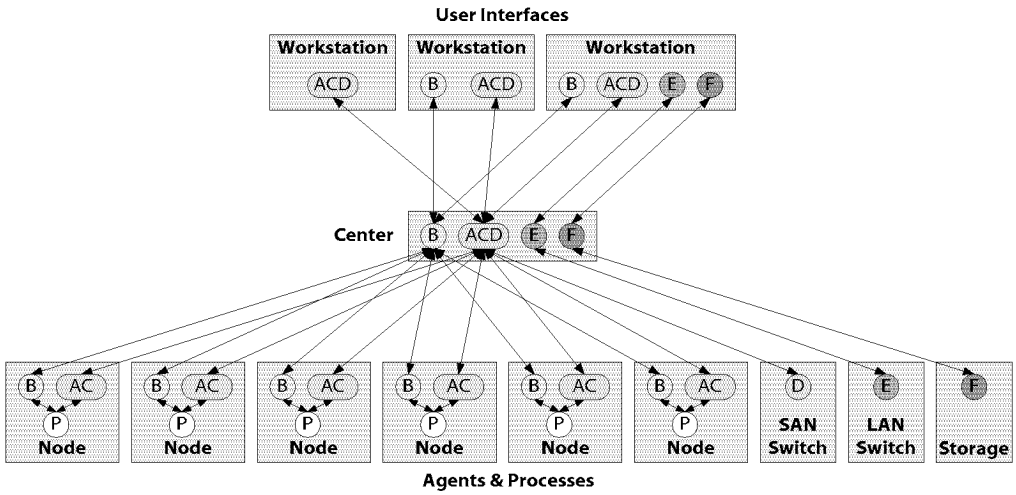


Figure 3.9: Glueware Management Architecture Sample

Since there are still multiple management applications, the users and administrators are forced to use various user interfaces. The integrated subsystems simplify the management, especially the fault handling, where inter-subsystem transactions are required (e.g. shutting down nodes that are connected to a defective SAN switch, aborting the affected applications).

Some of the basic functionality can still be solved using shell scripts (e.g. application spawning or system power control). These tasks are usually executed sequentially with high OS overheads, but the high-overhead jobs can now be executed by the integrated, compiled applications.

Based on the non-integrated cluster from Figure 3.8, Figure 3.9 illustrates a simplified sample supercluster that is managed by three single-subsystem management applications («B», «E» and «F») and one management application integrates the subsystems «A», «C» and «D» (e.g. applications, SAN NICs and SAN switches). Additional management components (scripts, MIB etc.) are omitted in this simplified figure.

- The four center applications run on the center computer.
- The three workstations run the user interfaces that are connected to the center:
  - The left hand workstation runs only the user interface of management application «ACD». Although the user only wants to monitor subsystem «A» (e.g. SAN NICs), additional information about the other subsystems is available.
  - The middle workstation runs the user interfaces of management applications «B» and «ACD». Although the user wants to monitor subsystems «B» and «C» (e.g. applications and their resource storage), additional information about the subsystems «A» and «D» (SAN NICs and switches) is available to him.
  - The right hand workstation runs the user interfaces of all management applications. Due to the integration of three subsystems, only four instead of six applications are run, and display data of the integrated subsystems in context.
- Only two agents are run on the nodes, because application and SAN NIC management is now integrated into one agent application. Both agents observe the processes.
- The agent for the SAN subsystem («D») is integrated into the management software for the subsystems «A», «C» and «D». Since additional features are possible (e.g. application-based SAN monitoring), more functionality can be integrated.

- The agents for the LAN («E») and storage («F») subsystems are the same as those in the non-integrated management architecture.

The reduction of user interfaces simplifies the utilization for the users and administrators. The management integration of the major subsystems into one application reduces complexity and allows functionality that cannot be reached with non-integrated management (such as SAN monitoring data with context to the application). Of course, this requires the integrated subsystems to have a direct relationship.

The evaluation table of this architecture is as follows:

Criteria	Value	System Size [Nodes] (+ 1 Switch per 10 Nodes)			
		100	1 000	10 000	100 000
Required bandwidth for monitoring (64 B/Comp.)	1 Hz	6.4 kB/s	64 kB/s	640 kB/s	6.4 MB/s
	1 kHz	6.4 MB/s	64 MB/s	640 MB/s	6.4 GB/s
Duration startup/shutdown	10 s/Comp.	18 min	3 hrs	1 day	13 days
Duration OS download	30 s/Node	1 hrs	8 hrs	3 days	35 days
Duration group creation	0.1 s/Comp.	11 sec	2 min	18 min	3 hrs
Duration process spawn	1 s/Comp.	2 min	18 min	3 hrs	1 day
Duration fault detection	2 s/Comp.	4 min	37 min	6 hrs	3 days
Duration config change	1 s/Comp.	2 min	18 min	3 hrs	1 day

Table 3.8: Quantitative evaluation table for glueware architecture

Comparing these values with those of the non-integrated management architecture (Table 3.5), it can be seen that there is no advantage to this architecture when considering the quantitative values.

The difference lies in the qualitative evaluation:

Criteria	Quality	Comments
Monitoring Synchronization	0	Global heartbeat and synchronization within integrated subsystems possible
Monitoring Context	0	Context between monitoring data within integrated subsystems possible
Development Time & Cost	0	Requires development of software for the integrated subsystems and interfaces
Management Overhead	+ / –	No additional hardware, many node agents
Reliability & Availability	–	All tools are basically independent and loosely coupled with interfaces
Scalability	–	Limited by the least scalable subsystem management software and shell scripts
Security & Vulnerability	–	The weakest security mechanism of all subsystems defines its vulnerability
Flexibility	0	Quickly adapted to changes, except for re-programming due to component changes

Table 3.9: Qualitative evaluation table for glueware architecture

This architecture has the advantage – compared to the non-integrated approach – that the monitoring quality is increased (with context and synchronization). This is compromised by lower flexibility and higher development cost, due to the development and maintenance of the software of the integrated management subsystems. Also the load on the nodes is lower, since the number of node agents has decreased.

Criteria	Quality	Comments
Design, Installation & Configuration	–	Manual hard work with no computer-aided support, many sources of mistakes
Power Management Implementation	–	Script-based, slow and unreliable
Monitoring Transport to User Interfaces	–	Each tool transports data independently to centers and user interfaces
Process Groups / Routing Tables Creation	– / +	If SAN, node and application management are integrated in one tool, this is simple. If not, it remains impossible
Application & Process Control	0 / +	If application and resource management are integrated, they scale and are reliable
Detection & Handling of Faults	0	Problems are detected quicker in integrated subsystems since they are observed by more instances from different sides

Table 3.10: Typical tasks evaluation table for glueware architecture

The glueware architecture has some advantages compared to the non-integrated approach:

- Reliability and scalability are boosted within the integrated subsystems, but the worst subsystem management software sets the limit.
- Monitoring, fault detection and handling, and trap mechanisms are simple to implement in the integrated subsystems, even if affecting multiple subsystems, since the borders have been broken down.

As with non-integrated management architecture, this architecture is also only suitable for small systems, since the management of the non-integrated subsystems sets the limit.

3.3.3 Integrated Architecture – Basic Version

In the integrated management architecture, all subsystems are managed by the same software, all tasks are performed within the same application, and the potential for efficient, effective and scalable management of superclusters is the highest of all presented architectures.

Figure 3.10 illustrates a simplified sample supercluster where all subsystems are managed by one integrated management application, based on the architectures presented previously.

- The center application runs on the center computer.
- The three workstations run the user interfaces that are connected to the center. Since there is only one user interface, all workstations run the same user interface.
- One agent runs on the nodes, and manages the node comprehensively, observing the processes of parallel applications.
- The agents of the SAN, LAN, and storage subsystem are connected to the center.

Because of the integration, the users are confronted with one single set of user interface applications. This simplification makes the work of the users – and administrators – easier, more efficient, and the user can concentrate more on his scientific work instead of on the complexity of using the supercluster.

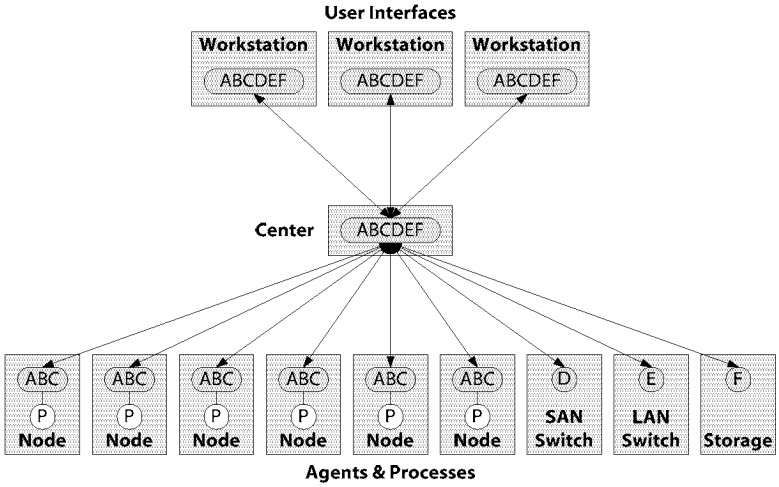


Figure 3.10: Integrated Management Architecture Sample (basic version)

The evaluation table of this architecture is as follows:

Criteria	Value	System Size [Nodes] (+ 1 Switch per 10 Nodes)			
		100	1 000	10 000	100 000
Required bandwidth for monitoring (64 B/Comp.)	1 Hz	6.4 kB/s	64 kB/s	640 kB/s	6.4 MB/s
	1 kHz	6.4 MB/s	64 MB/s	640 MB/s	6.4 GB/s
Duration startup/shutdown	10 s/Comp.	18 min	3 hrs	1 day	13 days
Duration OS download	30 s/Node	1 hrs	8 hrs	3 days	35 days
Duration group creation	0.1 s/Comp.	11 sec	2 min	18 min	3 hrs
Duration process spawn	1 s/Comp.	2 min	18 min	3 hrs	1 day
Duration fault detection	2 s/Comp.	2 sec	2 sec	2 sec	2 sec
Duration config change	1 s/Comp.	2 min	18 min	3 hrs	1 day

Table 3.11: Quantitative evaluation table for integrated architecture (basic version)

Comparing these values with those of the non-integrated and glueware management architecture (Table 3.5 and Table 3.8), it can be seen that there is no major advantage to this architecture when considering the quantitative values – except for the fault detection duration. This short period of time can be achieved because disconnected agents can quickly be detected by the center and defects of the components can be detected by the agents, which inform the center immediately.

The integrated approach has a huge advantage when considering the qualitative evaluations:

Criteria	Quality	Comments
Monitoring Synchronization	+	Full synchronization possible
Monitoring Context	+	Full context possible
Development Time & Cost	0	Requires development of open interfaces, standardization and broad acceptance
Management Overhead	+	One additional server required for center, only one agent on nodes
Reliability & Availability	0	Center is single point of failure
Scalability	0	Limited size, but efficient and effective execution of management tasks
Security & Vulnerability	+	Integrated, monolithic design reduces security hole size and number
Flexibility	0	Limited by the availability of subsystem modules for the center and agent

Table 3.12: Qualitative evaluation table for integrated architecture (basic version)

Because of the integrated design, monitoring is massively improved. This is because a global heartbeat for synchronization is available and full monitoring context is also available (such as the consumed network bandwidth per application process). The integrated design also reduces the number of weak points where attackers can try to intrude into the management system.

Depending on the design of the center and agent applications, the flexibility concerning supercluster component selection varies. If open and standardized interfaces are available, the manufacturer of the component can implement himself center and agent subsystem modules that allow integration into the management software. This also has an effect on the development time and cost of the management software development company or team: Open interfaces can motivate manufacturers to implement the module themselves, allowing the integration of their hardware and software into superclusters that are managed by this center and agent software.

Criteria	Quality	Comments
Design, Installation & Configuration	+	Integration allows integrated design, simulation and installation software
Power Management Implementation	+	Integrated in management software
Monitoring Transport to User Interfaces	0	Center is bottleneck
Process Groups / Routing Tables Creation	+	Integrated in SAN & application modules
Application & Process Control	+	Integrated in node & application modules
Detection & Handling of Faults	+	Detected by agents and center

Table 3.13: Typical tasks evaluation table for integrated architecture (basic version)

The integration allows the design of an additional toolset for automated design, test, simulation installation and configuration of superclusters, making the supercluster management complete. Configuration changes due to application execution (group creation/deletion) or faults are generated on the fly and enforced by the agents immediately. The integration makes many things very simple and comfortable.

The biggest problem of this design is the center. It is the point of failure and bottleneck:

- If the center becomes unavailable or crashes, the management stalls, all applications must be aborted and the supercluster needs to be reset.
- Socket-based Ethernet communication is limited by the OS overhead and design to a few thousand concurrent connections. Since all agents need a stable connection, this leads to a system size limit of a few thousand managed components.

All this makes this architecture suitable for medium-sized systems of up to a few thousand nodes. More nodes break the limit for concurrent connections between agents and center. The available bandwidth for monitoring data between agents and center, as well as center and user interfaces, sets the system size limit to a reasonable size of about 1000 nodes.

3.3.4 Integrated Architecture – Reliable Version

One of the problems of the previous architecture – missing management availability in case of center faults – can be solved by using multiple instances of the center application. The workload is distributed to the available centers which build a reliable cluster. If a center becomes unavailable, the workload is re-distributed to the remaining members, and the agents that were connected to this center become connected to any other available center.

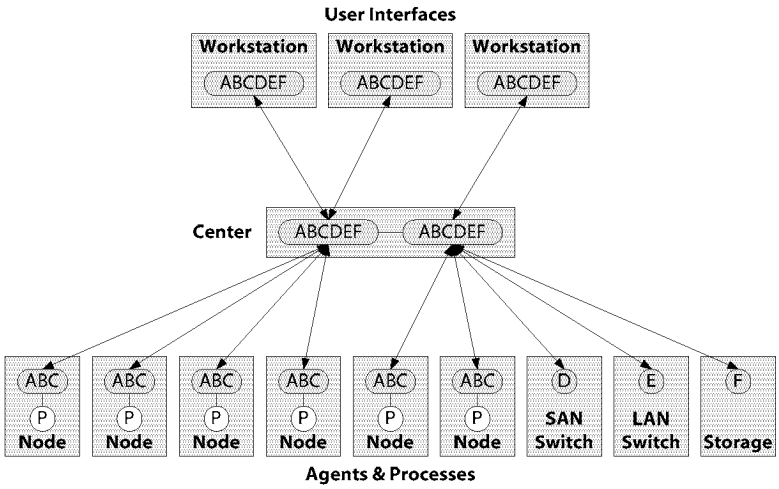


Figure 3.11: Integrated Management Architecture Sample (with clusters)

Membership mechanisms prevent multiple center clusters from trying to manage the supercluster independently. It is usually sufficient for just the cluster with the majority of the potential center cluster members to be allowed to manage the supercluster.

Figure 3.11 is based on the integrated architecture previously detailed, showing an example of a reliable, integrated management architecture.

- The center cluster consists of two centers that are ideally run on separate computers.
- The three workstations run the user interfaces that can connect to any available center.
- All agents connect to the centers, and the centers distribute the agents in such a way that all centers receive a similar workload.

The evaluation table of this architecture is as follows, based on the assumption that there are four members in the center cluster:

Criteria	Value	System Size [Nodes] (+ 1 Switch per 10 Nodes)			
		100	1 000	10 000	100 000
Required bandwidth for monitoring (64 B/Comp.)	1 Hz	1.6 kB/s	16 kB/s	160 kB/s	1.6 MB/s
	1 kHz	1.6 MB/s	16 MB/s	160 MB/s	1.6 GB/s
Duration startup/shutdown	10 s/Comp.	5 min	46 min	8 hrs	3 days
Duration OS download	30 s/Node	13 min	2 hrs	21 hrs	9 days
Duration group creation	0.1 s/Comp.	3 sec	28 sec	5 min	46 min
Duration process spawn	1 s/Comp.	28 sec	5 min	46 min	8 hrs
Duration fault detection	2 s/Comp.	2 sec	2 sec	2 sec	2 sec
Duration config change	1 s/Comp.	28 sec	5 min	46 min	8 hrs

Table 3.14: Quantitative evaluation table for integrated architecture (reliable version)

Compared to the integrated version, the figures are cut to one quarter. It is possible to increase the number of centers in the center cluster much higher than 4, but above a certain number of centers, the efficiency drops because inter-center communication overhead quickly increases.

Criteria	Quality	Comments
Monitoring Synchronization	+	Full synchronization possible
Monitoring Context	+	Full context possible
Development Time & Cost	0	Requires development of open interfaces, standardization and broad acceptance
Management Overhead	+	Additional servers required for centers, only one agent on nodes
Reliability & Availability	+	No point of single failure in design
Scalability	0	Limited size, but efficient and effective execution of management tasks
Security & Vulnerability	+	Integrated, monolithic design reduces security hole size and number
Flexibility	0	Limited by the availability of subsystem modules for the center and agent

Table 3.15: Qualitative evaluation table for integrated architecture (reliable version)

From the qualitative point of view, this reliable design has the advantage that there is no single point of failure, so the probability of a management service crash is very low, increasing the supercomputing service availability.

One additional advantage is that the workload can be distributed to the centers and the management service therefore scales better than that of the previous architecture with one single center – but the small number of centers (ideally below 10) does not reduce the required bandwidth for monitoring and the number of concurrent connections far enough. Additional «tricks» are required to enable scalable management.

One disadvantage is added to the design compared with the one-center approach: The development of reliable cluster software is complex and requires more time and money.

There are no differences in the typical task table when compared to the architecture with only one center, since from a scalability point of view, it is irrelevant whether there is one center or a cluster with ten centers – compared to potential systems with 100 000 or more nodes, this number is very low and monitoring data transport to the user interfaces is still unsolved as is the number of connections that each center must maintain.

Criteria	Quality	Comments
Design, Installation & Configuration	+	Integration allows integrated design, simulation and installation software
Power Management Implementation	+	Integrated in management software
Monitoring Transport to User Interfaces	0	Centers are bottleneck
Creation of Process Groups / Routing Tables	+	Integrated in SAN & application modules
Application & Process Control	+	Integrated in node & application mod.
Detection & Handling of Faults	+	Detected by agents and center

Table 3.16: Typical tasks evaluation table for integrated architecture (reliable version)

For medium-sized systems of some thousand nodes, this architecture is sufficient – and even larger systems are allowed, as long as high-performance monitoring is not required by the users.

3.3.5 Integrated Architecture – Scaleable Version

Bandwidth for monitoring data transportation to the centers and user interfaces, as well as the number of concurrent connections, are the only problems left by the previously presented reliable management architecture using center clusters. Both problems can be solved using scalability-enabling components that act as intermediate centers.

This «middle management» takes commands from the centers, distributes them to the agents, collects data from the agents and forwards compressed information to the centers. These components can be called «management proxies», influenced by the term «proxy» used by other computing technologies.

These proxies can be layered in order to increase the number of components used in a supercluster. With 4 centers in the center cluster, each serving 64 proxies, and each of them serving another 64 proxies that serve 64 nodes, the final supercluster can have 1 million nodes – although managed by 4 164 proxies and centers.

The proxies must support the reliability mechanisms that are used in the center cluster:

- If there is only one center, it is possible to use independent proxies. This approach has not only the disadvantage that the center is a point of failure, but that also every single proxy is a point of failure. The components controlled by the proxy are unmanageable. This approach is therefore only acceptable when there is a small amount of proxies.

- If there are multiple centers in a center cluster, the proxies can also build reliable clusters. Ideally those proxies serve nodes that are topologically close. This approach has the advantage that there is no single point of failure.

The following figures show both approaches: The architecture with individual proxies serving one center (Figure 3.12), and the architecture with a center cluster and proxy clusters (Figure 3.13).

Both approaches reduce the number of concurrent connections to the center, since only the first-level proxies are connected to the centers. The solution for reducing the bandwidth required for monitoring data transportation has already been sketched: The additional connections between the rightmost user interface and the rightmost proxies.

The reason for this is that the proxies can be used to store live monitoring data that is accessed by the user interfaces. The centers only contain concentrated statistical information for archiving and accounting. This reduces the amount of monitoring data sent to the centers. If the user wants to monitor the SAN or his application processes on the nodes, he can connect his user interface to those proxies that manage the nodes that house those processes. Only the bandwidth available from the user interface to the proxies sets the limit.

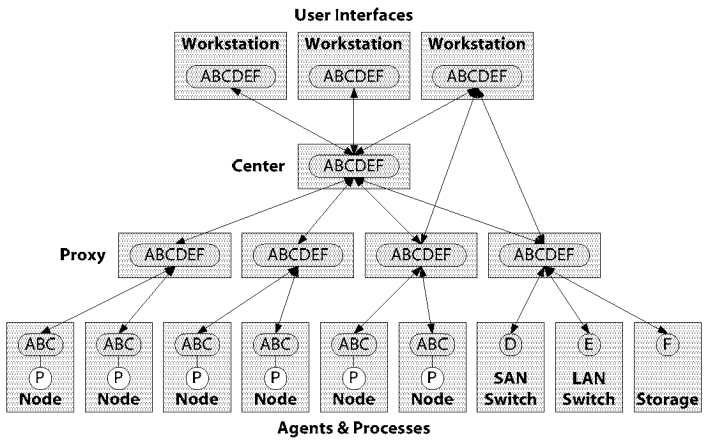


Figure 3.12: Integrated Management Architecture Sample with individual Proxies

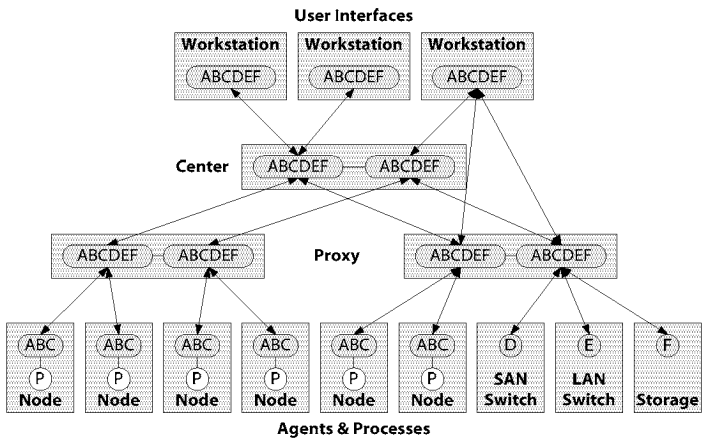


Figure 3.13: Integrated Management Architecture Sample with Proxy Clusters

The following tables assume that there are one and four centers in the center cluster, with 30–40 agents (or proxies) being served by each proxy, and with one and two proxies

per proxy cluster. The tables with one center and individual proxies have systems from 100 to 100 000 nodes, those with the clusters have between 1 000 and 1 000 000 nodes.

Criteria	Value	System Size [Nodes] + Proxies/Layers			
		100	1 000	10 000	100 000
		3 / 1	30 / 1	310 / 2	3103 / 3
Required bandwidth for monitoring (64 B/Comp.)	1 Hz	2.0 kB/s	20 kB/s	200 kB/s	2.0 MB/s
	1 kHz	2.0 MB/s	2.0 MB/s	2.0 MB/s	2.0 MB/s
Duration startup/shutdown	10 s/C + 0.1 s/P	6 min	6 min	6 min	6 min
Duration OS download	30 s/N + 30 s/P	18 min	32 min	37 min	51 min
Duration group creation	0.1 s/C + 0.1 s/P	4 sec	7 sec	8 sec	10 sec
Duration process spawn	1 s/N + 0.1 s/P	34 sec	37 sec	38 sec	41 sec
Duration fault detection	2 s/Comp.	2 sec	2 sec	2 sec	2 sec
Duration config change	1 s/C + 0.1 s/P	37 sec	40 sec	41 sec	44 sec

Table 3.17: Quantitative evaluation table for integrated architecture with proxies and one center

The use of proxies boosts the performance. Under the assumption that each management proxy adds 0.1 seconds of latency (except for the OS download), the duration of all tasks grows in logarithmical relationship to the system size. The bandwidth for monitoring data has been limited artificially for statistical purposes. Start-up and shutdown – as well as the fault detection – are performed by the lower-most proxies. This is the reason why it takes the same amount of time, no matter how many nodes are served.

Criteria	Value	System Size [Nodes] + Proxies/Layers			
		1 000	10 000	100 000	1 000 000
		30 / 1	310 / 2	3100 / 2	31030 / 3
Required bandwidth for monitoring (64 B/Comp.)	1 Hz	2.0 kB/s	20 kB/s	200 kB/s	2.0 MB/s
	1 kHz	2.0 MB/s	2.0 MB/s	2.0 MB/s	2.0 MB/s
Duration startup/shutdown	10 s/C + 0.1 s/P	6 min	6 min	6 min	6 min
Duration OS download	30 s/N + 30 s/P	21 min	34 min	45 min	53 min
Duration group creation	0.1 s/C + 0.1 s/P	5 sec	7 sec	9 sec	11 sec
Duration process spawn	1 s/N + 0.1 s/P	35 sec	36 sec	40 sec	41 sec
Duration fault detection	2 s/Comp.	2 sec	2 sec	2 sec	2 sec
Duration config change	1 s/C + 0.1 s/P	38 sec	39 sec	43 sec	44 sec

Table 3.18: Quantitative evaluation table for integrated architecture with proxy clusters

The clustered version has the advantage, that the centers only serve the directly connected proxies and that the proxies not only serve their lower-level proxies or nodes, but also the other proxies of the same proxy cluster. This advantage shows up in the duration of the above tasks, where 10 times larger systems have similar figures.

The proxy adds scalability, but also implementation costs into the qualitative evaluation table. If there is only one center per center cluster and one proxy per proxy cluster, the reliability drops because of the many points of failure. With reliable clusters, the quality is a lot higher.

Criteria	Quality	Comments
Monitoring Synchronization	+	Full synchronization possible
Monitoring Context	+	Full context possible
Development Time & Cost	–	Development of proxies necessary, and additionally reliability features for clustering
Management Overhead	+	Additional servers required for center, only one agent running on nodes
Reliability & Availability	0 / +	Too many points of failure if only one component per cluster present
Scalability	+	Proxies enhance scalability massively
Security & Vulnerability	+	Integrated, monolithic design reduces security hole size and number
Flexibility	0	Limited by the availability of subsystem modules for the center and agent

Table 3.19: Qualitative evaluation table for integrated architecture (with proxies)

Also the user interface is more complex to implement, because the basic MIB data is still received by the center, but the detailed monitoring data (exceeding the accuracy stored in the center's MIB) must be downloaded from the respective proxies.

Criteria	Quality	Comments
Design, Installation & Configuration	+	Integration allows integrated design, simulation and installation software
Power Management Implementation	+	Integrated in management software
Monitoring Transport to User Interfaces	+	User interface is bottleneck
Process Groups / Routing Tables Creation	+	Integrated in SAN & application modules
Application & Process Control	+	Integrated in node & application modules
Detection & Handling of Faults	+	Detected by agents and center

Table 3.20: Typical tasks evaluation table for integrated architecture (with proxies)

The typical tasks are well supported.

It is evident that the integrated management architecture with proxies and clusters is the crowning design, making it suitable for large systems of up to some 100 000 nodes, and is only limited by available space, energy and money.

3.3.6 Summary and Conclusions

The evaluation tables of the previous subsections can be reorganized in order to compare the architectures directly. The qualitative tables show the advantages and disadvantages of the architectures, the quantitative tables are fixed for some system sizes (100, 1 000, 10 000 and 100 000 nodes) that show the time required for each task.

The table headers use the following abbreviations for the architectures:

- Non-integrated management architecture (N)
- Glueware management architecture (G)
- Basic version of the integrated management architecture (I)
- Integrated management architecture with clusters (C)
- Integrated management architecture with proxies and reliable clusters (P)

In the integrated designs with proxies, there is 1 proxy for 30–40 agents or lower-level proxies. The reliable center cluster consists of 4 members. The architecture with one component per proxy or center cluster is omitted in this summary.

3.3.6.1 Qualitative Evaluation Tables

The following table summarizes the quantitative evaluation of the architectures.

Criteria	Quality per Design				
	N	G	I	C	P
Monitoring Synchronization	–	0	+	+	+
Monitoring Context	–	0	+	+	+
Development Time & Cost	+ / –	0	0	0	–
Management Overhead	+ / –	+ / –	+	+	+
Reliability & Availability	–	–	0	+	+
Scalability	–	–	0	0	+
Security & Vulnerability	–	–	+	+	+
Flexibility	+	0	0	0	0

Table 3.21: Qualitative evaluation summary for all architectures

The advantage of the integrated architectures is obvious. But for small and well protected systems with reduced expectations in application reliability and observability, the glueware and non-integrated management architectures are good enough.

The following table summarizes the typical tasks evaluation of the architectures.

Criteria	Quality per Design				
	N	G	I	C	P
Design, Installation & Configuration	–	–	+	+	+
Power Management Implementation	–	–	+	+	+
Monitoring Transport to User Interfaces	–	–	0	0	+
Process Groups / Routing Tables Creation	–	– / +	+	+	+
Application & Process Control	–	0 / +	+	+	+
Detection & Handling of Faults	–	0	+	+	+

Table 3.22: Typical tasks evaluation summary for all architectures

The advantages of the integrated architectures are also easily visible for the typical management tasks. The glueware architecture is only reasonable, if at least application and SAN management are integrated into one tool.

3.3.6.2 Quantitative Evaluation Tables

The following tables summarize the quantitative evaluation of the architectures for a system size of 100, 1 000, 10 000 and 100 000 nodes. Each table is summarized with a «+/0/–» evaluation in regard to the usability of the architecture for this specific system size.

Criteria	Values per Design				
	N	G	I	C	P
Required bandwidth (1 Hz)	6.4 kB/s	6.4 kB/s	6.4 kB/s	1.6 kB/s	–
Required bandwidth (1 kHz)	6.4 MB/s	6.4 MB/s	6.4 MB/s	1.6 MB/s	–
Duration startup/shutdown	18 min	18 min	18 min	5 min	–
Duration OS download	1 hrs	1 hrs	1 hrs	13 min	–
Duration group creation	11 sec	11 sec	11 sec	3 sec	–
Duration process spawn	2 min	2 min	2 min	28 sec	–
Duration fault detection	4 min	4 min	2 sec	2 sec	–
Duration config change	2 min	2 min	2 min	28 sec	–
Usability	+	+	+	+	

Table 3.23: Quantitative evaluation summary for all architectures with 100 nodes

For small systems of 100 nodes, each of the presented architectures is well suited.

Criteria	Values per Design				
	N	G	I	C	P
Required bandwidth (1 Hz)	64 kB/s	64 kB/s	64 kB/s	16 kB/s	2.0 kB/s
Required bandwidth (1 kHz)	64 MB/s	64 MB/s	64 MB/s	16 MB/s	2.0 MB/s
Duration startup/shutdown	3 hrs	3 hrs	3 hrs	46 min	6 min
Duration OS download	8 hrs	8 hrs	8 hrs	2 hrs	21 min
Duration group creation	2 min	2 min	2 min	28 sec	5 sec
Duration process spawn	18 min	18 min	18 min	5 min	35 sec
Duration fault detection	37 min	2 sec	2 sec	2 sec	2 sec
Duration config change	18 min	18 min	18 min	5 min	38 sec
Usability	–	0	0	+	+

Table 3.24: Quantitative evaluation summary for all architectures with 1 000 nodes

For medium systems of 1 000 nodes, scalability already plays an important role, since startup and shutdown takes some hours and regular tasks some minutes without integration and proxies.

Criteria	Values per Design				
	N	G	I	C	P
Required bandwidth (1 Hz)	640 kB/s	640 kB/s	640 kB/s	160 kB/s	20 kB/s
Required bandwidth (1 kHz)	640 MB/s	640 MB/s	640 MB/s	160 MB/s	2.0 MB/s
Duration startup/shutdown	1 day	1 day	1 day	8 hrs	6 min
Duration OS download	3 days	3 days	3 days	21 hrs	34 min
Duration group creation	18 min	18 min	18 min	5 min	7 sec
Duration process spawn	3 hrs	3 hrs	3 hrs	46 min	36 sec
Duration fault detection	6 hrs	6 hrs	2 sec	2 sec	2 sec
Duration config change	3 hrs	3 hrs	3 hrs	46 min	39 sec
Usability	–	–	–	–	+

Table 3.25: Quantitative evaluation summary for all architectures with 10 000 nodes

For large systems of 10 000 nodes, the integrated architecture with proxies is required. Scalability, security and reliability are important and cannot be provided by the other architectures.

Criteria	Values per Design				
	N	G	I	C	P
Required bandwidth (1 Hz)	6.4 MB/s	6.4 MB/s	6.4 MB/s	1.6 MB/s	200 kB/s
Required bandwidth (1 kHz)	6.4 GB/s	6.4 GB/s	6.4 GB/s	1.6 GB/s	2.0 MB/s
Duration startup/shutdown	13 days	13 days	13 days	3 days	6 min
Duration OS download	35 days	35 days	35 days	9 days	45 min
Duration group creation	3 hrs	3 hrs	3 hrs	46 min	9 sec
Duration process spawn	1 day	1 day	1 day	8 hrs	40 sec
Duration fault detection	3 days	3 days	2 sec	2 sec	2 sec
Duration config change	1 day	1 day	1 day	8 hrs	43 sec
Usability	–	–	–	–	+

Table 3.26: Quantitative evaluation summary for all architectures with 100 000 nodes

For huge systems with 100 000 and more nodes, only architectures with layers of proxies and reliable clusters are adequate. Availability is very important, since many processing hours are lost if the management becomes unavailable for one minute or crashes.

3.3.6.3 Conclusion

The following graph shows how a suitable architecture may be selected:

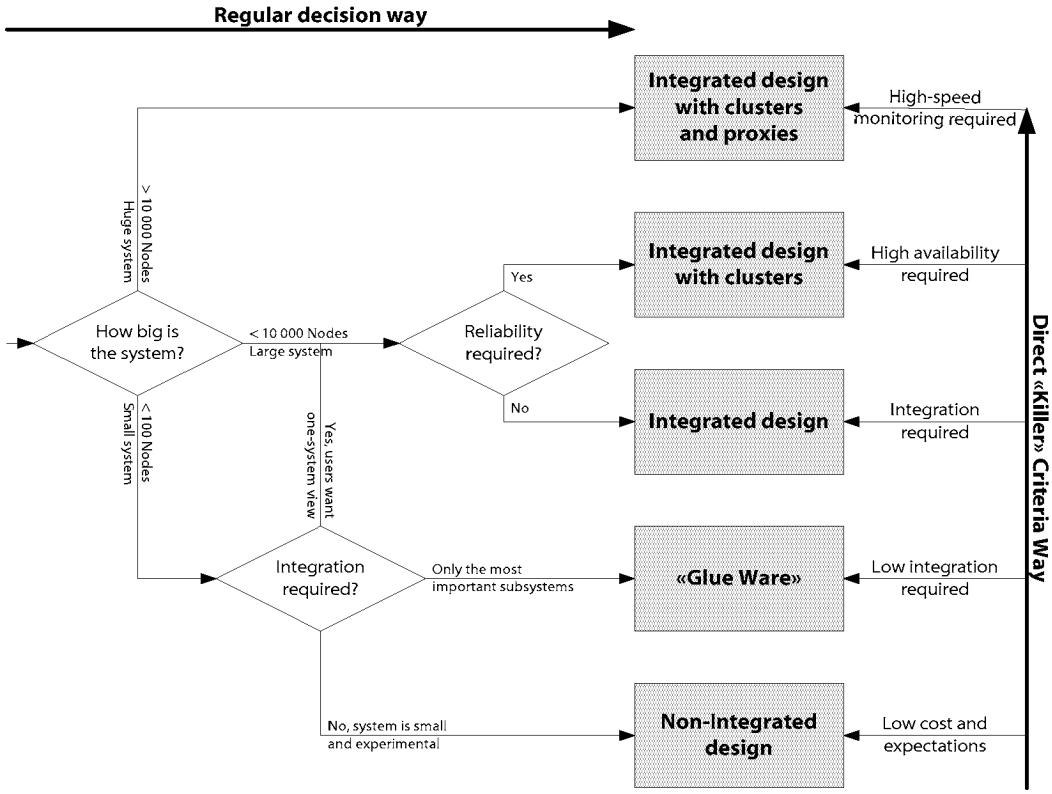


Figure 3.14: Quick guide to finding the right management software architecture

3.4 Management Component Design

The previous sections presented the software components of the supercluster management software (3.1) and the software system architecture (3.3). This section presents the internal structure of the software components on a theoretical, qualitative level. Although the illustrations of this section suggest an integrated architecture, the presented structures can be used for all software system architectures.

3.4.1 Basic Component Design

Large software products are preferably designed in a modular fashion, with a framework and (independent) modules that communicate using documented interfaces. Depending on the computer platform where the software will later be executed, the availability of development tools, and the skills of the software developers, various technologies can be used for creating software products.

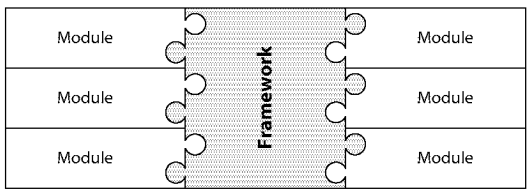


Figure 3.15: Basic modular component design

The framework interconnects the connected modules, allowing fast and simple interactivity. The documented and standardized interface between the framework and modules – together with an open architecture – allows third-party modules to integrate easily. This is similar to the driver software strategy of modern operating systems, where the drivers of hardware manufacturers usually integrate easily into the computing system.

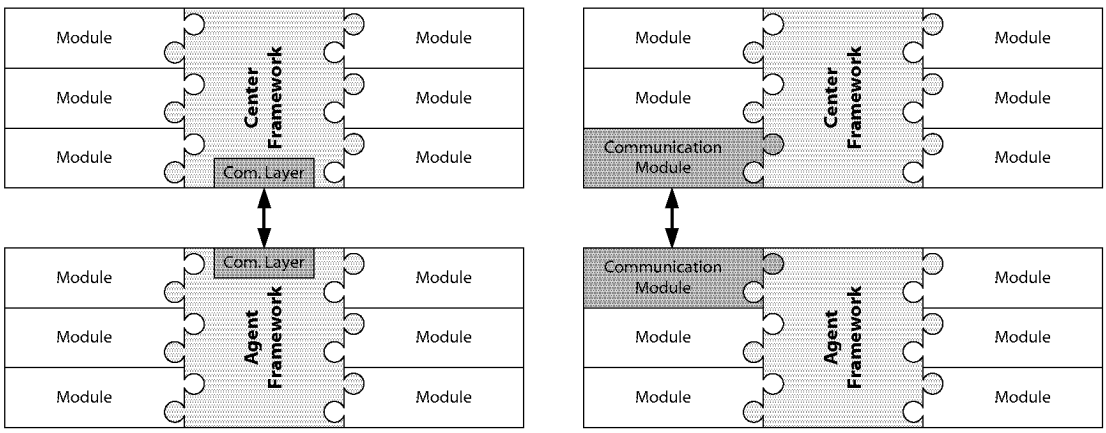


Figure 3.16: Communication strategies between components, using an in-framework communication layer (left) or a dedicated communication module (right)

In hierarchical software architectures, the frameworks of the components communicate with each other – either directly through an integrated communication layer, or by using a dedicated communication module connected to the framework.

For the management software, there is a relationship between the modules of the center framework and the modules of the agent frameworks: The center module that manages a subsystem communicates through the frameworks with the agent module that actually performs the actions requested by the center module.

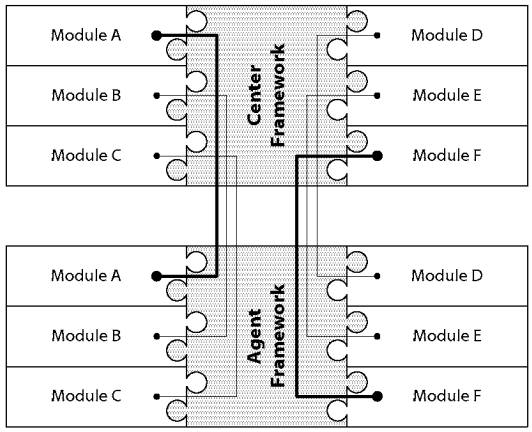


Figure 3.17: Relationship between center modules and agent modules

Whereas the center manages all subsystems, the agents only manage their associated components. This is the reason why the center framework has all subsystem management modules connected to it, and the agent framework on the other hand has only those subsystem management modules connected to it that the managed component is part of: The node is usually part of all subsystems and has therefore almost all modules connected to the node agent's framework, whereas the SAN switch agent usually has only the management module for managing the SAN switch connected to it (plus possibly the resource and application management module).

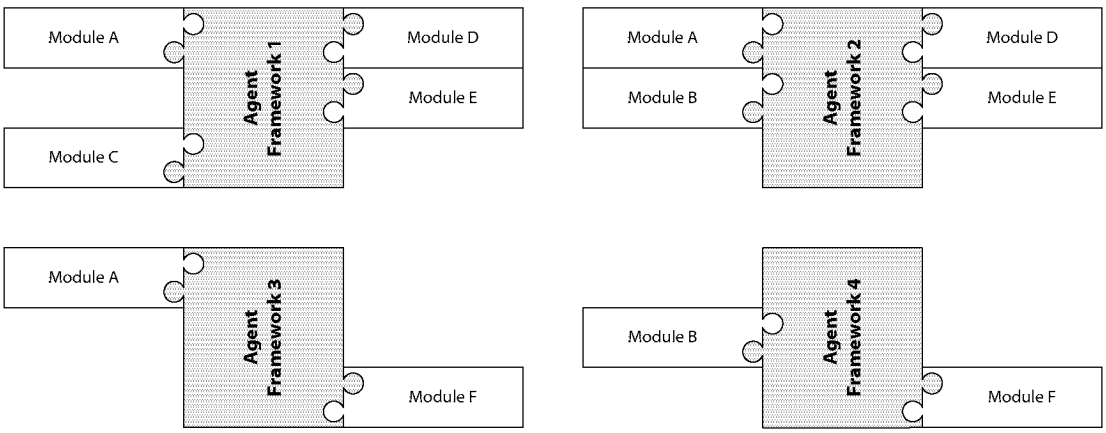


Figure 3.18: Agent frameworks with only those modules connected that they actually need

The interface between the framework and all modules is bi-directional, where the framework offers functionality to the module, and the module offers functionality to the framework and all other connected modules (illustrated as the two «puzzle knobs» in

the previous figures). Only this bi-directionality allows the interactive integration of subsystem management as required by the integrated or glueware architectures.

3.4.2 Management Center Design

The management center is the point of authority in the supercluster. It contains all management modules, since it manages all subsystems centrally, and it contains additional modules that are required for further features and integration into other higher-level management systems. The center framework interconnects all center modules.

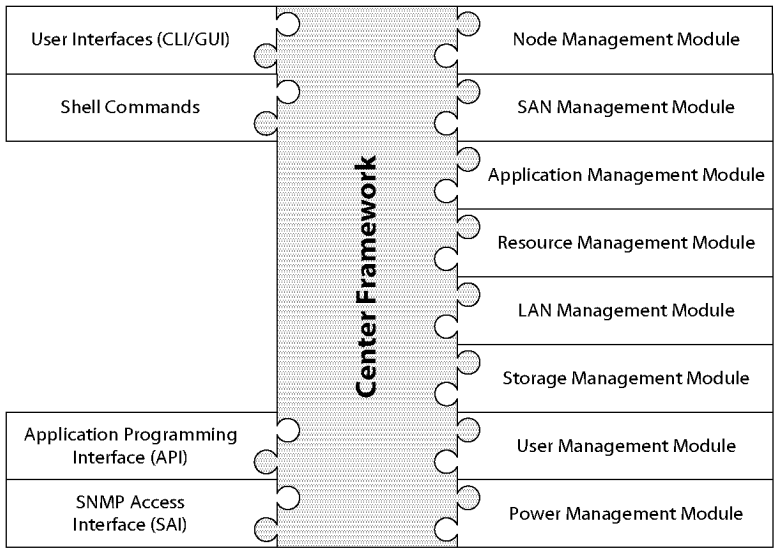


Figure 3.19: Software structure of the supercluster management center software

Figure 3.19 shows a sample design of the management center software with all subsystem management modules (right), and additional modules (left).

The **User Interface Module** handles the messages that are received from the connected user interfaces. They hide the implementation of the MIB (usually speed-optimized, object-oriented data structures for fast and efficient access) and present the data in a structured way (usually hierarchical, such as the SNMP MIB). Command line interfaces (CLI) usually have «data pull design» (the user types a request and expects a response), whereas graphical user interfaces (GUI) usually have «data push/pull design» (the center can additionally send messages that are not intended by the user, such as failure notices and other interactive requests). Both data handling designs must be implemented in this module.

The **Shell Command Module** allows simple management interaction without the need for the use of structured user interfaces (CLI/GUI). For shell scripting in UNIX environments, it is often sufficient to permit management actions by using shell commands: Opening or closing queues, aborting or submitting jobs, etc. The users are used to this user interface, since most management tools available for supercomputers and superclusters also offer this interface.

The **Application Programming Interface** allows third-party software manufacturers to use the functionality of the management software. External higher-level management tools, that manage whole farms of superclusters, may have an influence on the supercluster managed by this management software. In the simplest case, the external software only gathers monitoring data for web-based services (e.g. current usage and performance), in the most complex case, the external software controls a large computing environment, which the supercluster is only one part of.

The **SNMP Access Interface** is a tribute to the computing de-facto management standard SNMP, allowing SNMP user interfaces to access the MIB of the supercluster managed by the management software. Since SNMP is not secure, only monitoring should be allowed.

The framework for the management center holds a virtual representation of the whole supercluster in its MIB, potentially shared with additional instances of the center. It knows the **strategy** («what to do») for managing the supercluster, but the subsystem management modules hold the component-specific **tactical** («how to do») information.

- The **Node Management Module** manages all nodes of the supercluster, handles messages of the node agent's node management module, and forwards tasks received by the framework or other management modules to the node agent's management module.
- The **SAN Management Module** manages all SAN switches (using the SAN switch agents) and the SAN NICs (using the node agents), handles the agents' messages concerning SAN components and forwards requests from other modules and the framework.
- The **Application Management Module** manages the applications and their parts, and the processes run on the nodes. It handles the messages received by the application management modules running on the nodes and forwards tasks received by the framework and other management modules.
- The **Resource Management Module** manages the queues and their availability to applications and users. It only handles messages received by the framework and other modules, since it has no agent module.
- The **LAN Management Module** manages the LAN components (using the LAN agents) and the LAN NICs (using the node agents), handles the agents' messages concerning LAN components and forwards requests from other modules and the framework.
- The **Storage Management Module** manages the storage subsystem using either dedicated storage agents or node agents, depending on its implementation (storage agents if using an independent storage subsystem, node agents if using in-node storage devices for a distributed file system).
- The **User Management Module** manages the users, user groups, projects and their credits and permissions. It handles messages received by the agents' user management modules, and the center framework and its modules.
- The **Power Management Module** manages the power supply of all supercluster components. Depending on the interface design between managed component and associated agent, the power subsystem uses external power switches or addi-

tional software components, if the agent is shut down together with the managed component[7].

Most of the management modules have a corresponding party on some of the agents. Some of the management modules are center-based without agent modules – these modules do not usually manage real hardware, but a virtual «soft» subsystem that is used for planning or accounting.

The job of the framework is not just gluing the modules together. It has a complex list of tasks:

- It manages the MIB, no matter if implemented as a persistent database or in the main memory.
- It maintains the connections to all other management components.
- It reacts to faults, traps, and other events that require management action. Such actions are performed by the framework, which delegates the real work to the center's management modules, which in turn send messages to their agent modules. This behavior of the framework is stored in the MIB, and is thus manageable by the administrators.
- It communicates with the administrators and users using the user interfaces.
- It reacts when the management itself has problems (crashing centers, proxies, agents), and takes appropriate action. These actions are stored in the MIB, and are thus adjustable by the administrators.

The framework is a complex part to design and implement, and takes most of the development time.

3.4.3 Management Proxy Design

For the proxies, the management modules have the task of gathering and concentrating monitoring information and other data from the agents for the center, and for distributing messages from the center to the agents. Since user interfaces and shell commands for monitoring connect to the proxies to gather monitoring data from their MIB, there are also modules for the user interfaces and the shell commands, similar to those of the management center.

The modules are simplified compared to the agent or center modules, since they only collect and distribute data, but do not perform actions received by the center, nor decide anything based on the messages received from the agents.

The only exception is the power management module that can connect to components to switch their power on and off or start up or shut down their services.

[7] Some components have a permanently running console server, allowing powering up even if switched off. Some components require a wake-up IP packet. Other components require an external remote power switch.

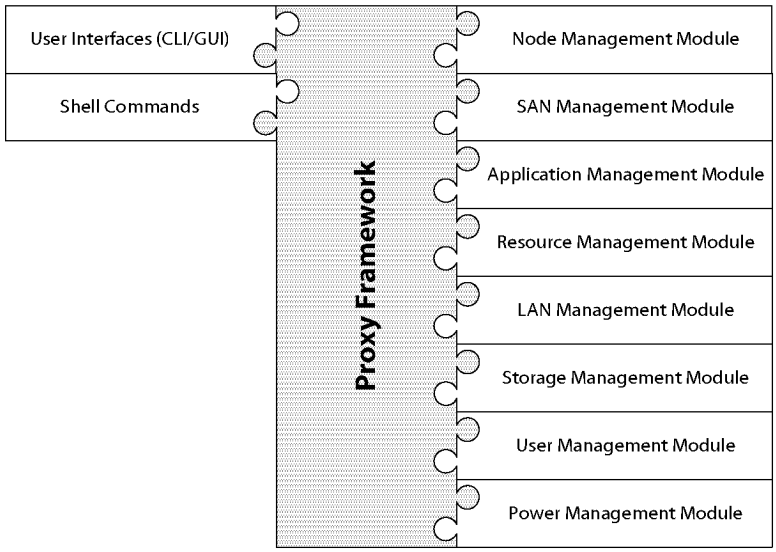


Figure 3.20: Software structure of the supercluster management proxy software

3.4.4 Management Agent Design

The framework of the management agents keeps the virtual representation in the center MIB identical with the physical reality. The connected subsystem management modules know how to perform these tasks – the division between the strategic framework («what») and tactical modules («how») is present here also.

The management modules actually perform the «physical task» that was ordered by the center module: They change the configuration, abort the application, shut down the node, report faults, and send monitoring data.

The agent module is specifically implemented for the component technology in use, and it is replaced when the component is upgraded. Of course, this simple replacement is only possible when using open interface architectures for integrating third-party modules to the framework.

The agent framework has only those subsystem management modules connected that are indeed managed by the agent. The following illustration shows four sample agents (node agent, SAN switch agent, storage agent and LAN agent) with only one to six connected subsystem management modules.

Additionally, the other modules that were present in the center and proxy are not used in the agent. Since the user interfaces and shell commands do not communicate with the agent directly (only via the center as its single authority), there are no user interface modules. Also third-party software products communicate with the API and SNMP module of the center only.

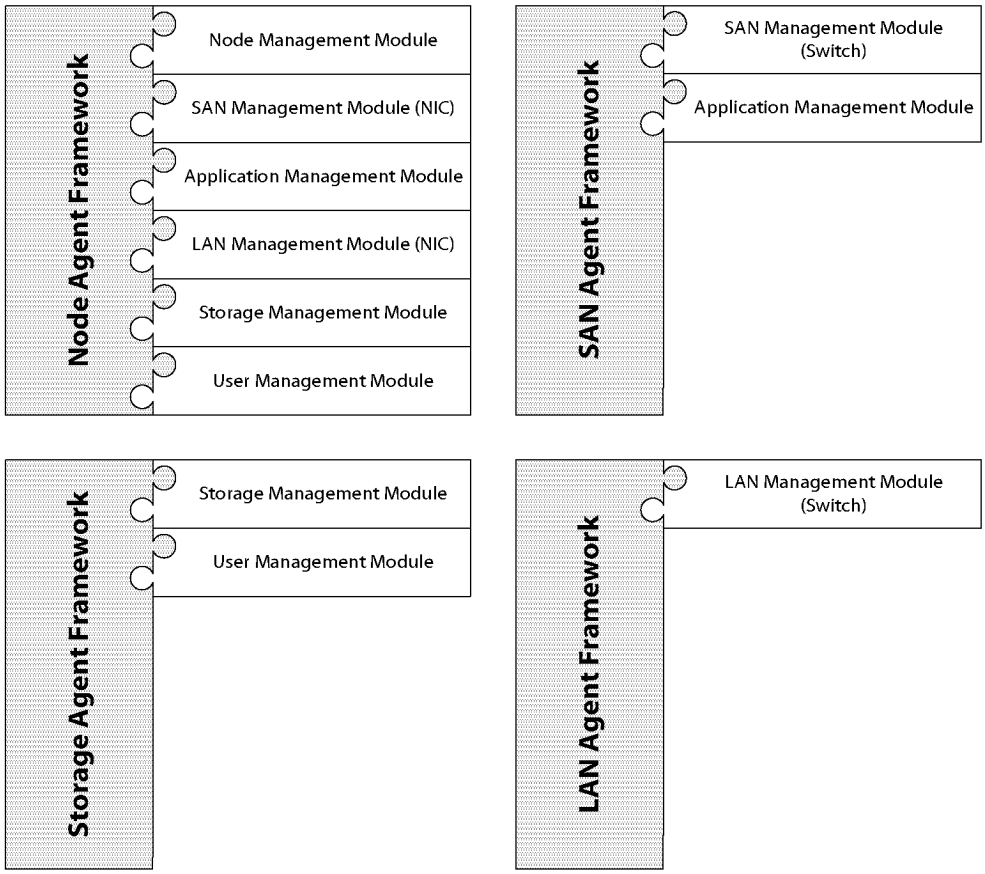


Figure 3.21: Software structure of the supercluster management agent software

The modules are specifically designed for their agent usage:

- The **Node Management Module** communicates with the OS of the node, gathers all information for the center and manages the node in accordance with the center's order.
- The **SAN Management Module** on the node agent communicates with the driver of the SAN NIC, gathers data for the center and manages the NIC in accordance with the center's order. The module on the SAN switch agent communicates with the SAN switch firmware, using either a communication channel (LAN, console, serial port) if the agent is run externally or with the switch OS directly if the agent is run on the SAN switch.
- The **Application Management Module** on the node agent starts application processes, manages them and sends signals to control them. It gathers their data and sends it to the center and controls the processes in accordance with the center's order. The application module on the SAN agent has the goal of managing application-specific configuration changes (such as routing table entries) which need to be removed after finalization or abortion.
- The **LAN Management Module** on the node agent manages the LAN NICs and their configuration using the OS. The LAN module on the LAN agent manages the LAN components directly (if run on the component) or using a communication channel (LAN, console, serial port, SNMP) if run externally. Both collect data for the center and perform the actions in accordance with the center's order.

- The **Storage Management Module** is located either on the node agent (if a distributed file system using the node's storage devices is being used) or on a dedicated agent (if an independent subsystem is being used). Both collect data for the center and perform actions in accordance with its orders.
- The **User Management Module** on the node agent enforces the user permissions to the application processes owned by that particular user. This enables user-specific environments and reduces vulnerability (e.g. if the processes are being run with root permissions). The user management module is also useful for the storage agent, when quotas of users, projects or departments must be maintained or file permissions are to be enforced.

The modules can have multiple threads, where some threads execute the messages received from the framework, some threads communicate with other parties (e.g. OS or device driver), and other threads are regularly invoked for checks and cleanup purposes.

A lot of care must be taken in the design and implementation of the node agent modules and framework. The performance loss due to management must be minimal, since the node is primarily used for calculation. Additionally, the management load must be homogenous when considering geographical distribution (all nodes have the same load) and temporal distribution (all nodes receive the additional load at the same time). The slowest node defines the maximum performance.

3.4.5 Management Action Examples

The following examples illustrate how management actions are distributed between the management modules of the center and agents, and how they interact to solve problems in common management tasks.

3.4.5.1 Example 1: Application spawning

The user has entered an application into the queue and the resource management has blocked the requested amount of neighboring nodes for this application. When the time arrives, the application will need to be started on the supercluster.

- The resource management detected that the application is scheduled for startup and performs some checks before it indeed starts the application.
  - It checks the MIB to see if the process slots on the projected nodes are empty.
  - It checks to see if there are enough free SAN NIC channels for the processes.
- The resource management requests the application management to start the application.
  - The center application module sends the spawn message to the agent modules.
  - The agent application module downloads the application executable and additional files as described by the user at job submission.
  - All application processes are started on all nodes and go into the startup barrier. The application is ready to start.
- The other subsystems are informed that the application is about to start.
  - The SAN subsystem receives information about the application (e.g. application ID, SAN NIC channels of the processes on the nodes). This is required for

the SAN switch and NIC routing tables and other features such as broadcasts or multicasts.
  - The node management receives the process IDs of the processes on each node. This is required for the monitoring of the application operation and resource usage.
- If all subsystems are ready for the application, the processes are allowed to start.

During execution, the application processes are closely coupled to the node agent, exchanging management messages (such as group creation that requires routing table changes on the SAN switches) and standard I/O. The node agents also monitor the processes. The management messages of the application are first handled by the agent management modules, but are usually later forwarded to the center management modules that decide what to do.

3.4.5.2 Example 2: Crash of an application

Failing applications are unfortunately a known and common scenario. Programming errors, data errors and hardware faults can cause a process to abort, and since all processes are required for regular execution, the whole application needs to be wiped out of the supercluster.

- One process of a parallel application detects a serious problem that requires an abortion of the whole application, thus aborting all processes of the application.
  - If the process detects problems, it sends an abortion request to the node agent.
  - If the process catches a signal, it either sends an abortion message to the node agent (if handled by the signal handler) or just disappears (e.g. kill signal).
  - The process can just exit, disappearing from the node.
- The node agent receives either an abortion message from the process, or it detects a connection drop with the process. The node then sends an abortion message to the center.
- The center decides to abort the application and advises its subsystems.
  - The event is logged into the system management journal.
  - The application management module has to wipe out the affected application.
  - The SAN management module has to clean up the SAN.
  - The resource management is informed that the application is aborting.
- The center management modules perform the requested actions.
  - The application module requests the node module to abort the application.
  - The SAN module requests the switch and node module to remove all messages of this application from the network, as well as all its routing table entries. Also the SAN NIC channels used by the application processes are freed.
  - The resource module reorganizes the scheduling map to reduce the number of unused time slots.
- The node agents remove the application processes and all dependencies.
  - The application module aborts the processes by sending abortion signals.
  - The SAN NIC module closes the NIC communication channels for those processes, and removes any temporary configurations associated with the application.
- The SAN switch agent also removes any temporary configurations and messages associated with the application.
- All agents report the success of their actions to the center.

- The resource management module is allowed to use the available process slots.

The application abortion mechanism is also used in cases, where the application itself is working well: The application has used up all reserved time, the nodes need to be shut down because of a fault or trap, etc.

3.4.5.3 Example 3: Creation of groups

The programming environment MPI (and other environments too) allow the creation of process groups. The processes within this group can use group-internal synchronization and communication mechanisms such as barriers or broadcasts. These group-internal mechanisms are usually supported with SAN features that require temporary configuration changes or extensions.

- The process sends a group creation request to the node agent.
- The node application module forwards this request to the center.
- The center application module forwards the request to the center SAN module.
- The center SAN module creates the new routing table entries for SAN switches and NICs and sends them to the SAN switch agents and node agents.
- The SAN modules of the SAN switch agents enter the new entries into the routing table, and the SAN modules of the node agents adapt the SAN NIC configuration.
- After the agents confirm that the changes have been applied, the center SAN module returns the multicast handle to the center application module.
- The center application module forwards this handle to the node application modules.
- The node application modules forward this handle to the processes.

The groups can also be deleted, requiring the routing table entries to also be deleted.

3.4.5.4 Example 4: Crash of a node

In large superclusters, nodes crash regularly, caused by hardware faults or operating system problems. Restarting or shutting down a node is a frequent task.

- The node management module of the node agent either detects a problem which is then forwarded to the center's node management module, or the node just crashes.
- The node management module of the center either receives the message or detects a disconnected node agent. A disconnected node agent has a certain time to reconnect before it is considered to have crashed.
- The center node module informs the other subsystems that the node has crashed.
  - The resource management removes the node from the resource pool, and reorganizes the applications in the resource map.
  - The application management aborts all applications with processes on that node. It also requests the resource management to restart those applications.
  - The SAN management removes the node from the routing tables.
  - The node management tries to restart the crashed node.
- The center forwards the messages to the agents.
  - The application modules of the nodes abort processes of affected applications.

- The SAN NIC module of the node agent adapts the SAN NIC configuration based on the routing tables received by the center.
- The SAN switch agent adapts the routing table based on the center data.

- As soon as the node is operational again, the node is entered into the node pool of the resource map and the applications are re-organized again.

The applications that were restarted because of the node crash are usually restarted from the last complete checkpoint. In contradiction to the previously presented application crash (where the application is not restarted), the applications were working well and the applications are safe to continue operation on a new set of nodes.

3.4.5.5 Example 5: Crash of a SAN Switch

One SAN switch has crashed and cannot be restarted. In this case, all connected nodes need to be deactivated, the applications currently running on these nodes need to be re-started on a new set of nodes, and the neighboring SAN switches need new routing tables.

- The framework of the center holds the supercluster topology and knows which switches to re-configure and which nodes to take out of the pool and to switch off. It transfers this information to the application, resource, node and SAN management modules of the center.
  - The resource management module stops submission and takes the nodes, with their slots, out of the pool. It also finds a new set of nodes for the re-submitted application and schedules it for execution from the last complete checkpoint.
  - The application management module aborts the application and re-submits it.
  - The SAN management module calculates new routing tables for all SAN switches and turns off the crashed SAN switch.
  - The node management module shuts down and switches off the nodes that are connected to the crashed SAN switch.
- The center framework receives the messages from the management modules and forwards them to the respective management agents.
- The framework of the node agent receives the messages from the center framework and forwards them to the respective agent management module.
  - The application management module aborts those application processes, which belong to the applications running on the nodes connected to the crashed SAN switch.
  - The node management module shuts down and turns off the node, if the center has decided that this node is to be shut down and switched off.
  - The SAN management module adapts the routing table in the SAN NICs.
- The framework of the SAN switch agent receives the messages from the center framework and forwards them to the respective SAN switch management module (usually the SAN management module only).
  - If the center has decided that this SAN switch is to be shut and powered down, it will do so.
  - If the message contains routing changes, the routing table will be adapted.
- After all the tasks have been successfully completed, the queues are enabled again.

One issue is open in this action: There is a possibility that additional applications stall because of the crashed SAN switch. When application processes are distributed in the supercluster topology and the communication between those processes passes through the affected SAN switch, it is possible that those messages will be lost, leading to hanging processes waiting for messages that never arrive. This issue can be handled by a mechanism implemented in the application management, which detects applications without consumption of CPU time, aborting them after a certain time. This is a nice demonstration, where a hard-to-detect side-effect of a problem of one subsystem management (SAN switch crash causes hanging applications) is solved by a mechanism of another subsystem management (hanging application are aborted).

3.4.5.6 Example 6: Supercluster start-up

Starting up the supercluster is a standard, but not often executed task. Its goal is to make the supercluster fully operational as fast as possible.

- The management is first started up, since except for the center (or one center, if there is a cluster), all other supercluster and management components are powered off.
  - If there is a center cluster, the center which is running powers on all other centers that are powered off at the moment. All centers are required to start up the supercluster. All centers wait until enough center demons (to reach the quorum) have started.
  - The centers power on all proxies and wait until they have started the proxy demon.
- Now the supercluster subsystems are started in a controlled fashion.
  - The proxies and centers start all computers that act as agents (except for the nodes).
  - The agents start the components that they manage (LAN, SAN, and storage). It is often necessary to have the subsystems already running when the nodes start.
- Finally, the nodes are started.
  - Since starting all nodes at once may create a power bounce, the start-up is staggered, starting each part individually, waiting a few seconds, starting the next part etc.
  - When the nodes have been booted, the node agent configures the subsystems for operation (e.g. downloading firmware to the SAN NIC, installing the basic configuration).
- When the supercluster is running, the management checks its configuration.
  - The supercluster is checked to see if all components to be managed are present.
  - During a system downtime, servicing tasks by administrators and field service personnel are performed, such as replacing defective hardware, or updating software. The management software schedules some applications that check the current status.

When everything is working well, the supercluster is opened for the users and scheduled jobs.

3.4.5.7 Example 7: Supercluster shut-down

Shutting down the supercluster works in the opposite way to starting it up.

- For a regular shutdown, scheduling for all application queues is stopped.
- As soon as all jobs have finished, the node agents shut down and switch off the nodes.
- The agents shut down the components and the subsystems that they manage. Finally, they also shut down the computers that they are running on.
- The proxies shut down the computers they are running on.
- If the centers form a cluster, the main center continues to run and all other centers shut down the computers they are running on.

There is of course an emergency scenario, e.g. if there is a fire or unexpected power loss.

- The centers abort all applications currently running on the supercluster. Alternatively they receive a checkpoint signal (to write their current status on persistent storage for later continuation from that status).
- The nodes are shut down and switched off as soon as the applications have finished.
- The agents shut down the components they are managing and switch off the computer they are running on.
- The proxies shut down the computers they are running on.
- If the centers form a cluster, the main center continues to run and all other centers shut down the computers they are running on. Alternatively, also the main center shuts down.

Shutting down the whole supercluster is useful for downtime tasks (such as hardware upgrade, replacing defective hardware, re-routing cables, dust removal, etc.). For large superclusters, it may be reasonable to allow the shutting down of currently unused parts of the supercluster to save electrical energy (of the nodes and of the air conditioning).

4 Superclusters Worldwide – Competing Technologies

The secret to creativity is knowing how to hide your sources.

Albert Einstein

Supercomputing or managing supercomputers is nothing new. The structure and the size of the supercomputers have changed, requiring new strategies and products to manage these systems. Comparing the strategies and implementations created elsewhere with the presented concepts and problems of this thesis, it is possible to find new solutions and to create new designs that respect both the special requirements of commodity massive-parallel supercomputing and the gained experience and knowledge of similar existing solutions.

4.1 Management of the ASCI Supercomputers

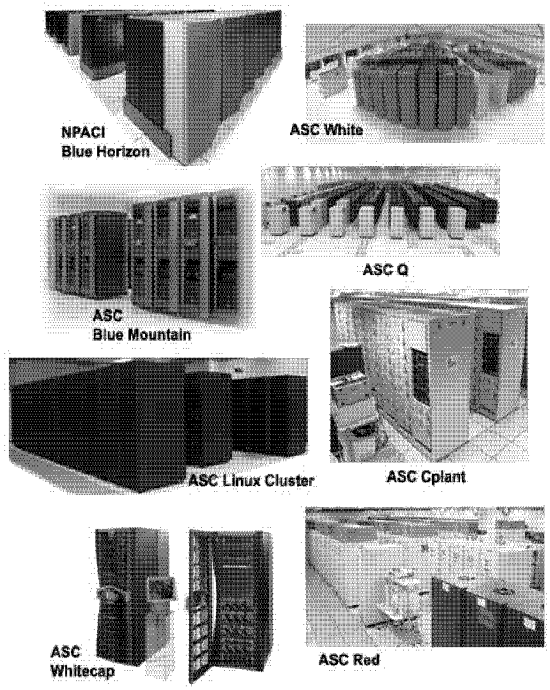


Figure 4.1: The ASCI Supercomputer Family (February 2004)

The management software used for the ASCI supercomputers is usually an extension of the software already available for the specific platform, except for the supercomputers that are not based on commercially available components. Unfortunately most information is classified since the computers are also used for secret applications.

- IBM extended their basic packages *LoadLeveler*, *Parallel Environment* (PE) and *Parallel System Support Tools* (PSSP) for the ASCI White and ASCI Blue Pacific.
- SGI/Cray extended the *Advanced Cluster Environment* (ACE) for the ASCI Blue Mountain.
- Sandia developed custom management software for the ASCI Cplant.
- Intel extended the *Desktop Management Interface* (DMI) for the ASCI Red.

In the following, the management software approaches are presented (as far as the information is available and allowed to be published). The management software of the ASCI Red is presented in detail, since great effort has been put into its design and implementation.

The following table summarizes the four ASCI supercomputer specifications.

Criteria	Red	White	Blue Pacific	Blue Mountain
Integrator	Intel	IBM	IBM	SGI
Installation year	1997	2000	1999	1998
Installation site	SNL	LLNL	LLNL	LANL
Peak performance [GFLOPS]	3 207	12 288	3 857	3 072
Sustained performance [GFLOPS]	2 379	7 304	2 144	1 608
Processor type	Pentium Xeon	Power3	604e	R10000
Processor frequency [MHz]	333	375	332	250
Processor count	9 632	8 192	5 856	6 144
Interconnect	Custom	SP Switch	SP Switch	HIPPI
Topology	2½D mesh	Fully switched	Fully switched	Crossbar
Main memory [GBytes]	1 212	8 192	2 600	1 536
Storage memory [TBytes]	12.5	160	82.5	76
Operating system	Custom	AIX 5.2	AIX 5.2	IRIX
Management software	DMI-type	PSSP 3.5	PSSP 3.5	AS, LSF
Management architecture	Integrated	Non-integrated	Non-integrated	Glueware
Footprint [m^2]	233	1116	900	930
Energy consumption [kW]	1 200	2 000	486	1 600
Cost [million US$]	55	110	54	78
Weight [t]	44	106	53	
MTBF [h]	10	40	22	20

Table 4.1: Specifications of the first-generation ASCI supercomputers

4.1.1 ASCI White & Blue Pacific (IBM, Lawrence Livermore NL)

During the last decade, IBM built several software packages for management and administration of their high-end AIX-based servers. Since the RS/6000 SP-type servers are based on independent nodes that are interconnected using a switch, there is no big leap toward superclusters.

The management is based on the following products:

- **IBM AIX Operating System**
  The AIX operating system is the UNIX implementation of IBM for their workstations and servers. As in many other UNIX implementations, it contains custom features for automated remote management as well as an adaptable SNMP agent.
- **IBM PE (Parallel Environment) for AIX**
  The PE software contains the libraries and applications required to develop parallel applications. It contains the MPI (Message Passing Interface) library, math libraries, parallel profilers and benchmark tools, and checkpoint/restart enhancements required for large and long-running jobs.
- **IBM LoadLeveler**
  The LoadLeveler software evolved from CONDOR and is the resource and batch management application for AIX-based computers. It includes the gang scheduler, the workload manager, the checkpoint/restart management and the file system monitor.
- **IBM PSSP (Parallel System Support Programs) Toolkit**
  The PSSP tools provide a comprehensive suite for installation, operation, management and administration of AIX-based servers from a single point of control.

Every SP server is managed by a control workstation which runs a managing demon, which communicates with the server components through Ethernet and a serial link. It monitors the cabinet, the nodes and the SAN switch (power, fans, and temperature) and it can control the power of each component separately. It executes commands and logs messages using the logging service.

The system configuration of every SP computer is entered using the hardware management console (HMC) which allows the servers installed at a location to be graphically managed. For a homogenous pool of servers, the HMC may be extended to manage all servers of a supercomputer. The administrators developed a web-based monitoring service open to the registered users. The users basically use shell commands for the management of their applications.

There are three software packages managing the supercluster. A web-based monitoring user interface displays current data. The software design scheme is the non-integrated variant as described in section 3.3.1, with the extension of the HMC that resembles the functionality of management proxies.

4.1.2 ASCI Blue Mountain (SGI/Cray, Los Alamos NL)

The Cray supercomputers are the best known supercomputers worldwide, with their unique shape and uncompromised search for processing performance. The supercomputer knowledge was integrated into new products that allowed SGI to enter into high-end technical computing.

For the management of their large servers and clusters, SGI developed the Advanced Cluster Environment (ACE) that provides an integrated and comprehensive management solution. ACE is used in the ASCI Blue Mountain supercomputer and in the server clusters for technical and business computing installed worldwide. It provides a single system view from a single point of administration, and manages all (SGI) hardware and software installed.

ACE consists of the following components:

- **SGI IRIX Operating System**
  The IRIX operating system is the UNIX implementation of SGI for their workstations and servers. As with many other UNIX implementations, it contains custom features for automated remote management as well as an adaptable SNMP agent.
- **SGI Message Passing Toolkit (MPT)**
  The Message Passing Toolkit (MPT) integrates the most used standardized message passing libraries. This allows the users and developers to easily port and create applications.
- **Job Management (SGI Performance Co-Pilot, LSF)**
  The Performance Co-Pilot allows performance problems to be analyzed in parallel systems. The cluster-wide performance, resource utilization, activity and bottlenecks are analyzed from one single point of administration. The Load Sharing Facility (LSF) is a third-party product (from Platform Computing), used for the job scheduling and management.
- **SGI Array Services and RoboInst**
  The Array Services are a scalable and highly available solution for large-scale computing. RoboInst allows software to be installed and maintained on a set of computers.

Of course, the administrators of the ASCI Blue Mountain developed their own applications on top of ACE for their daily tasks.

The management software ACE works like glue between the components, and the software architecture can therefore be described as «glueware».

4.1.3 ASCI Red (Intel, Sandia NL)

Although the ASCI Red supercomputer is based on commodity CPUs, memories and storage devices, the architecture and design is very customized, using custom boards, interconnections, libraries and operating systems. The management software «Scalable Platform Services» (SPS) is based on the Desktop Management Interface (DMI) [DMI], which has been designed for the management of networked desktops.

Two processors are grouped in a node and two nodes are placed on a node board. Eight node boards are managed by one Patch Support Board (PSB) which also manages the backplane of this group. Four groups are placed in one cabinet, which leads to a total of 76 cabinets, placed in a 4×19 grid, and matching the SAN topology (2½D grid, 2×32×38 node boards) very well.

The supercomputer is managed by a management station running Microsoft Windows NT, which exports the functionality to administrators through CLI- or GUI-based management applications. The following illustration shows the architecture of the system management.

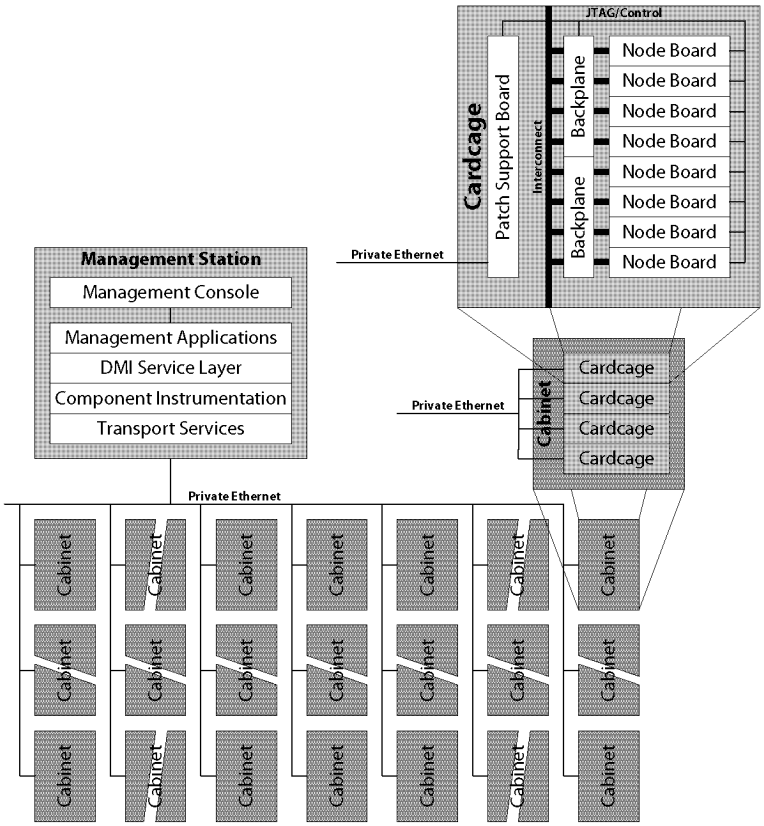


Figure 4.2: SPS Architecture [Mit98]

The automation functionality of SPS allowed the supercomputer to be built, installed and configured within a tight schedule. The functionality includes the following:

- *Scripted Booting/Shutdown* allows the starting or stopping of the supercomputer partitions and the booting of one of the two available operating systems installed on the machine.
- The *Fault Management* is based on the software agents. If they detect a hardware or software failure, the affected components will be isolated and automatic recovery operations will be initiated and all available information will be stored in the log.
- The *Configuration Management* is also based on software agents that keep the hardware and software inventory data up-to-date.
- The *Repair Services* include script-based support for board repair, power control, firmware upgrade and hardware reset operations.
- The *Field Diagnostics* include script-based support for encapsulated diagnostic test scenarios, covering many platform hardware components.
- The *Operating System Console Access* allows the consoles to be accessed from remote clients.

The management software operates as a 32-bit Microsoft Windows NT application and has the following structure:

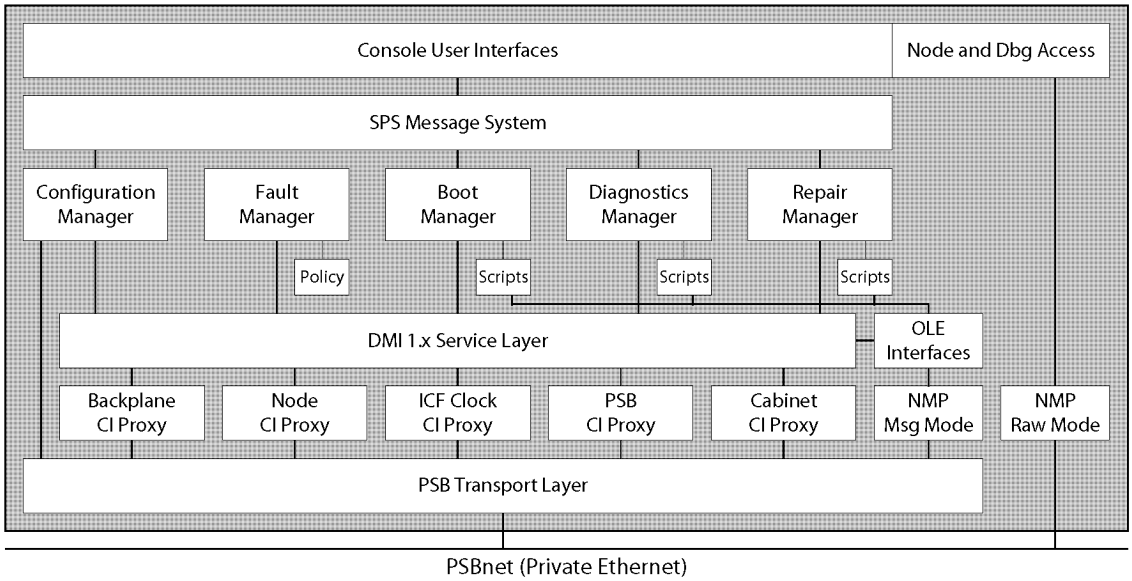


Figure 4.3: Management Software Structure & Components [Mit98]

The *Console User Interfaces* run either on the management workstation itself or on a remote console. They communicate through the SPS message system to the managers. In the GUI, the supercomputer is shown physically as four rows of 19 cabinets each. An additional view allows zooming into the cabinets, looking at the components within, and creating selections for actions.

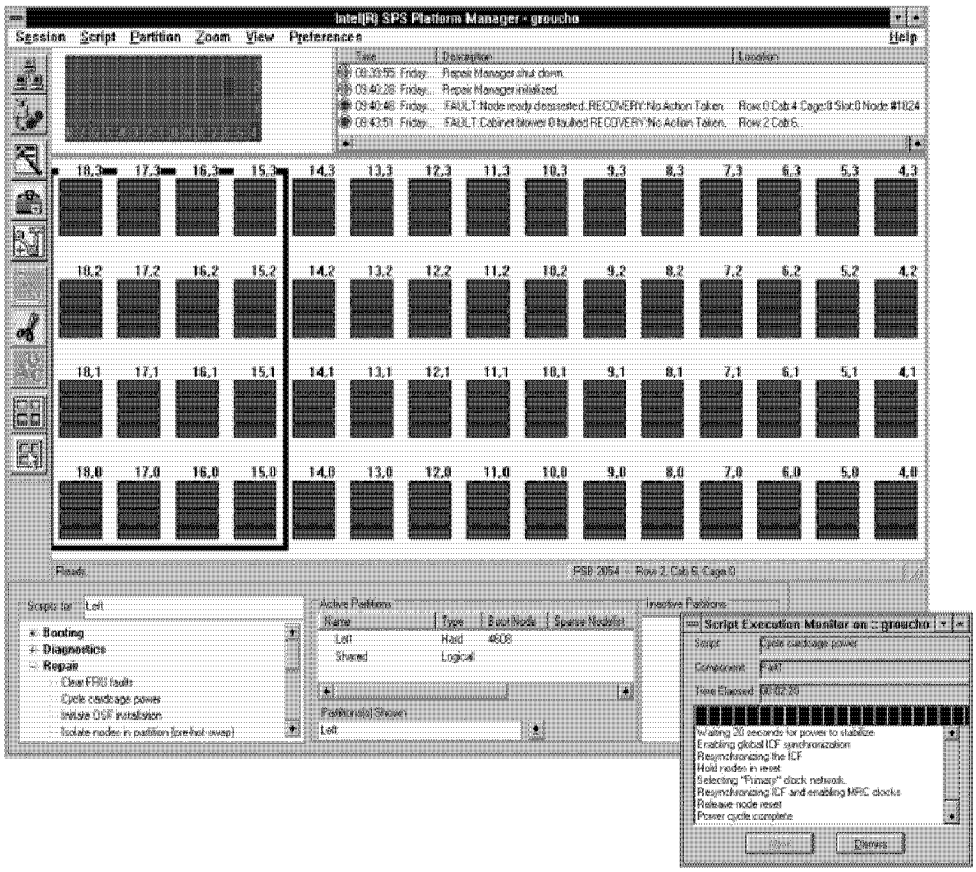


Figure 4.4: Management GUI of the ASCI Red supercomputer [Mit98]

The SPS Message System allows reliable datagram communication between the (potentially) remote consoles and the SPS managers. Its main purpose is to support user interfaces on computers other than the management station. It also contains authentication mechanisms to restrict administrative access in accordance with the membership in security groups.

The managers operate as separate NT services on the management station. The Boot, Diagnostics, Fault and Repair Managers advertise a list of control operations, which are backed by a script. The scripts are triggered by the administrators, except for those of the Fault Manager, which are executed automatically.

The Desktop Management Interface (DMI) contains the DMI database and provides the illusion of managing a single, unified system. The supercomputer consists of 3600 components, resulting in 21500 DMI groups with 99000 DMI attributes. An extension was necessary, since the standard DMI only supported 254 components. Building the database takes 15 minutes, compared to 2 days without further scalability enhancements.

The DMI standard defines a Component Instrumentation interface, which performs the required low-level management operations. Since the operations are performed on the PSB, Component Instrumentation Proxies (CSP) are needed. The five proxies (for nodes, backplanes, PSBs, clock and cabinets) run as separate NT services and communicate with the corresponding agent on the PSB for data collection and management requests.

The lowest layer is the PSB transport layer, which operates as an NT service. It provides reliable datagram communication between the proxies and one or more of the PSBs. The PSB transport layer also detects when a PSB drops off the network and triggers a fault.

The developers of the management software found the following challenges:

- The most pervasive challenge was the integration of a diverse collection of off-the-shelf software components.
- Another problem was the lack of available target hardware, leading to a specific design for the ASCI Red with few testing platforms.
- A variety of scalability challenges were encountered, which forced the developers to design the software carefully to prevent performance bottlenecks.
- For the installation, support for manufacturing tests was needed, requiring incomplete versions of SPS for booting, power control, diagnostic and other features. This required trade-offs between feature availability, stability and progress towards the final product.
- In large systems, the secondary management problem – managing the management software and hardware itself – is as difficult as the primary management problem – managing the supercomputer. Installation, upgrading and debugging of management software on the PSBs and assignments of IP addresses proved to be non-trivial tasks in practice.
- The selection of the off-the-shelf components has a significant effect on the product quality, system availability and management software project schedule.
- The acceptance of Windows NT was a cultural barrier within supercomputing centers, since traditional management environments are based on UNIX operating systems. These concerns were largely ameliorated by providing UNIX-like commands and a telnet server on the management station.

The SPS management software was able to profit from the following issues:

- The administrators were hand-picked and acquired in-depth knowledge of the ASCI Red.
- The supercomputer is homogenous in respect of used components.
- The size of the supercomputer is fixed and will not change.

The SPS played an important role in enabling Intel to win the teraflop performance race.

With the adapted DMI architecture, the management software architecture resembles the integrated version with the PSBs operating as proxies.

4.1.4 Summary

The ASCI-like supercomputers are homogenous systems installed by the node manufacturers. They provide the most integrated system management software: Like a suit, the management software is tailored to the underlying platform and it does not fit any other body. It is not possible to easily replace any individual product, because the software is not modular. It is also not possible to massively increase the system size, since the management software runs at its scalability limit.

The management software packages of the ASCI supercomputers have something in common: The complaints of the administrators that are described in section 2.4 (compare the values with the numbers in Table 3.25):

- Monitoring is only possible at a very slow rate and the data is without context.
- Boot takes between 4 and 8 hours.
- The management software is the cause of 50% of the crashes.
- Updating the system with a new boot image takes eternities and is unreliable.
- Complex group routing mechanisms in the SAN are not supported.
- Starting applications take up to one hour.
- The MTBF is less than 10 hours.
- Lost processes block the system, requiring regular reboots of the whole computer.

The administrators and users feel lucky, but they are far from being confident.

4.2 Management of Clusters of Workstations

Clusters of workstations are currently managed with the following strategies:

- Each manufacturer supplies management software for his subsystem only, leading to the non-integrated software architecture, since the administrators have to put those subsystem management packages together using scripts and other self-made tools.
- One manufacturer supplies management software for his own and some associated subsystems (e.g. SAN, node and applications), while the other subsystems are managed using tools of their respective manufacturer. This strategy leads to the glueware architecture.

- Existing management software used for one subsystem is extended towards an integrated management application, e.g. SNMP or resource management applications.

In the following subsections, these strategies are briefly described and evaluated.

4.2.1 Quadrics RMS (Resource Management System)

Quadrics Supercomputers World is better known under the former name *Meiko* that built parallel supercomputers such as the CS-2. The architecture of RMS (Resource Management System) and its GUI resemble the management software of the Meiko CS-2, which allows RMS to be seen as its further development.

RMS is used for superclusters that use the SAN technology *QsNet* [Qua00] like some Alpha-based superclusters. RMS itself has been extended to cover more than the SAN only. Unfortunately, QsNet only supports fat tree topologies (with a fan-out of 64), which are good in performance, but bad in system scalability and price.

RMS is based on a persistent database (that can be accessed using SQL statements and is controlled by the *Transaction Log Manager*) and a set of manager demons that can run on the same or different computers to distribute load. The manager demons have the following jobs:

- **Machine Manager**
  The *Machine Manager* is responsible for detecting and reporting changes in the state of each node computer in the system. It records the current state of each node and any changes in state in the persistent database. When a node is working correctly, the node demon will periodically update the database. However, if the node crashes or IP traffic to and from the node stops, these updates will stop.
- **Partition Manager**
  The system can be divided into partitions, groups of nodes that can be used for user jobs. Each partition is controlled by a *Partition Manager*. The partition manager mediates each user request for resource availability to run jobs in the desired partition.
- **Switch Network Manager**
  The *Switch Network Manager* controls and monitors the data network. It checks for network errors and can monitor network performance. If it detects an error in the switch network, it will update the status of the concerned switch and generate an event. It collects fan, power supply and temperature data, and generates appropriate status changes and events if components fail or temperatures exceed their allowed limits.
- **Event Manager**
  The *Event Manager* executes recovery scripts that will either correct detected faults or report them to the operators if manual intervention is required. On receiving an event notification, the *Event Manager* looks for a matching entry in the event handler's table, executing the handler script if it finds a match. If no match is found, it runs the default script.

Applications are spawned using a mpirun-like application called *prun* that distributes and starts the processes – the MPI information is transmitted through environment variables (size, rank) and the manufacturer-supplied SAN-specific MPI implementation (routing information to the other participating processes). The resource management is not performed by RMS itself but is outsourced to any batch management software package suited to the users.

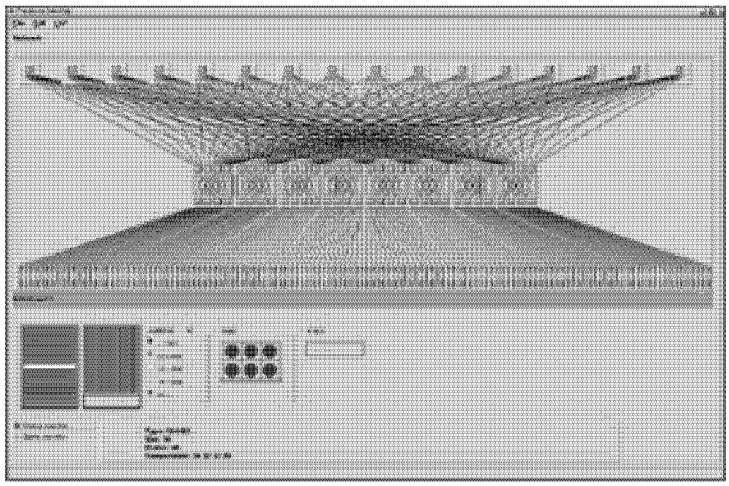


Figure 4.5: Pandora, the user interface of RMS

RMS integrates SAN management, node monitoring and application management (as COSMOS, see 5.2), resembles the glueware software architecture as presented in 3.3.2. The selected design has the following advantages and disadvantages:

- The distribution of the modules into separate demons has the following effects:
  - The risk of a total management functionality outage is reduced, since the demons can run on separate computers, which also allows for workload balancing.
  - Since all demons are expected to run, one failed demon (or crashed computer) blocks the management, since redundancy is currently not featured in RMS.
  - Each server demon has a connection to each workstation demon, leading to a huge number of connections for large systems.
- The persistent database has the following effects:
  - MIB data is not lost if demons crash or are restarted.
  - The MIB can be accessed using queries with standard SQL statements.
  - The persistent database is a single point of failure.
  - Inter-demon synchronization mechanisms are necessary to prevent inconsistency.
- The system size is limited since the design does not scale (no proxies, no mechanisms for high availability).

The evaluation tables of this architecture are as follows:

Criteria	Quality	Comments
Monitoring Synchronization	0	Synchronized node/SAN subsystem
Monitoring Context	0	Context in node/SAN subsystem
Development Time & Cost	+	Integration of in-house products
Management Overhead	+	No additional hardware and software
Reliability & Availability	–	One broken demon blocks management
Scalability	–	Limited by demon communication
Security & Vulnerability	–	Only SAN is fully managed
Flexibility	–	Only works for QsNet and Alpha systems

Table 4.2: Qualitative evaluation table for RMS

Criteria	Quality	Comments
Design, Installation & Configuration	–	No support
Power Management Implementation	–	Script-based, slow and unreliable
Monitoring Transport to User Interfaces	0	Efficient transport to Pandora
Process Groups / Routing Tables Creation		Not necessary because of Fat Tree
Application & Process Control	+	Integrated
Detection & Handling of Faults	0	Only SAN and node faults are detected

Table 4.3: Typical tasks evaluation table for RMS

Extending the management software used to manage the nodes or SAN is a complex task and often ends up with the selection of only a few additional products to manage. This then leads to closely coupled platforms such as UNIX-based COMPAQ AlphaServers and Quadrics SAN products. Replacing subsystem products with other manufacturer's products is hard or impossible, requiring homogenous monolithic systems.

The biggest disadvantage of RMS is the agent software design: The agent is part of the operating system kernel, and each OS version and patch requires re-implementation of the agent software.

4.2.2 Supercluster Management using Resource Management

There are numerous resource management tools available today, with their origins in the batch management of supercomputers. They optimized the supercomputer usage, and in «networks of workstations», they catch wasted cycles in workstations for the execution of parallel applications.

Three resource management software packages are commonly used today:

- The open source product **PBS** (Portable Batch System) and the extended commercial version **PBS Pro**.
- The commercial products **GRD** and **Codine** that are planned to go open source.
- The commercial product family **LSF** (Load Share Facility).

These program packages usually contain the following functionality:

- Combining available calculation resources (processing elements in supercomputers, individual computers) into virtual parallel supercomputers.
- Offering these calculation resources to users for parallel applications.
- Limiting the access to known users, groups and projects.
- Building queues, where users can submit their applications and these jobs are processed in a defined order using priorities and other scheduling mechanisms.
- The consumed resources are logged and accounted to the user, group or project.

These packages are very powerful and include parts of application and node management besides the essential resource management features. It seems to be a good idea to extend the existing and established products with the system management features into integrated and comprehensive system management software.

Since resource management software is not designed for system management, many changes are necessary for the integration, but some products are open enough (and the sources are available) to allow this integration. There are the following advantages and disadvantages:

- The existing framework with managers and agents has the following effects:
  - Existing frameworks with basic functionality are available which save time.
  - Remote management using shell commands and GUI applications is available.
  - Designed for low overhead on the node computers.
  - Platform interoperability (UNIX and Linux, some support Windows).
- The available and missing functionalities of the toolset have the following effects:
  - Node monitoring and fault handling is often already integrated, as well as parts of the application management (spawning, abortion, checkpointing).
  - Adding more features may be hard, depending on design and implementation.
  - There are usually no features for high availability, reliability and scalability.
- High cost for the development of the new features, leading to high license fees.
- Large installed base and therefore easy market penetration.

Depending on the integration grade, this approach resembles the (simple) integrated or glueware architecture with its advantages and disadvantages.

The evaluation tables of this architecture are as follows:

Criteria	Quality	Comments
Monitoring Synchronization	0	Synchronization in integrated subsystems
Monitoring Context	0	Context in integrated subsystems
Development Time & Cost	0	Dependent on external implementation
Management Overhead	+	No additional hardware or software
Reliability & Availability	0	Dependent on implementation
Scalability	–	Limited by communication overhead
Security & Vulnerability	0	Integrated in design
Flexibility	0	Supports many platforms and systems

Table 4.4: Qualitative evaluation table for Resource Management

Criteria	Quality	Comments
Design, Installation & Configuration	–	No support
Power Management Implementation	–	Script-based, slow and unreliable
Monitoring Transport to User Interfaces	0	Efficient transport to GUI
Process Groups / Routing Tables Creation	–	SAN management not supported
Application & Process Control	+	Integrated
Detection & Handling of Faults	0	Only node and application faults detected

Table 4.5: Typical tasks evaluation table for Resource Management

4.2.3 SNMP-Based Supercluster Management

Network management is one of the basic needs where computers are used and interconnected. Many hardware components are designed for networking: Ethernet network adapters, switches, hubs, routers and many more. In the beginning, every manufacturer supplied their product with specialized software for the management of the components. Incompatibility of hardware and software slowed down the use of networking in business, until standardization committees started to introduce networking standards that were respected by the manufacturers.

The management of the networking components was somehow not standardized, because every network component manufacturer believed his product to be special and unique. The network management protocol SNMP (Simple Network Management Protocol), introduced by Marshall T. Rose and Jeffrey D. Case [Ros96], was an attempt to standardize the management protocol chaos. This standard developed into a de-facto standard, followed by the manufacturers.

SNMP basically follows the idea of a «dumb component» that contains «useful stuff», an instrumentation layer and the management protocol that includes the management information (the MIB) and the transport protocol. The network management station accesses the management information by use of the transport protocol, and processes and displays its data to the users.

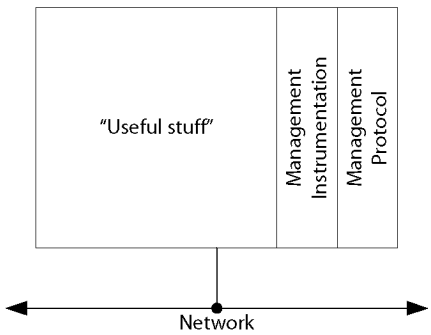


Figure 4.6: General management model [Ros96]

The SNMP MIB is basically a tree of values, which is accessed using structured names. The MIB entries are accessed individually or in bulk containers. The values are read, written, created and deleted using messages. The data is transported over the UDP/IP protocol, an unreliable datagram transportation protocol.

Faults are handled with trap messages that are sent to a specified destination computer. This trap is only handled if the destination computer receives the message, which is not guaranteed using the UDP/IP protocol.

Security in SNMP is not effectively implemented. Identification and protection of the MIB is only implemented using a common «community» identifier between the network management station and the managed node. Encryption of messages (planned for SNMP V2) is not available.

This is the reason why most components support SNMP for monitoring only, since «showing up already solves 80% of the problems». Configuration using SNMP is often supported, but the most vital parts of the component are managed either using telnet-based or HTTP-based user interfaces or private extensions of SNMP for security.

SNMP is used for supercluster management in several places [SMF00] with an extension: SNMP proxy computers (intermediate level manager) allow scalability and some values in the MIB represent the result of an application or script.

Extending SNMP for supercluster management has the following issues:

- The usage of a standardized and accepted product has the following effects:
  - The administrators already have the required know-how.
  - There are many open-source frameworks, applications and user interfaces available that allow SNMP-based management components to be developed.
  - Known security mechanisms and shortcomings allow intrusions.
- The utilization of SNMP has the following effects:
  - SNMP is modular and would allow extensions for supercluster management.
  - SNMP has no security mechanisms (encryption, authentication, integrity)
  - Malicious software that acts as manager can harm the agents, since the manager actively connects to the agents without any communication security.
- Without custom proxies, SNMP does not scale for large superclusters.
- SNMP does not contain mechanisms enabling high availability and reliability.

The evaluation tables of this architecture are as follows:

Criteria	Quality	Comments
Monitoring Synchronization	–	Unavailable in design
Monitoring Context	0	Implementation-specific
Development Time & Cost	0	Special features need time
Management Overhead	+	No additional hardware and software
Reliability & Availability	–	Unreliable center-based transportation
Scalability	–	Proxies required, unavailable in design
Security & Vulnerability	–	Unavailable
Flexibility	+	Supports many platforms and systems

Table 4.6: Qualitative evaluation table for SNMP

Criteria	Quality	Comments
Design, Installation & Configuration	–	No support
Power Management Implementation	–	SNMP- / Script-based, slow and unreliable
Monitoring Transport to User Interfaces	–	GUI pulls data, inefficient
Process Groups / Routing Tables Creation	0	Logic must be implemented in manager
Application & Process Control	–	Unavailable
Detection & Handling of Faults	0	Possible though SNMP traps

Table 4.7: Typical tasks evaluation table for SNMP

In theory and with a lot of work, SNMP-based applications can be extended to the integrated architecture with reliable clusters and proxies. In practice, the efforts required are too great compared with creating a new management system from scratch.

4.2.4 Script-based Supercluster Management

Most superclusters today – mainly small systems installed in the research institutes of universities and industry – are managed manually using shell scripts: *Perl*, *Crontab* and *Expect* are the environments used for this type of management. For the Swiss-T1, some tasks are also solved with scripts, such as starting and stopping the COSMOS node agent demons on all nodes, booting and shutting down the node computers or controlling the power state of the SAN switches.

Managing superclusters with scripts has the following advantages and disadvantages:

- Simple to use since UNIX administrators know scripting very well.
- Basic tasks are simple to achieve, first progress is very fast.
- Fast implementation adaptation to changes in the supercluster configuration.
- The complexity grows with the number of possible tasks.
- Fault handling is hard to achieve in monitoring-only scripting management.
- Scripts are unreliable.
- Debugging scripts is difficult.

The evaluation tables of this architecture are as follows:

Criteria	Quality	Comments
Monitoring Synchronization	–	Unavailable
Monitoring Context	–	Unavailable
Development Time & Cost	0 / –	Quick prototype, bad debugging
Management Overhead	+	No additional hardware and software
Reliability & Availability	–	Shell-Scripts are unreliable
Scalability	–	Shell-scripts do not scale
Security & Vulnerability	–	Unavailable
Flexibility	+	Supports many platforms and systems

Table 4.8: Qualitative evaluation table for scripting

Criteria	Quality	Comments
Design, Installation & Configuration	–	No support
Power Management Implementation	–	Script-based, slow and unreliable
Monitoring Transport to User Interfaces	–	GUI scripting is slow and inefficient
Process Groups / Routing Tables Creation	–	Unavailable
Application & Process Control	–	Unavailable
Detection & Handling of Faults	–	Serial and slow

Table 4.9: Typical tasks evaluation table for scripting

Script-based management is a bad idea. For some tasks, it is an efficient tool, but it cannot be used for comprehensive management such as described in this thesis.

4.3 Conclusions

Analyzing the related work and comparing it with the research of the previous chapters, the following observations can be made and conclusions drawn:

- The management of the ASCI supercomputers is adapted to their respective hardware and software, allowing an integrated view and usage. They can be seen as successors of the monolithic management applications of the earlier supercomputers. But they are not flexible enough to serve systems of different sizes or types.
  - The software cannot be used on systems that are different from the original supercomputer (different SAN, platform, size, or topology).
  - The systems are «one-shot» systems: They are installed once and never altered. Upgrades are performed on a per-component level; system structure or size changes are difficult. If another system is ordered, a copy with minor differences from the original is built.
  - It is very unlikely that the software will be published officially, because the integrating company will not want to uncover secrets or the mission-critical ASCI security mechanism.
  - The management software is not designed to be modular and extendable.
- Resource management applications are already being run on superclusters today. It seems that the extension toward comprehensive integrated management is a short path. Unfortunately this shortcut turns out to be a very long trip.
  - The software already uses manager/agent architecture. Adding further features into those existing software components seems easy.
  - Additional components for scalability (proxies) and changes to the architecture to add reliability and availability (clusters, master/backup) are necessary.
  - The extension of commercially available software itself is unlikely, since the development efforts are high and the market is (currently) small.
  - It is more likely that new or available open-source products will be extended by the research institutes of universities, suppliers and integrators.
  - Extending an existing product with features and functionality that it was not designed for is very difficult – it is easier to start from scratch (using the acquired experience) instead of inflating a framework and struggling with flaws, protocols and interfaces.
- SNMP is already used for managing network components. Supercomputers and all their switches, nodes, applications and more can also be seen as network components. Turning SNMP into secure, reliable and scalable supercluster management software will require a huge effort and will lead to a completely different product.
  - The connection between agents and manager is insecure and unreliable.
  - There is no security (authentication, integrity, etc.).
  - The architecture must be extended with proxies, reliable managers (e.g. clustering), persistent MIB storage, application management and more. Too much for a simple extension – it is easier to create something from scratch.

The current management tools, fixes, workarounds and extensions are too weak or unusable for the large commodity supercomputing systems of the future. Only global research and development effort with the goal of standardized interfaces for management

modules and a globally used framework can lead to integrated, comprehensive management software for superclusters.

Since the effort will lead to a global standard, the best idea is to keep the development in the open-source community. Because all manufacturers of supercomputing-enabled components will profit from this approach, all should contribute resources towards the goal. If a manufacturer wants to invest in its own management software, he will quickly recognize that his effort will be lost because the world outside will integrate, making his own product obsolete and unusable in the standardized environment.

5 COSMOS – The Implementation

It is common sense to take a method and try it. If it fails, admit it frankly and try another. But above all, try something.

Franklin D. Roosevelt

While the previous chapters presented problems, solutions and existing software for supercomputer management, this chapter is dedicated to the author's management software «COSMOS» (short for «**Co**mmodity **S**upercomputing **M**anagement **O**peration **S**oftware») and «Swiss-Tx», the research project where COSMOS was embedded.

5.1 The Swiss-Tx Supercomputing Project

The «Swiss-Tx» supercomputing project [GG97] [GD+98] [KG99] [BLF+99] was launched in 1997. Its goal was to strengthen and support computing experience and research of leading-edge technology for commodity parts-based supercomputing.

The following institutions participated in the project:

- The **Swiss Federal Institute of Technology in Lausanne (EPFL)** maintained the superclusters of this project. Their rich know-how and experience with supercomputers and their software allowed the right goals to be focused on.
- The **Swiss Federal Institute of Technology in Zurich (ETHZ)** developed the commodity supercomputing technologies for the project: The system area network, the communication libraries and the management software.
- The **Swiss Commission for Technology and Innovation (CTI)** funded the project and supported the project management.
- The **Swiss Center for Scientific Computing in Manno (SCSC)** supported the project by developing software for debugging and tracing data in superclusters.
- **Supercomputing Systems Ltd. (SCS)** built the superclusters and co-operated with ETHZ in developing the technologies for commodity supercomputing. The development teams consisted of members of both institutions, including graduate students of ETHZ.
- **COMPAQ Computer Corporation (CPQ)** provided the computers for the supercluster and supported the implementation of the supercluster technologies as well as the integration with the necessary software and the installation and configuration of the system.

The goal of the project was to develop the technologies and skills to build and install supercomputers with a peak performance of one TFLOPS. This supercomputer was to be built using standard workstations and as many commodity components as possible.

5.1.1 The five Swiss-Tx Superclusters

Five milestones were defined for the Swiss-Tx supercomputing project, represented by five superclusters. Each supercluster used the best technology available at the time and every design was analyzed for further steps. The five milestone superclusters were as follows:

Machine Name	Number of Processors	Memory [GBytes]	Storage [GBytes]	SAN Technology	Installation Date
Swiss-T0	8	2	64	EasyNet	12/1997
Swiss-T0 (Dual)	16	8	170	EasyNet	10/1998
Baby-T1	12	6	130	T-Net	08/1999
Swiss-T1	64	32	900	T-Net	01/2000
Swiss-T2	504	252	5000	Open	Open

Table 5.1: The five Swiss-Tx milestone superclusters

The last supercluster «Swiss-T2» was never built because the EPFL supercomputing center CAPA chose to buy an ordinary supercomputer. Creating a Tera-scale supercluster based on self-made products and technologies would have boosted the supercomputing research workplace in Switzerland – the spineless retreat, however, stopped these efforts.

5.1.1.1 Swiss-T0 Supercluster

The first supercluster, the Swiss-T0, fit with all its components into one single rack: Eight Alpha-based single-CPU workstations running COMPAQ Tru64 UNIX, interconnected with the bus-based EasyNet SAN technology, one Ethernet switch, one terminal server and a DLT library.

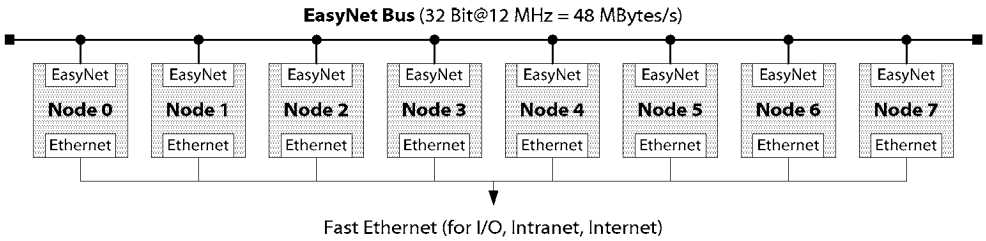


Figure 5.1: Structure of the Swiss-T0

This system was a copy of a prototype at ETHZ with eight Intel-based PCs running Linux. Both systems suffered from being used with the EasyNet bus, since the ribbon cable broke easily and the bus-based communication had limited performance that did not scale.

5.1.1.2 Swiss-T0 (Dual) Supercluster

The second supercluster, the Swiss-T0 (Dual), needed two cabinets containing eight Alpha-based dual-CPU SMP servers running Microsoft Windows NT. The operating system was later replaced with Tru64 UNIX because the users did not want to port their applications to Windows NT, although the performance was the same. The bus-based EasyNet SAN technology was used to interconnect the nodes, but the limitations in cable length and connection-to-connection distance required a special node placement within the racks.

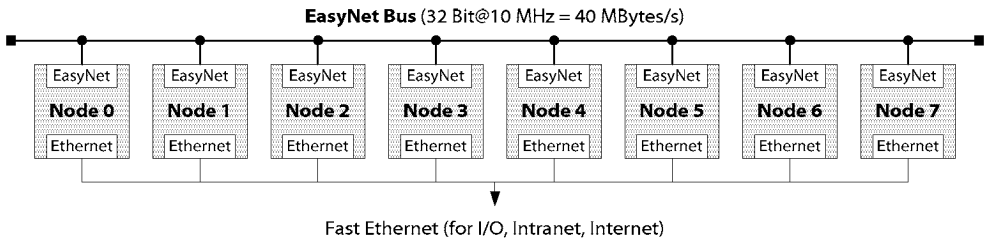


Figure 5.2: Structure of the Swiss-T0 (Dual)

Because the nodes were bigger than those of the Swiss-T0, the ribbon cable connecting the nodes was longer, leading to a lower maximal network bandwidth of 40 MBytes/s. The higher node performance could not compensate for the slower network performance, and lead to the Swiss-T0 (Dual) having the same performance for applications using all processors as the Swiss-T0.

5.1.1.3 Baby-T1 Supercluster

The Baby-T1 was the third supercluster. The switch-based T-Net SAN technology had been finished and the Baby-T1 was its prototype system. Eight Alpha-based dual-CPU SMP servers running Tru64 UNIX delivered the expected performance.

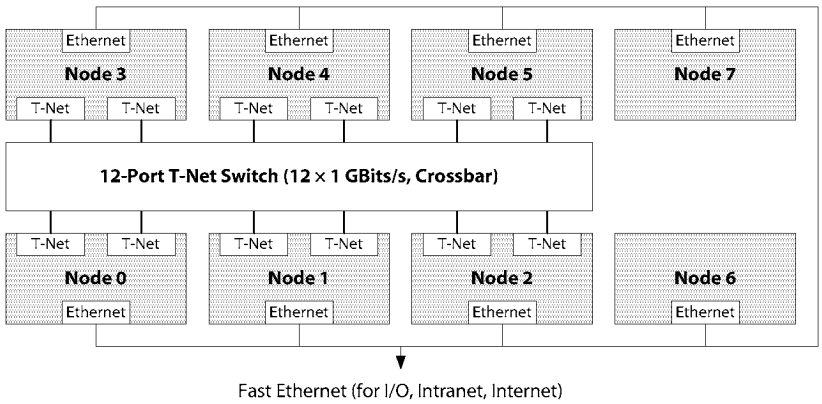


Figure 5.3: Structure of the Baby-T1

Because the T-Net technology was limited to one 12-port SAN switch and one CPU per SAN NIC, only six of the eight computers were connected to the switch. The largest job for the SAN-based MPI was limited to 12 CPUs, but all 16 CPUs could be used for other MPI implementations such as MPICH that used Ethernet for communication.

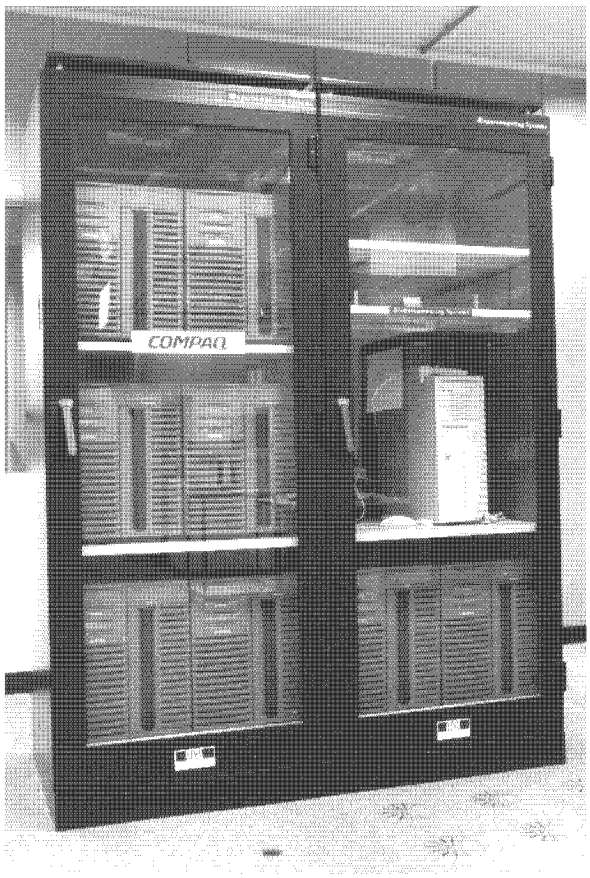


Figure 5.4: The Baby-T1 supercluster

5.1.1.4 Swiss-T1 Supercluster

The Swiss-T1 was the fourth supercluster and was in use until late 2003. The switch-based SAN technology T-Net was used here after it had proved its functionality in the Baby-T1. The Swiss-T1 was the first supercluster in which all components were designed for 19" racks.

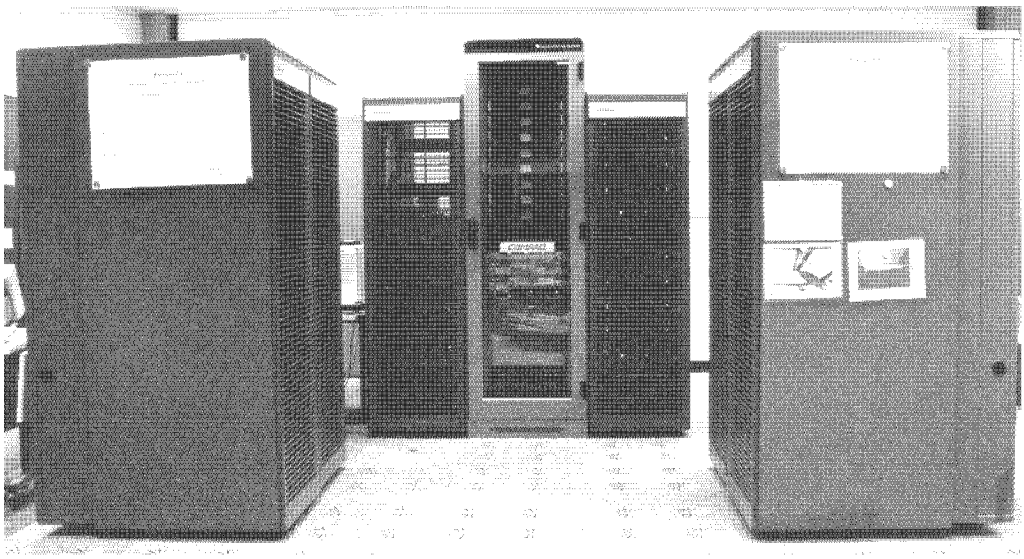


Figure 5.5: The Swiss-T1 supercluster

The supercluster consisted of 32 COMPAQ AlphaServer DS20 compute nodes with 2 Alpha 21264 microprocessors at 500 MHz and 1 GBytes main memory each. Two additional DS20 computers built the management layer (called «front-end nodes»), used as file server, management center and additional tasks. One single DS20 computer was used as a stand-alone test machine. All computers used the COMPAQ Tru64 UNIX operating system. The 32 nodes were mounted in 4 racks, the storage subsystem was placed in a separate rack, the two front-end computers and the independent 33rd node were found in another rack.

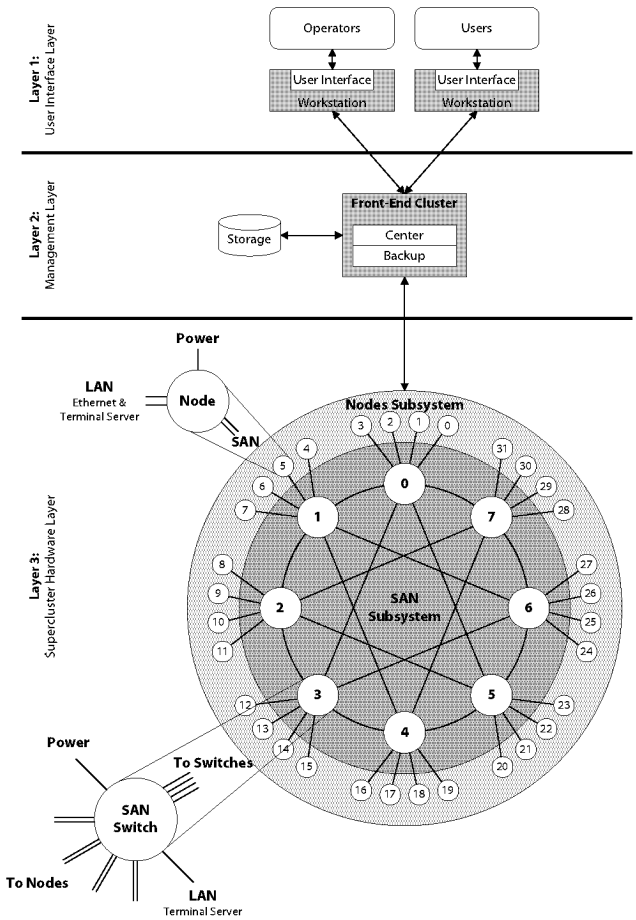


Figure 5.6: Structure of the Swiss-T1

The Swiss-T1 SAN topology was an invention of the EPFL: A one-dimensional 2-Ring of the size 8. The eight SAN switches were interconnected in two rings each having four neighbors, and building a cube with interconnected diagonals. Each SAN switch had four nodes connected to it and every node had two SAN NICs inserted into it. All nodes were connected to the Ethernet switch using a Fast Ethernet link. The two front-end nodes built a reliable cluster for the file server service. Both were connected to the compute nodes using a Gigabit Ethernet link.

One rack contained the 9 SAN switches (8 for the Swiss-T1, one separate one for the test system), the Gigabit Ethernet switch, the two power switches and the two terminal servers. This rack was used for the interconnections and the management of the supercluster.

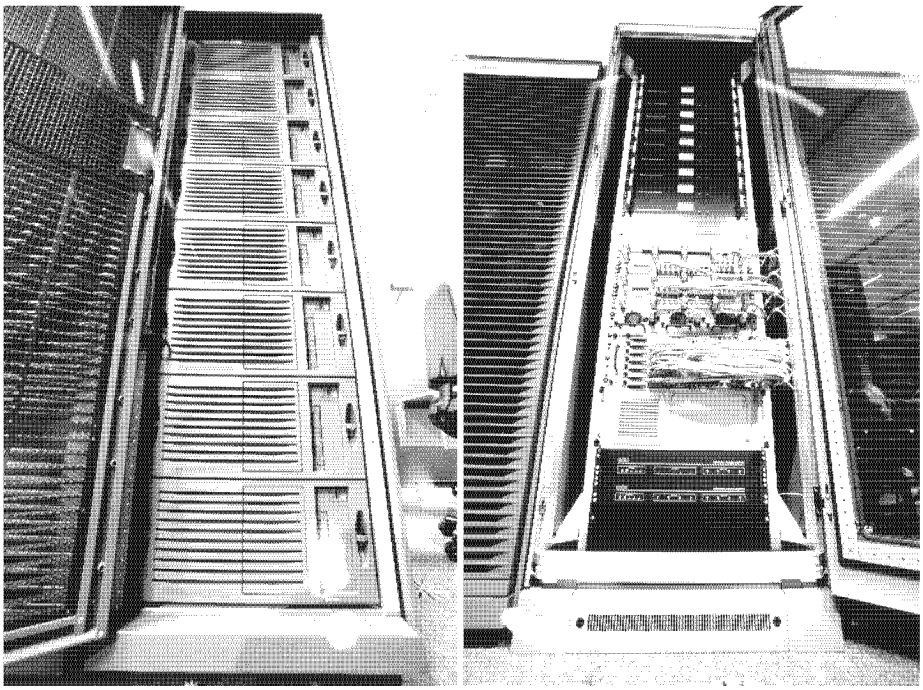


Figure 5.7: One of the four node racks (left), each containing 8 AlphaServer DS20E nodes. The network rack (right) containing 9 T-Net switches, two terminal servers, the Gigabit Ethernet switch, a management computer and two remote power switches (from top)

5.1.1.5 Swiss-T2 Supercluster

The Swiss-T2 supercluster would have consisted of 21 T-Net switches, interconnected in a 3-ring topology, with 6 quad-CPU computers per switch, making a total of 504 processors.

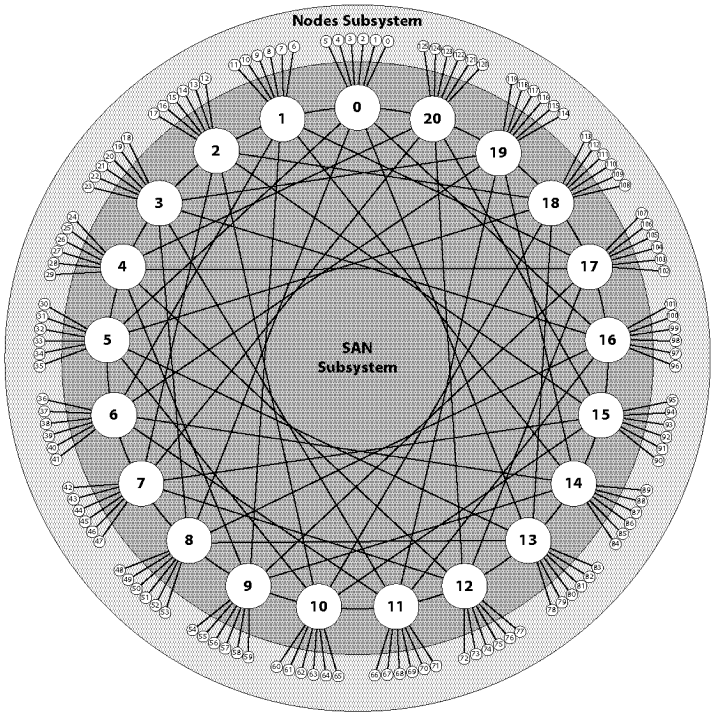


Figure 5.8: The structure of the Swiss-T2 with 21 switches, 126 nodes and 504 processors

5.1.2 The Swiss-Tx Networking Technologies

Two SAN products were developed for the Swiss-Tx project: The bus-based «EasyNet» and the switch-based «T-Net». Both were developed in co-operation between SCS and ETHZ. The details of the research and design have been summarized in the dissertation of Martin Lienhard [Lie00].

5.1.2.1 EasyNet

EasyNet is a low-cost, bus-based network. It consists of a PCI-based SAN NIC and a ribbon cable interconnecting the NICs. The NIC consists of a PCI bridge and a FPGA chip that contains the communication controller. The ribbon cable has a width of about 10 cm and contains 60 lines for a bus of 32 Bits width with error detection, control, synchronization and key management.

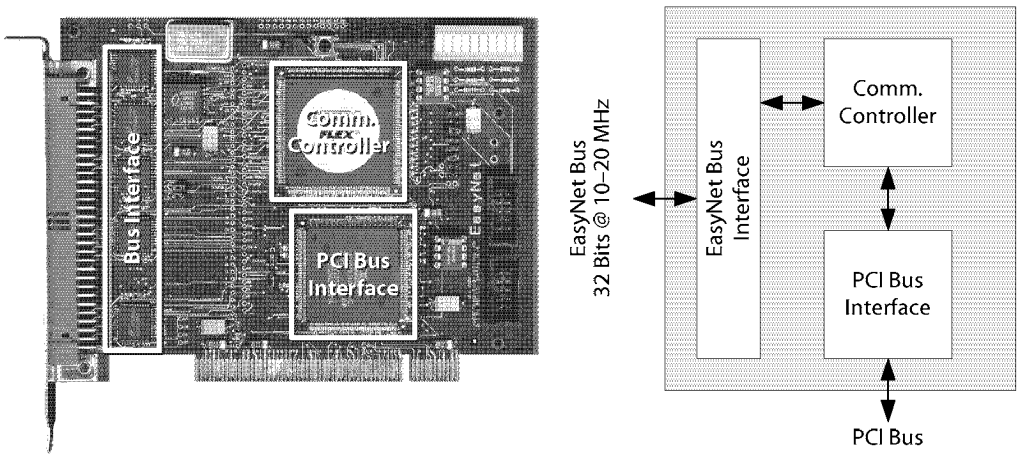


Figure 5.9: Image and block diagram of the EasyNet SAN NIC

The SAN NIC that is located in the middle of the bus provides the bus clock, and the two outermost SAN NICs terminate the bus signals. The design allows communication mechanisms that basically use broadcasts, such as the extended reflective memory model [JM99] [PTM96] [Mea96] that can be seen as globally distributed shared memory with global store and local read access.

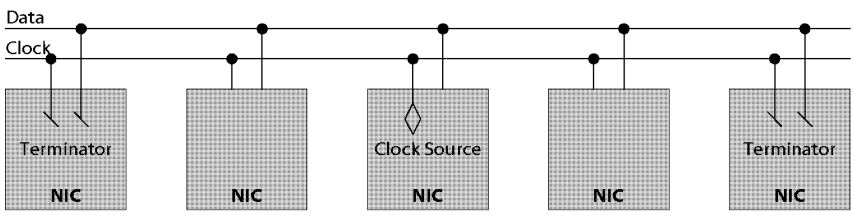


Figure 5.10: EasyNet network structure

The advantages of the design are the moderate bandwidth (between 40 and 80 MBytes/s) and low node-to-node latency (about 5 µs), together with the possibility of performing broadcasts, as well as the copying-free transfer mechanism between sending and receiving applications («zero-copy»). The bottleneck of the design is the bus, since the available bandwidth is divided by the communicating nodes and the bus clock is limited by the cable length.

The most annoying problem of this product was the ribbon cable: It was hard to manufacture, inflexible and broke easily. The connector fell out readily and the cable aged quickly. Debugging the software and driver was very difficult since it was hard to say if a problem was caused by a software bug or the unreliable hardware.

5.1.2.2 T-Net

T-Net was designed as a high-end, switch-based network. It consists of a PCI-based SAN NIC and a 12-port SAN switch. The NIC consists of a PCI bridge, two FPGA chips for the communication and link controller, and a link interface for the bi-directional Gigabit link. The link can be either a FC-compatible copper cable of up to 20 m, or an optical link of up to 500 m that requires MIA (media interface adapter) devices.

The SAN NIC contains a page table (for virtual-to-physical address translation in non-contiguous memory blocks) and an ID validation table that contains a list of process IDs that are allowed to send or receive messages.

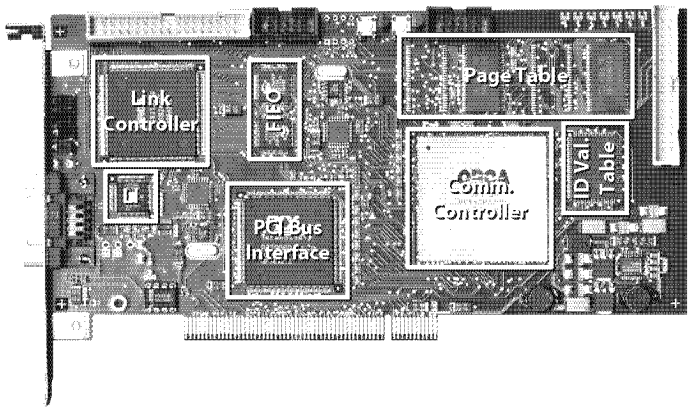


Figure 5.11: Image of the T-Net SAN NIC

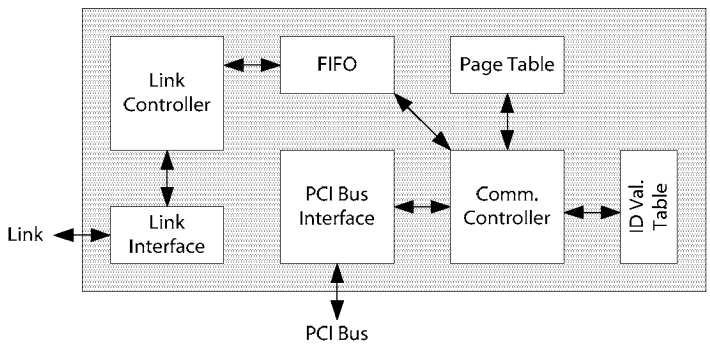


Figure 5.12: Block diagram of the T-Net SAN NIC

The SAN switch is a 12-port non-blocking crossbar switch. The on-board routing table allows destination-based routing with extensions for broadcasts and multicasts. The simple deadlocks that can occur in mesh-like topologies are prevented by hardware. The on-board controller manages the whole SAN switch. It programs the FPGA chips, maintains the routing table, updates the firmware, controls the switch display and provides the monitoring information.

All T-Net features were designed with MPI as the communication library in mind. This optimization allows high bandwidth (about 80 MBytes of sustained performance) and low latency (less than 10 µs) in combination with low cost.

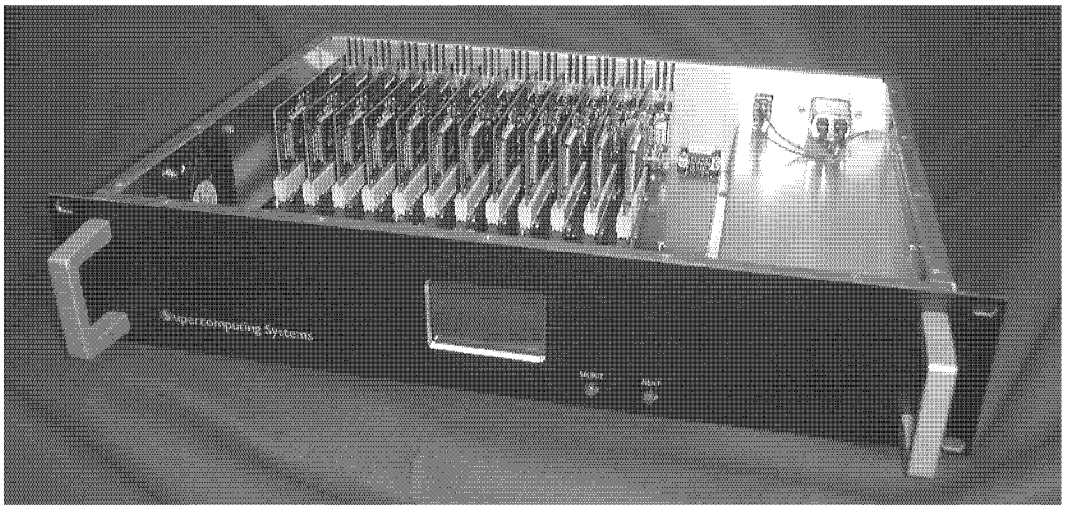


Figure 5.13: Image of the T-Net SAN Switch

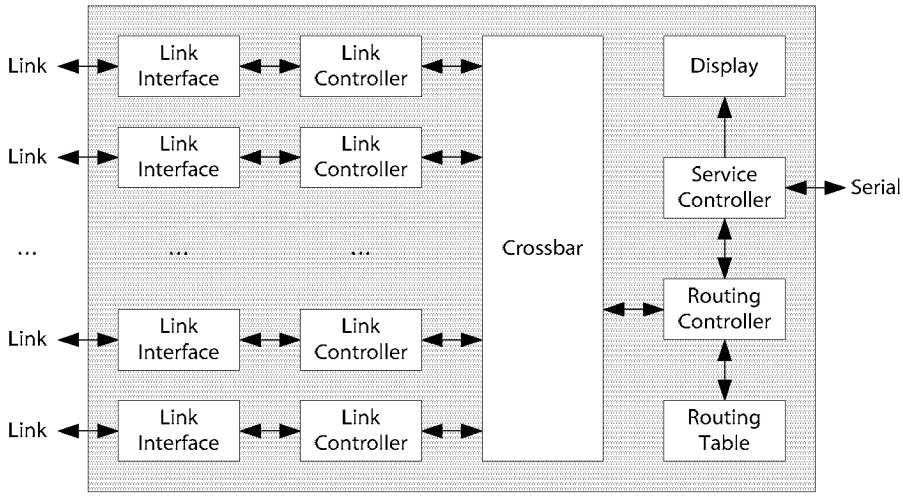


Figure 5.14: Structure of the T-Net SAN Switch

The SAN NICs and switches build a network topology and T-Net supports many classical regular topologies (multidimensional mesh and torus, hypercube) as well as modern topologies (fat tree, multidimensional k-torus).

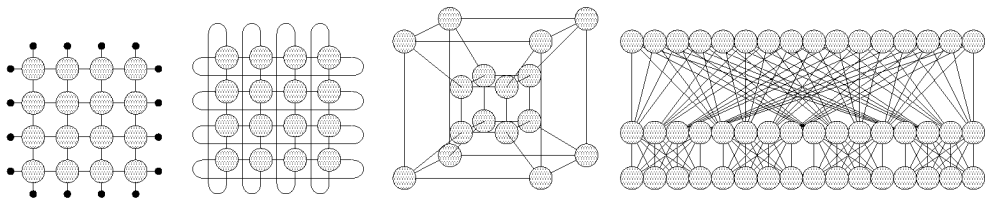


Figure 5.15: Topologies for switch-based networks (from left): 2D mesh, 2D torus, 4D hypercube, fat tree

5.1.3 The Swiss-Tx Communication Libraries

The programming model most often used for parallel programming is MPI (Message Passing Interface) [MPI95] [MPI97]. For the Swiss-Tx project, MPI was implemented as a layer on top of the core communication library called FCI (Fast Communication Interface). FCI communicates with the NIC device driver using a standardized custom interface (DDI – Device Driver Interface) and with the NIC hardware directly through use of another standardized custom interface (NHI – Network Hardware Interface).

Both standardized but custom interfaces DDI and NHI allow the reduction and simplification of the software modification if the underlying platform (EasyNet, T-Net, future SAN NICs) changes. The driver is only used for functionality where kernel support is necessary (interrupt and memory handling, inter-application security). For most message-passing operations, the SAN NIC is accessed directly without application and operating system invocation.

Data is moved between application memory regions on the source and destination nodes, allowing zero-copy operations and therefore low latencies and high bandwidth. With zero-copy, the message-passing send function writes the message directly into the SAN NIC and the destination moves the data directly into the application memory. Zero-copy allows the bus load to be reduced, and the operating system is not invoked which saves context switch time (some µs).

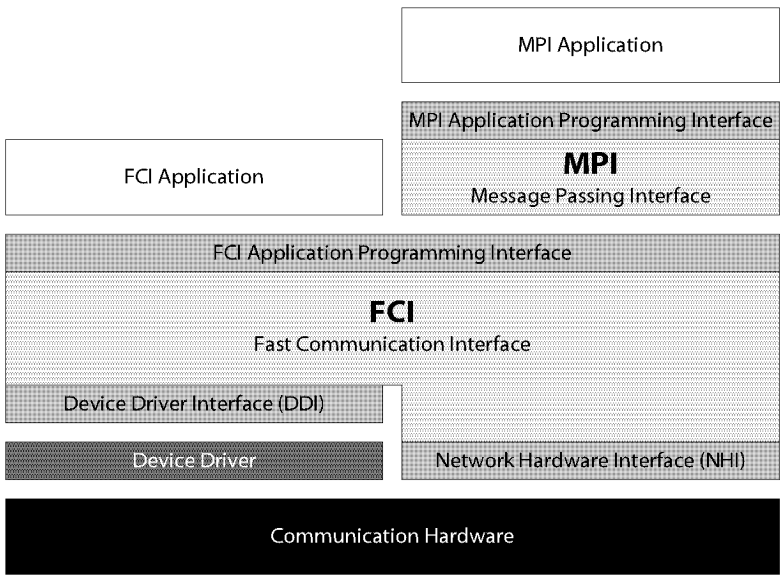


Figure 5.16: FCI and its integration in the environment

FCI provides the following functionality:

- Management of process groups.
- Blocking and non-blocking routines for message passing send, receive and probes.
- Collective operations such as barriers and broadcasts.
- Procedures for the Remote Store programming model.
- Routines for mutual exclusion (key management).
- Environmental management calls (start-up, shut-down, environmental queries).

The communication library system FCI/MPI has been successfully implemented and ported to the platforms COMPAQ Tru64 UNIX, Linux and Microsoft Windows NT 4.0.

The details of the research, design and implementation of the communication library technologies for the Swiss-Tx project have been summarized in the dissertation of Stephan Brauss [Bra00].

5.2 COSMOS Concept

From an ideal point of view – especially after reading chapters 2 and 3 of this dissertation – the management software for the Swiss-T1 (and its successors) should have been fully integrated. The users and administrators use one user interface and access the MIB of one management application, which manages the entire supercluster.

Since the presented research was performed after the creation of COSMOS, the advantages of the integrated approach over the later selected glueware architecture was not known at the time when COSMOS was implemented – the ideal (and more beautiful) design was only discovered later.

According to the Swiss-Tx project plan, COSMOS was considered to be a simple application, managing T-Net hardware and applications using the MPI/FCI parallel environment only. This consideration was based on the experiences which had been made with NOW systems (networks of workstations), where the subsystems are managed independently, expecting that the SAN and applications would be just more independent subsystems.

Based on the needs of administrators and users, as well as the needs of T-Net and FCI/MPI, COSMOS concentrated on software that allowed comprehensive, but not integrated, supercluster management. Due to the small system size of the Swiss-T1, the effects of this non-integrated approach did not pose too many measurable problems.

5.2.1 Administrator and User Needs

The needs of the administrators and users are basically explained in section 2.1. However, the needs of the Swiss-T1 supercluster were not as sophisticated when concerning functionality, integration and automatism.

The administrators' needs can be summarized as follows:

- The 32 nodes are the body of the Swiss-T1 and must be fully managed.
  - They want to be able to control the power state of the nodes remotely. This takes place through the console, which is connected to the LAN using a console server.
  - They want to be able to update the image that is stored on the local hard drive, containing OS, software, drivers, firmware (e.g. for the SAN NIC), and global or node-individual configuration data. This update requires special configuration and control software.

- The administrators want to have direct access to each individual node, which is quite simple because they can use the standard UNIX tools and mechanisms (telnet, remote shells).
- They want to be able to monitor the vital values: Status, active processes, and resource utilization.
- They want to be able to detect faults and problems quickly, because utilization and efficiency drops when hanging nodes or blocked resources prevent applications from being executed.

❖ The 8 SAN switches and 64 SAN NICs interconnect the nodes for fast communication.
- They want to be able to control the power state of the SAN switches. Since T-Net has no remote power capabilities, LAN-based power switches must be used.
- They want to be able to download and update the firmware and configuration data that is stored in the memory of the switches and NICs.
- They want to be able to monitor the vital values: Temperature, memory utilization, uptime, and for each port the status, data and error rate.
- They want to be able to detect faults, since the applications are forced to use massively slower MPI implementations (such as MPICH) instead of MPI/FCI if the SAN is unavailable.

❖ The LAN consists of one Gigabit Ethernet Switch, interconnecting all nodes, as well as the console servers that connect the nodes' console ports to the LAN.
- They want to be able to control the power state, the configuration, as well as the many features.
- They want to be able to detect faults, since unavailable LAN connections disable storage and management access to all supercluster components.

❖ The storage subsystem consists of a RAID array connected to the two front-end computers.
- They want to be able to control the storage service, as well as its configuration.
- They want to be able to detect faults as well as exceeded limits, such as user quotas, soft errors, consumed storage space etc.

❖ The administrators want to be able to manage the computation resources: Queues and CPU hours.
- They want to be able to control the queues and node resources (CPU slots) as well as be able to update the configurations relating to open times, user access, priorities, etc.
- They want to be able to account for the resource usage per user, department and project.
- They want to be able to detect faults: Unavailable CPU slots, blocked queues and applications, etc.

❖ The administrators want to be able to manage the resource consumers, the applications:
- They want to be able to control the applications (mainly aborting them).
- They want to be able to monitor the applications and their processes: Status, resource consumption, node location, time stamps, etc.
- They want to be able to detect application abortions and the reasons for such abortions.

❖ The administrators want to be able to manage the database for users, departments and projects, mainly in regard to the permissions for resource access and consumption, as well as user and group rights on the node computers.

The requirements of the users and developers concerning the management software toolkits can be summarized as follows:

- The users want to have fair and deterministic access to the supercomputer resources:
  - They want to know when their applications will start and finish.
  - They do not want higher-priority users always to expel their applications.
- The users want to have enough information and permissions for their job:
  - They want to know on which nodes the processes of their applications are to be executed, so they can connect debuggers, tracers, and other development tools to those nodes.
  - They want to have permissions to connect to the nodes, using those tools, and to send signals to their application processes.
- The users want their applications to receive all available computing time, and no resources to be lost on management or other non-productive tasks.

All these needs are respected in the management requirements and constraints presented next.

5.2.2 Requirements and Constraints

The administrators of the Swiss-T1 were already used to various management applications. They expected that they would be able to use the same software as with their NOW systems for the various tasks and components. This included the following software:

- The Gigabit Ethernet switch was managed by using either a web browser for http-based management of the whole configuration and functionality, or one of the many available free or commercial SNMP manager tools.
- The node computers were managed individually using manufacturer-supplied management software (SYSMAN). The manufacturer also created special software and script-based tools that allowed more global tasks (such as global boot, image update, or shutdown).
- The resources were managed using GRD/Codine from GRIDWARE (now a subsidiary of Sun Microsystems), which included the management of queues, users, and applications.
- The storage subsystem was managed using manufacturer-supplied software.

The previously presented requirements can be put into a matrix, with the subsystems as presented in 1.4.3 in one dimension, and the functionality groups as presented in 2.3.1 in the other dimension. The content of each cell is the management software that takes care of the functionality of the respective supercluster subsystem:

Subsystem	Functionality						
	Control	Config	Monitor	Fault	Trap	Account	Sched.
Resource Management	GRD	GRD	GRD	GRD	GRD	GRD	GRD
Application Management	GRD	GRD	GRD	GRD	GRD	GRD	GRD
User Management	GRD	GRD	GRD	GRD	GRD	GRD	GRD
Node Management	SYSMAN	SYSMAN	GRD	SYSMAN	GRD		
SAN Management							
LAN Management	SNMP	SNMP	SNMP	SNMP	SNMP		
Storage Management	COMPAQ	COMPAQ	COMPAQ	COMPAQ	COMPAQ		

Table 5.2: Distribution of the functionality to the management applications (initial)

Accounting and scheduling is only necessary for the resource usage by users and applications, and not for the other subsystems – this is the reason why these cells have been left empty. The empty SAN management line should have been filled by COSMOS, since it was considered to be a simple application for managing the SAN.

This matrix suggests that the Swiss-T1 is managed in the same way as the classical NOW systems, where the resource management software (in this case GRD/Codine) takes care of the software subsystems, and manufacturer-supplied management software packages take care of the respective hardware subsystems. The administrators are accustomed to this setup and know how to use the software to manage the supercluster.

During the implementation of MPI/FCI for EasyNet, the scope of COSMOS was enlarged, since shortcomings in GRD/Codine and the selected design of MPI/FCI meant that some of the functionality had to be covered by COSMOS. At the beginning of the design and implementation of COSMOS, the management software matrix looked as follows:

Subsystem	Functionality						
	Control	Config	Monitor	Fault	Trap	Account	Sched.
Resource Management	GRD	GRD	GRD	GRD	GRD	GRD	GRD
Application Management	COS/GRD	COS/GRD	COS/GRD	COS/GRD	COS/GRD	GRD	GRD
User Management	GRD	GRD	COS/GRD	GRD	GRD	GRD	GRD
Node Management	SYSMAN	SYSMAN	GRD	COS/SYS	GRD		
SAN Management	COSMOS	COSMOS	COSMOS		COSMOS		
LAN Management	SNMP	SNMP	SNMP	SNMP	SNMP		
Storage Management	COMPAQ	COMPAQ	COMPAQ	COMPAQ	COMPAQ		

Table 5.3: Distribution of the functionality to the management applications (definitive)

This matrix leads to the glueware architecture as presented in chapter 3, where the users and administrators integrate the functionality provided by the various management applications, and some management applications cover more than one subsystem. Since the users prefer using management software and user interfaces that they have known for years, and can hardly accept any new management software, this decision was very reasonable.

The functional requirements for COSMOS can be summarized as follows:

- It manages the SAN switches and SAN NICs.
- It manages the applications and their processes.
- It monitors the nodes, and detects and handles crashes.
- It monitors the users as owners of the applications.
- It provides information to users using graphical and text-based user interfaces.
- It does not consume too many computation resources on the nodes.
- It interfaces or overlaps with other management tools where necessary, but does not interfere with them (e.g. does not perform tasks that are owned by other tools such as shutting down nodes or closing queues).

Besides these functional requirements, there were also the following project-based requirements:

- Support the Swiss-Tx supercluster platform:
  - Nodes with COMPAQ Alpha or Intel x86 processors.
  - COMPAQ Tru64 UNIX or Linux operating system.
  - T-Net SAN products (NIC and switch).
  - FCI/MPI communication library.
- Support the potential Swiss-Tx supercluster architectures:
  - 32 to 126 node computers, 64 to 504 processors.
  - 8 to 21 SAN switches, 64 to 504 SAN NICs.
  - One-dimensional 2-ring or 3-ring SAN topology.
- Time constraints:
  - Working prototype for installation day (6 months after development start).
  - Reliable production version for inauguration day (10 months after development start).
  - Whole development (including fancy GUI) to be finished within 15 months.

Besides these requirements, which were defined by the Swiss-Tx project and their managers, the development of COSMOS also faced additional constraints created by the project setup:

- Provide text-based interface from the beginning, graphical user interface later.
- Use open-source programming libraries that are available on all platforms.
- Programming in C (except for the GUI that required C++).
- Since GRD/Codine uses a center application on the front-end computers, and connected demons (background services [Ste98]) on the nodes, COSMOS should use a similar design.
- Development of COSMOS closely coupled to FCI and T-Net development, using same source tree and tools (compilers, libraries, source management).

These requirements and constraints – following on from the needs of users and administrators – were the only basis for the design and implementation of COSMOS. Everything else was left to the author's discretion.

5.2.3 Specifications

The previously presented requirements and constraints were consequences of the needs of the administrators and users. In order to create the software, these requirements and constraints had to be converted into detailed specifications, which were defined during intensive discussions with the project team members at EPFL, ETHZ and SCS.

Part of the specification process was the separation of management tasks into those that were to be solved using software, and those that were to be solved using shell scripts. Some functionality was easier and faster to design and implement with tools that are available in UNIX and Linux shell environments (using remote shell or expect scripts, Crontab etc.), and some functionality required specially compiled software in higher-level programming languages.

The requirements led to the following detailed specifications for COSMOS:

- COSMOS is to manage the SAN switches and SAN NICs:
  - It is to control the power state of the switches, using remote power switches with a telnet-based user interface. The power state is to be controlled with expect scripts.
  - It is to download and update the firmware and configuration of switches and NICs. The switch firmware is to be stored in persistent memory and updated using shell scripts. The firmware of the NIC is to be stored in volatile memory and must be downloaded during the node boot process. A script is to download the current firmware into the NIC, together with the basic NIC configuration.
  - It is to monitor the vital values of the switches. The switch monitoring is to be performed using programmed software, but the NIC is not to be monitored because it does not include any monitoring functionality (insufficient on-board hardware resources).
  - It is to detect switch errors using programmed software. Since the switches present «temporary soft errors» (bit errors in messages) in the same way as they present «permanent hard errors» (open link), handling those errors automatically (e.g. aborting applications on nodes with erroneous T-Net links) would have been very difficult. COSMOS therefore was only required to display these errors. The NIC was not covered because it was impossible to monitor.
- COSMOS is to manage the applications and their processes using MPI/FCI:
  - It is to distribute the processes to the reserved nodes and process slots, using remote shell scripts that are called by GRD/Codine.
  - It is to control the processes and applications using programmed software, because the processes must receive T-Net access information, detect its PID that is required by the management software for reliable process control.
  - It is to allow users and administrators to reliably abort applications with user interfaces. The user interface is to communicate with the programmed software which is then to send abortion signals to the processes of an application.
  - It is to provide the system and application configuration to all processes. This information is to be transmitted by the programmed software during startup.
  - It is to monitor the vital values of processes and applications. This information is to then be stored by the programmed software in its MIB.
  - It is to detect abnormal behavior in the processes and applications. The programmed software is to detect time-outs of the processes (during startup, finalization, abortion and other management-relevant actions) and is to handle them in accordance with policies.
  - It is to detect and handle faults of processes and applications. The programmed software is to detect process crashes (and their causes) and initiate the application abortion.
- COSMOS is to monitor the nodes, and detect and handle crashes. The programmed software is to detect whether nodes are ready to accept new processes, or whether they have crashed and applications with processes on that node need to be aborted.
- COSMOS is to monitor the users as owners of the applications. The programmed software is to detect the users (and further information) with the PID detection process during startup.

- COSMOS is to provide information to the users and administrators:
  - It is to show vital values of each node, SAN switch, SAN link, application and process. The programmed software is to collect this data and provide the information to user interfaces.
- COSMOS is not to consume computation resources on the nodes. Inspired by the model of GRD/Codine with one center and agents on the nodes, the programmed software on the nodes is to be passive and is to react only when application processes or the programmed software running on the center sends messages.
- COSMOS is to provide graphical and text-based user interfaces. For efficiency reasons, both user interfaces must be programmed software (instead of script-based tools, e.g. Tcl/Tk) which connects to the programmed monitoring data collector and manager on the center.
- COSMOS is to interface or overlap with other management tools where necessary, without interfering with their tasks or exchanging data that might be inconsistent with their MIB:
  - GRD/Codine is to use the remote shell script provided by COSMOS for distributing the application processes on the nodes.
  - GRD/Codine is to use its own resource usage map to decide which slots to use, but COSMOS is to check these slots as well because they are bundled with a T-Net NIC channel. If COSMOS detects an inconsistency (e.g. already occupied node slot), it is to abort this application before being started.
  - COSMOS is to detect crashed or unavailable nodes, but it is not to boot or shut them down.
  - GRD/Codine and COSMOS are to detect application abortions or can trigger them. Since the administrators detected that GRD/Codine tends to lose processes, COSMOS must include a mechanism to reliably abort application processes that may occupy slots.
- COSMOS must include mechanisms for its own management:
  - Shell scripts are to start, restart or stop COSMOS on the whole supercluster.
  - The programmed software on the center is to detect unmanageable nodes.
  - The programmed software on center and nodes is to deduct its own configuration from global configuration files.

To be honest, some of the points in this specification were missed from the first version, since not all points had the same importance, or were not required in the first version. Indeed, COSMOS evolved over three generations:

- The first version was required as a prototype system at SCS and for the installation of the Swiss-T1 at EPFL in January 2000. This version required only basic application management functionality (centralized startup and finalization support).
- The second version was available for the inauguration of the Swiss-T1 in August 2000. This version included the complete specification list as presented above, except for the SAN monitoring and the graphical user interface.
- The last version became available in March 2001 and included the complete specification list, including SAN monitoring and the graphical user interface.

The initial specification list was created in December 1999 (for the first software version), with slight modifications and extensions being made between January 2000 and May 2000 (for the inauguration version). After inauguration, the author created the

specification for a second implementation of COSMOS that would have been used on the Swiss-T2. After EPFL decided not to order this system, the author concentrated on the third software version, adding the SAN management features and the graphical user interface to the above specification.

During implementation, some of the points which were to be solved using scripts were experimentally implemented in software (such as distribution of processes on the nodes). Although the results of those experiments were promising, the initial scripting solutions were used until the end. The main reason for this decision was that adding those features to the software would not bring any benefits to the users.

5.3 COSMOS Design

The design of COSMOS follows the previously presented concept:

- COSMOS uses the same «mental model» as GRD/Codine or SYSMAN, with a center application, agents on the nodes, and user interfaces:
  - The users are used to this architecture and they accept it.
  - Documentation and explanations are unnecessary because COSMOS works similarly to the tools they are already used to.
  - The design decisions taken by the other tools' developers were based on reasons, and there is no need to create something completely different if these designs work well.
- The center is the central authority in the supercluster:
  - It manages the applications and the process slots on the nodes.
  - It detects node faults (such as disconnected node agents).
  - It holds the central MIB in its memory.
  - It is the central connection point for all user interfaces.
- The node agents support the center as reliable partners in process management:
  - They monitor the vital values of the node and FCI/MPI processes.
  - They control all FCI/MPI processes running on the local node.
  - They provide configuration data for FCI/MPI processes.
  - They detect FCI/MPI process faults and abnormal behavior.
- The SAN agent monitors the SAN switches and collects their vital values.
- The user interfaces allow the users to browse the center's MIB and aborting applications:
  - The CLI is a text-based application that can be used from any shell.
  - The GUI is a graphical application that requires a UNIX workspace.
- The FCI/MPI applications are integrated into management using a shared library.

The following illustration presents this six-component design of COSMOS:

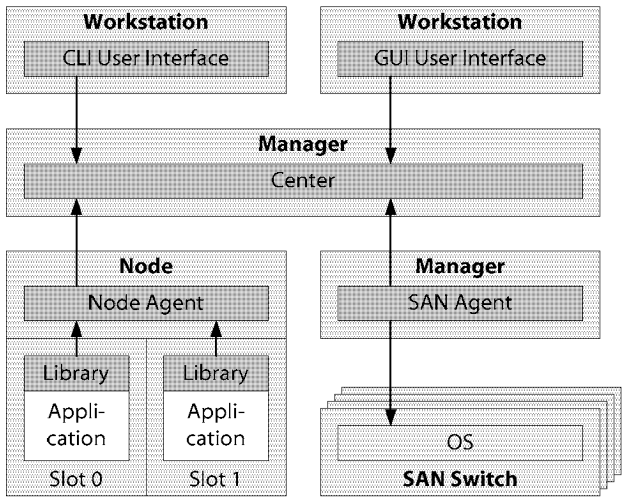


Figure 5.17: COSMOS component hierarchy (with COSMOS software in grey)

This illustration also presents one design decision, the «bottom up» connection strategy: The library connects to the node agent, and the agents connect to the center, as well as the user interfaces. Only the SAN switches are connected by the agents «top down», because the OS is unable to create connections autonomously. The reason for this strategy – which is the opposite of the connection strategy of all other known management tools – is simplicity and security:

- The center has more important tasks to do than connecting to all agents, which consumes a lot of time and resources. Especially in fault cases, where nodes are unavailable, the center is forced to lose time that should be used for handling these faults instead of trying to connect them. Additionally, the servants should go to the king, and not vice versa.
- Malicious software can easily act as center and harm the operation if the agents are required to accept center commands originating from incoming connections. Since the agent accepts the center as the single management authority and knows how to contact it to receive commands, malicious software cannot gain control over nodes or SAN switches.
- The library connects upwards to the local node agent, and the GUI and CLI connect to the center, so it seems logical that the agents should connect to the center as well.

In addition to the specifications described above, there were also several additional requirements for the design and implementation process of COSMOS:

- The software design must allow extensions easily, thus allowing evolutionary steps («rapid prototyping»). This increases the design time, but reduces total implementation time:
  - Software redesign usually requires new implementation, whereas a design extension allows the created software to be reused and only requires features to be added.
  - The intermediate versions are fully functional and can be used by the users.
  - For example, this approach allowed the SAN agent and GUI to be added to the COSMOS design with only a few changes to the already existing software (center and CLI).
- Use custom software only where necessary, and use commodity software wherever possible:
  - There are many established standard software packages and libraries that can be used for free or come with the platform. There is no need to re-create these features.

- Some of the standard libraries behave differently on different platforms (sockets, threads) and these differences must be hidden in «wrappers» – custom software that can make these libraries behave identically between platforms where needed by COSMOS.
- Some features could have been provided by use of standard software. In some cases, the author did not select this path because he did not have enough knowledge to adapt these standards for his own design (as with the MIB that could have been solved with persistent SQL-based databases), and in other cases the available software did not match the required quality (such as COM/CORBA for communication between the software parts).

❖ Less is more. Users prefer fewer features but ones which are practical and useful and work well instead of a greater number of features which are hard to use (or flawed) and will be seldom used anyway.

The following figure illustrates a simplification of the management architecture used in the Swiss-T1 supercomputer, with the three main management toolkits SYSMAN (S) for node management, GRD/Codine (G) for resource management, and COSMOS (C) for SAN and applications.

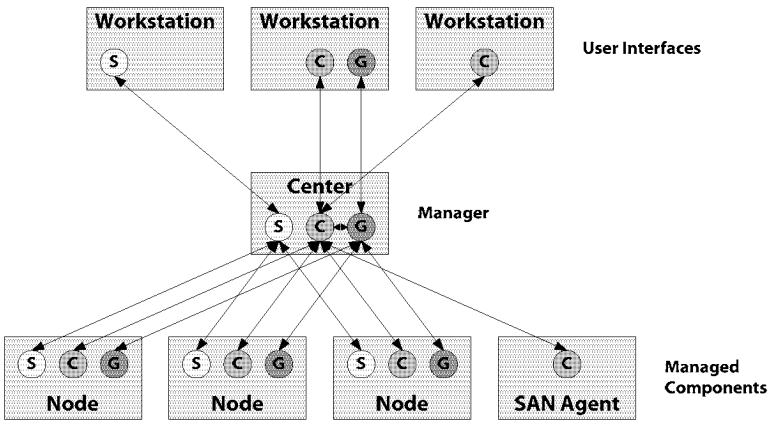


Figure 5.18: Glueware management architecture on the Swiss-T1 with the three main management applications SYSMAN (S), GRD/Codine (G) and COSMOS (C).

The center – as central authority in the supercluster – holds the management information base (MIB). Its structure is created during start-up, based on the configuration files. During operation, the center fills the MIB with data, which has been collected by the SAN agent and the node agents, and of course with the data created by the center's operation as well. The MIB structure is similar to SNMP, where data is stored and accessed in a tree-like structure as in the illustration below. The MIB is stored in the main memory and is lost after the center is shut down or crashes.

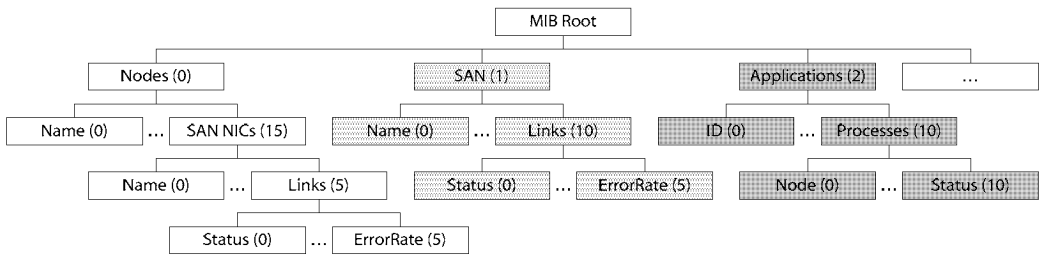


Figure 5.19: Basic hierarchical structure of the COSMOS MIB (see Appendix B.8)

The design process was performed in December 1999, immediately after writing down the specifications. Since the design was flexible and extendible, no major changes were required for the second and third versions, after the specifications had been modified.

5.4 COSMOS Implementation

The implementation of COSMOS was performed between December 1999 and February 2001. It evolved in three stages that were also the three implementation milestones:

- The first version was required by January 2000 for the prototype system installed at SCS. Consisting of few nodes, it was used by the developers of T-Net and FCI/MPI. It was also used on the freshly installed Swiss-T1 at EPFL in February 2000. Based on the needs analysis, only the following components from Figure 5.17 were implemented:
  - The center, with basic application management and node monitoring features.
  - The node agent, with basic application management features.
  - The application library, with all specified features.
- The second version was required for the Swiss-T1 inauguration day on August 23rd 2000. The productive version required more features and components:
  - The center was complete, except for SAN management and GUI support.
  - The node agent was complete as specified.
  - The application library, with some minor enhancements.
  - The CLI for application monitoring.
- The third version was installed on the Swiss-T1 in February 2001. Since EPFL did not order the succeeding Swiss-T2 supercluster, there was no longer any need to create a second-generation COSMOS, and the time saved was invested in completing COSMOS:
  - The center demon was extended with SAN monitoring and GUI support.
  - The node agent demon was slightly enhanced.
  - The SAN agent demon was created for monitoring.
  - The application library was already finished.
  - The CLI was enhanced with SAN monitoring and application control.
  - The GUI was created with some fancy features.

Although the six components of COSMOS evolved through these three versions, this section describes the final version of the software as if it had been developed this way straight from the beginning.

5.4.1 COSMOS Demons: Center, Node Agent and SAN Agent

In technical terms, the three COSMOS demons (center, node agent, SAN agent) have similar functionality and behavior. This is the reason, why they share the same software framework and many parts of the source code. Additionally, such a shared design also allowed implementation and debugging time to be saved.

This demon framework design must cover the following basic functionality:

- Reliable communication protocol with extendable message handling.
- Safe multithreading and signal handling within the demon.
- Internal interfaces for those parts that are different between the three demons.
- Access to and storage of the global (center) or local (agents) MIB.

This basic functionality leads directly to a three-layer framework, as illustrated and described below.

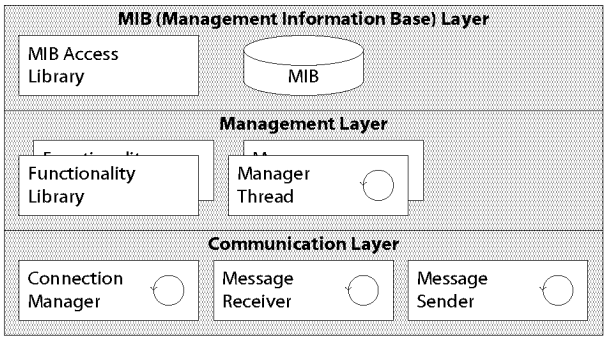


Figure 5.20: Basic COSMOS demon structure

The **MIB Layer** includes the MIB itself and some access procedures.

The **Management Layer** includes those parts that differ between the demons:

- The **Functionality Libraries** contain the management knowledge. There is one library per subsystem that is managed by the demon. The library functions are not only called upon by the associated manager thread, but also by other manager threads and the threads of the communication layer.
- The **Manager Threads** actively manage the components and MIB of the associated subsystem. There is usually one thread per subsystem that is managed by the demon.

The **Communication Layer** takes care of all the socket-based communication of the framework. The communication between the center and all agents is very important. This is the reason why a lot of time was invested in creating an effective and efficient design and implementation of this layer. The basis for this layer was a socket library created for a former version of FCI (for EasyNet), with added features for reliable communication. It consists of the following three threads:

- The **Connection Manager** that accepts incoming and creates outgoing connections.
  - It creates socket connections for outgoing communication.
  - It creates a socket that accepts incoming connections.
  - All connections are checked regularly to see whether they are active or broken.
- The **Message Receiver** that checks all active connections for incoming messages.
  - It traverses all connections and checks for incoming messages.
  - Incoming messages are checked and forwarded to the appropriate manager.
- The **Message Sender** that sends queued messages through active connections.
  - It checks all connections to see whether there are messages that need to be sent.
  - New messages are sent immediately if the connection is active.
  - Messages with a «time-out» acknowledgment are sent again with a «retry» flag.
  - The thread can be activated by any manager thread when messages need to be sent.

There would have been several commercial or open-source products available for use with the communication layer (such as COM or CORBA), but there were some special requirements that made a custom implementation necessary.

- Broken connections must be able to be detected immediately. If an application or node crashes, standard products need minutes or hours before they detect such a failure.
- When problems arise, messaging calls block the calling thread for several seconds. Since the threads have important things to do, the blocked time must be reduced and the threads must be reactivated immediately when problems arise. Standard products do not permit the reduction of timeout limits for messaging calls.
- COSMOS uses a message format that allows quick content decoding. The standard products also include such mechanisms, but they are of no additional benefit to COSMOS.
- COM and CORBA (requiring C++) are not designed to be used in demons (requiring C).

It turned out that using available standard products would increase implementation complexity and development time and would not add any measurable advantage.

5.4.1.1 COSMOS Center

The COSMOS center demon, called `cosmos_center`, is started on a front-end computer, and accepts incoming connections of the node and SAN agents, as well as the user interfaces. It is the main authority that decides what is wrong or right.

There can only be one active center demon per supercluster. It performs the following checks before start-up and aborts if one of these fails:

- The center demon checks to see whether its own computer is listed as center in the configuration files.
- The center demon checks to see that no center is running on the local computer.
- The center demon checks to see that no center is running in the supercluster.

The COSMOS center demon has the following structure:

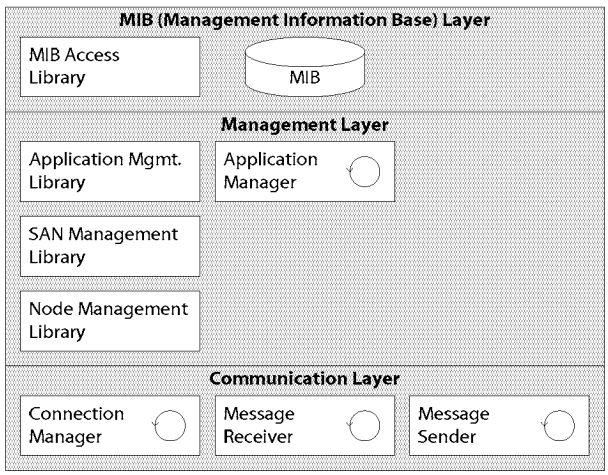


Figure 5.21: COSMOS center demon structure

The **MIB layer** holds the complete supercluster MIB that is queried by the center demon itself and by the user interfaces. It is implemented as a hierarchical tree that could be re-implemented using SQL-based relational databases.

The **Management Layer** of the center contains the following:

- The **Application Manager** manages the applications running on the supercluster.
- The **Application Management Library** contains the knowledge for managing applications.
- The **SAN Management Library** contains the knowledge for T-Net switch monitoring.
- The **Node Management Library** contains the knowledge for fault handling.

The **Application Manager** regularly traverses the active applications, checking their status:

- The application is in the start-up process until all processes have started and joined. If this start-up takes too long, some of the processes are lost or aborted before they have joined. The application must be aborted after a predefined period of time to save resources.
- The application is in the abortion process if the user (or GRD/Codine) sends an abortion request or a process either crashes or detects a condition that requires an abortion. The application usually saves the results for later recoveries to storage, but if the abortion takes too much time, the application is wiped out because resources are spoiled.

The **Communication Layer** has some important functions, as there is no node or SAN manager thread in the center demon.

- The availability of the nodes is determined by the connection status to the node agent.
  - If the node agent connects, the node resources are enabled for applications. The center expects that the resources on the node are unused and free.
  - If a node agent disconnects unexpectedly, the node is assumed to have crashed and applications that had processes running on the node are aborted.
- The monitoring data delivered by the SAN agent is inserted into the MIB.

The COSMOS center has a source tree size of about 1 MBytes (50 000 lines of code, of which 30% is shared framework). The executable size is below 400 kBytes.

5.4.1.2 COSMOS Node Agent

The COSMOS node agent demon, called `cosmos_agent`, is started on each node computer, and accepts incoming connections of the application processes through the COSMOS library. There can only be one node agent demon active per node computer – a second instance would be terminated by the center demon.

The COSMOS node agent demon has the following structure:

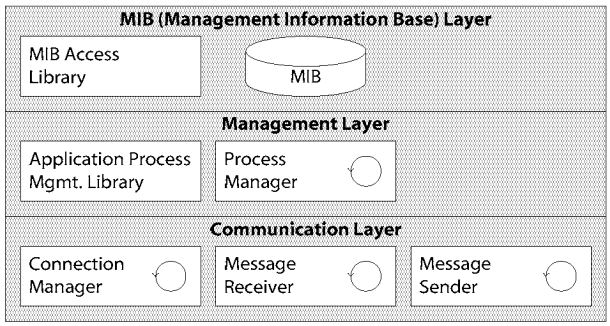


Figure 5.22: COSMOS node agent demon structure

The management layer of the node agent contains the following:

- The **Process Manager** manages the application processes running on the node.
- The **Application Process Management Library** contains the knowledge to manage processes of parallel applications.

The process must follow a certain protocol to be accepted as a FCI/MPI application process:

- It must connect to the node agent after being started on the node.
- It must be authenticated as a valid FCI/MPI process within a period of time, otherwise the connection will be closed by the node agent:
  - Using a special SHMEM procedure, it must discover its PID (for process control, such as sending abortion signals).
  - It must send the mpirun parameters, which are forwarded to the center.
- The process must wait until the center returns the permission to start (with configuration data for FCI/MPI and T-Net) or aborts this start-up (either time-out or blocked resources).
- After start-up, the processes use barriers for global synchronization of FCI/MPI.
- The processes start operation and continue, until they either send a finalization message on success, or until they send an abortion message if something went wrong.

The process manager traverses the active processes, and checks their current status. Anomalies are immediately sent to the center which decides whether the application is to be continued or aborted. If a process disconnects unexpectedly, the center is informed and aborts the application immediately on the whole supercluster.

The node agent demon could also connect to the SAN NIC driver, but since the drivers and SAN NICs do not allow management actions, this functionality was not implemented.

The COSMOS node agent has a source tree size of about 1 MBytes (50 000 lines of code, of which 30% is shared framework). The executable size is below 400 kBytes.

5.4.1.3 COSMOS SAN Agent

The COSMOS SAN agent demon, called `cosmos_sagent`, is started on a computer that can access the SAN switches through the LAN and terminal servers. There can only be one SAN agent demon active per supercluster – a second instance would be terminated by the center demon.

The COSMOS SAN agent demon has the following structure:

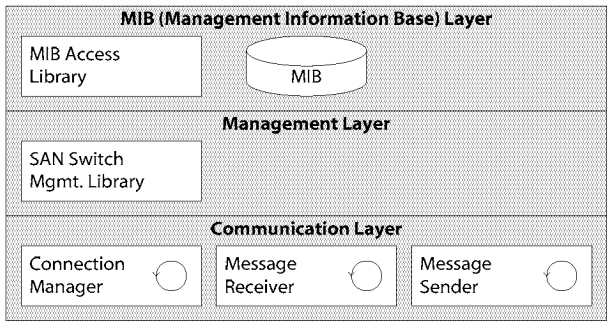


Figure 5.23: COSMOS SAN agent demon structure

The management layer of the SAN agent only contains the **SAN Switch Management Library** which contains the knowledge for monitoring the T-Net switches. There is no additional thread (such as a «SAN Switch Manager») in the framework, since all library calls are made by the connection manager and the message receiver threads. There would be a SAN Switch Manager thread if the SAN agent were to do more than just monitor the switches, such as analyze non-fatal errors and manage the T-Net switches, or support the center demon for full coverage of the T-Net switch management. Since the OS of the T-Net switch can not reliably support more than monitoring, the management part was not implemented for COSMOS.

The COSMOS SAN agent has a source tree size of about 600 kBytes (30 000 lines of code, of which 50% is shared framework). The executable size is below 300 kBytes.

5.4.2 COSMOS Application Library

The COSMOS library, called `libcosmos.a`, integrates the application process into the supercluster management. The library connects to the local node agent, authenticates itself as a valid FCI/MPI process, and exchanges management messages.

It consists of a data structure that describes the application, as received by the center demon, and four simple calls. FCI/MPI has been re-designed to support the COSMOS environment, and other communication libraries (such as MPICH) can be readily adapted.

Integration of the processes into the management allows for full control over the applications, including monitoring the applications and reliably terminating the applications without orphaned processes blocking the supercluster resources.

The four COSMOS library calls are the following:

- **Cosmos_Init()** initializes the data structures of the library, installs the signal handler, connects the process to the local COSMOS node agent, authenticates the connection, sends the startup request and waits for the response. The response is parsed and – if the application is allowed to start – configuration data is returned to FCI/MPI to open the SAN NIC device and initialize its own data structures.
- **Cosmos_Barrier()** enters the process into a global barrier. The call returns when all processes have entered the barrier and continue to work. Global barriers are used

by FCI/MPI to synchronize management actions, such as opening the SAN NIC device, setting up the SAN NIC registers or starting or stopping SAN-based services.

- **Cosmos_Finalize()** informs the management subsystem that the process has finished its job successfully and that the application is about to end. There is no time limit between the first and last finalize call of the application processes.
- **Cosmos_Abort()** informs the management subsystem that the process has detected a failure that requires a global abortion of the application. The processes of the application are terminated by the node agents.

Additional calls can be implemented for the management of process groups (MPI calls them communicators, the hardware designers multicast groups) such as creation, deletion and merge. Also user-level calls are possible, such as calls that tell the management subsystem the completion status of the application (such as «80% completed») or store text into the application log.

The COSMOS application library has a source tree size of about 300 kBytes (15 000 lines of code). The library size is about 1.1 MBytes.

5.4.3 COSMOS User Interfaces

All previously presented components are invisible to the users and administrators. The functionality operates in the background. The two user interfaces allow COSMOS to step out of the darkness, and allow him to see what is happening or has happened. The CLI was available from the first day, and the GUI was added at the end.

5.4.3.1 COSMOS CLI (Command Line Interface)

The COSMOS Command Line Interface, called `cosmos_cli`, can be started on any computer that is allowed to connect to the COSMOS center. The CLI has been implemented for computers running COMPAQ Tru64 UNIX, Linux or Microsoft Windows NT. The center has no limits with regard to the number of connected CLI applications, because its request-based design creates low computing load and communication overhead on the center.

The CLI writes a prompt and waits for user entry. The CLI commands have a similar structure to the COSMOS MIB. The first token specifies the management module (`node`, `san`, and `app`, extended with the simple commands `exit` and `help`). The second token specifies the entity, the third token usually a sub-tree of the MIB etc.

The entered command is parsed and sent as a MIB data request to the center. The center queries the MIB and replies with the data – except for the application abortion command. The CLI waits for the reply which is parsed and printed in a readable format.

```
COSMOS: app
Number of applications: 1402

COSMOS: app 1 status
Application   1:   Status  4 (Aborted)

COSMOS: app 1 rank 1 acause
App   1 Rnk   1:   Received SIGSEGV (Segmentation Violation, programming Error)
```

Example 5.1: Some request commands of the CLI

Some data can be requested as bulk information, such as an overview of all currently running applications or all information about a particular application.

```
COSMOS: app 0
Application   0:        ID  180388626474 (42/42)
Application   0:      Name  ./mpitest9
Application   0:      Path  /usr/scs/nemecek/tnet/OS_DEC_UNIX
Application   0:     Owner  11067 (nemecek)
Application   0: Barriers  1
Application   0:  Created  13.03.01 16:44:23
Application   0:  Started  13.03.01 16:44:24
Application   0:  Barrier  13.03.01 16:44:24
Application   0: Finished  --.--.-- --:--:--
Application   0:   Status  2 (Started)
Application   0:     Size  2
Application   0:     Slot  Host    PID  Time Registered    Time Finished      Status
Application   0:        0  gs3     800  13.03.01 16:44:24  --.--.-- --:--:--  Started
Application   0:        1  gs3     842  13.03.01 16:44:23  --.--.-- --:--:--  Started
```

Example 5.2: Request complete application information of the CLI

The CLI is simple to use and was very helpful for error and bug tracking by the administrators, the communication library developers and the author. It not only consumed few cycles on the center, but also required low bandwidth, allowing it to be used with slow modem connections.

The COSMOS CLI has a source tree size of about 600 kBytes (30 000 lines of code). The executable size is below 200 kBytes. Its manual can be found in Appendix D.

5.4.3.2 COSMOS GUI (Graphical User Interface)

One of the first requests of the Swiss-Tx project managers was for a graphical user interface with flashing lights and boxes, showing the current performance of the whole supercluster. Although the CLI was effective for the administrators and users, the presentation of information as text was not satisfying and the GUI was created as the last component of COSMOS.

The COSMOS Graphical User Interface, called `cosmos_gui`, can be started on any computer that is allowed to connect to the COSMOS center. The GUI has been implemented for computers running COMPAQ Tru64 UNIX and Linux. The center has no limits in regard to the number of connected GUI applications, but because it creates a lot of repetitive traffic for on-line monitoring, only one or two should be used at the same time.

The COSMOS GUI has two threads: One thread handles the user actions and the other updates the local MIB copy of the GUI. The MIB thread uses the available bandwidth to the center efficiently by fetching only those parts that are requested by the user, and by caching those parts locally that will not change anymore (MIB data of already terminated applications).

The **Configuration** tab is shown first when the GUI is started. The user can enter the center name and port for connecting to the center, as well as the MIB update rate. Although a user name and password can be entered, this feature was not supported. The other widgets allow the layout of the contents of the SAN topology tab to be configured.

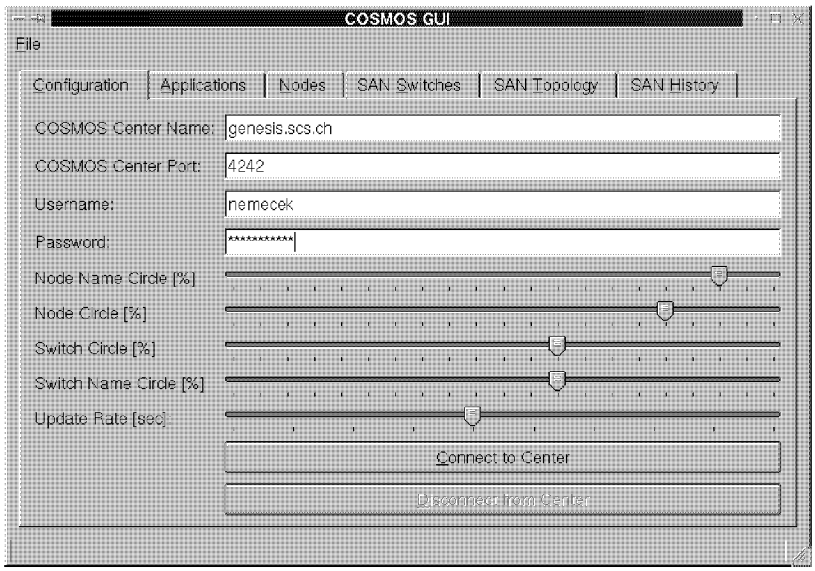


Figure 5.24: COSMOS GUI configuration tab

In the **Applications** tab, the upper half displays the application records of the MIB and the lower half shows the process information of the selected application.

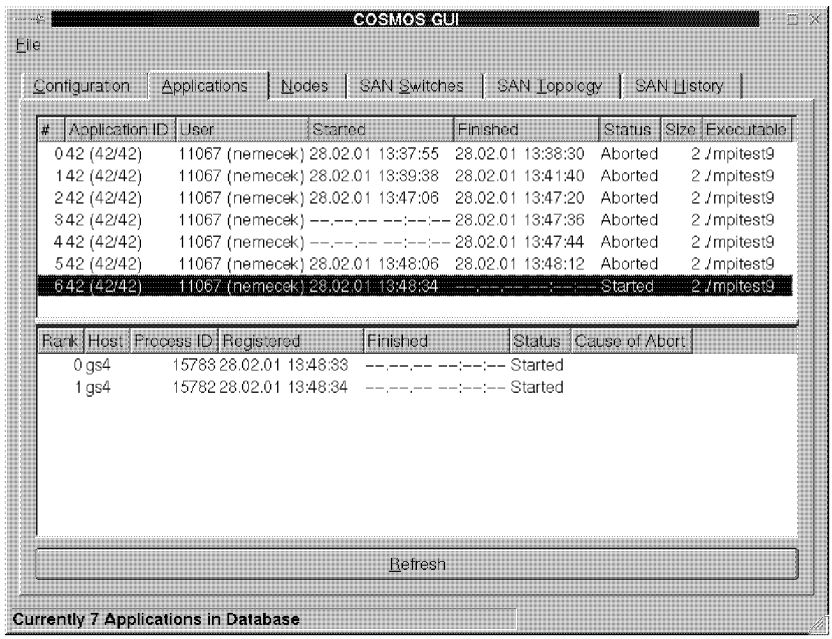


Figure 5.25: Applications Tab

Every application is written on one line and is updated if selected or if the update thread detects a change of the COSMOS MIB and has downloaded the updated application records. In the lower half (process information), each line represents one process and contains the rank number, the node host name, the process ID on that node, the time of registration and termination, the current status and the abortion cause. The context menu (accessed by clicking on an application with the right mouse button) allows the selected application to be aborted (after a confirmation request).

The **Nodes** tab shows all nodes (with their status) in the upper half and their slots (with their status and currently owned processes) in the lower half.

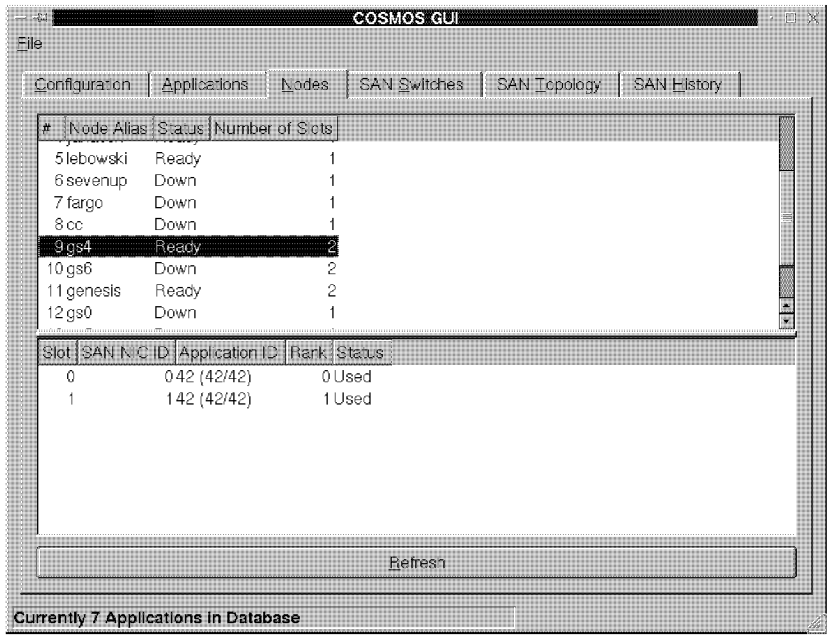


Figure 5.26: Nodes Tab

The content on this page should correspond to the current status of GRD/Codine. This page was the first page that was referred to by the administrators when applications began to be aborted without any visible reason. GRD/Codine tends to lose applications, and schedules new applications for start-up although the resources are already in use. This page allows the application currently blocking the resources to be detected and the application to be manually aborted using COSMOS.

The **SAN Switches** tab shows all switches (with status, temperature and basic configuration) in the upper half and their ports (with current status and monitoring data) in the lower half.

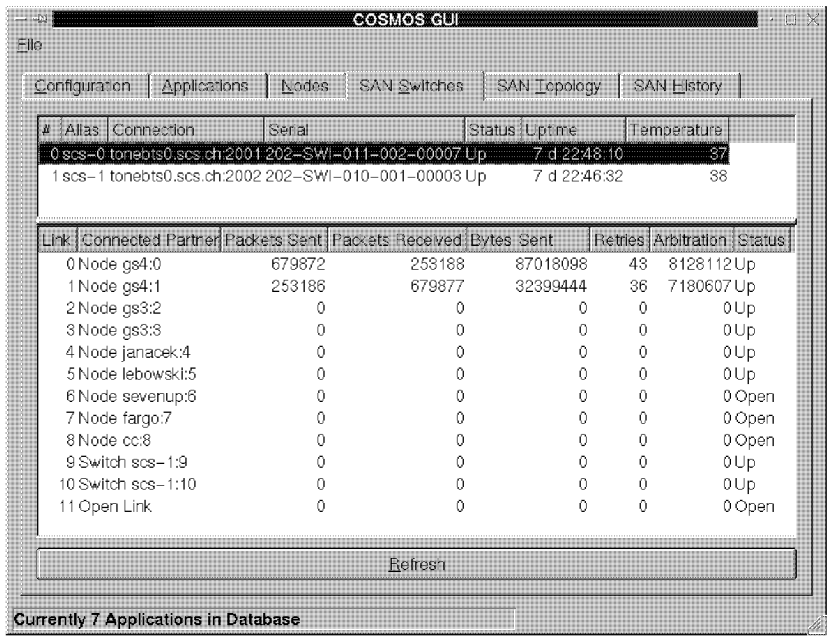


Figure 5.27: SAN Switches Tab

The content allows numerical information about the current load of the SAN. This was the first page that was inserted into the GUI after the SAN agent had been implemented. It enabled information regarding the quantity of data being sent over the network links to be displayed in absolute figures. This page was very important for the managers, since it made it possible to see remotely that T-Net indeed allowed about 100 MBytes/s to be transported.

The **SAN Topology** tab shows the physical connections between nodes and switches. The top selector allows the data set being displayed in the topology plane to be selected. The possible selections are «*Bandwidth [Bytes/s]*», «*Bandwidth [Packets/s]*», «*Retries [Packets/s]*» and «*Status*». The *Refresh* button at the bottom updates the topology plane immediately; otherwise the data is updated regularly by the update thread (usually every 1 to 10 seconds).

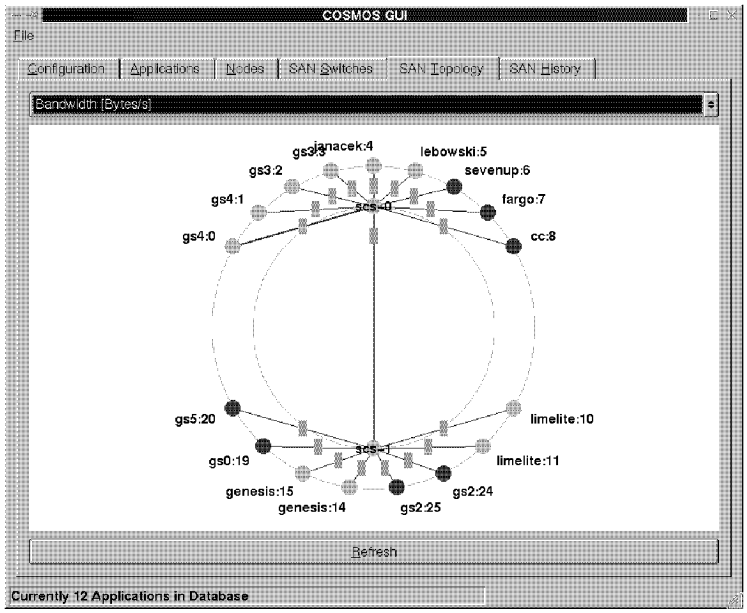


Figure 5.28: SAN Topology tab of the testing cluster

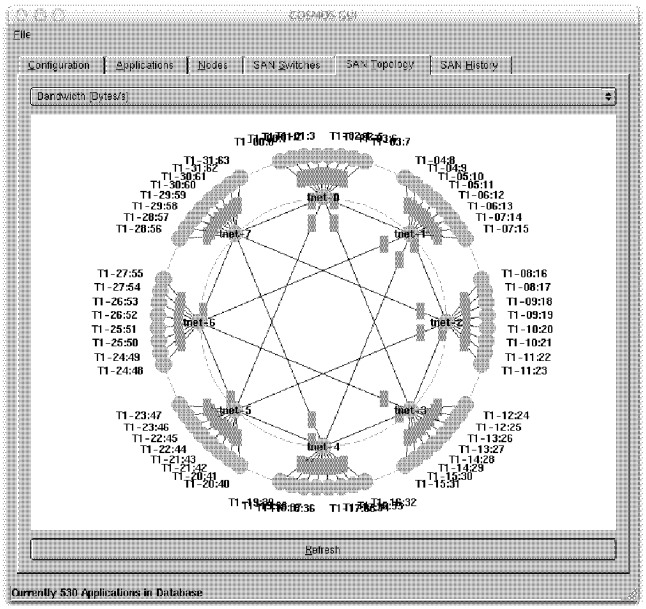


Figure 5.29: SAN Topology tab of the Swiss-T1

The SAN switches are placed on the inner circle and the SAN NICs of the nodes on the outer circle. The color of the nodes and switches identifies its current status (red means switched off, green means ready). The lines between the switches and nodes represent links.

The width and color of the links vary depending on the data set and data value. The small boxes in the links contain gauges that also represent the value.

- For the bandwidth record sets, the line varies from blue (size = 1 pixel) when no traffic is sent through the link, and red (size = 10 pixels) when the traffic is at its maximum rate.
- For the error rate record sets, the line varies from green (size = 1 pixel) when no errors occur, and red (size = 10 pixels) when only errors occur.
- For the status record sets, the line changes from green when the link is up to red when the link is down.

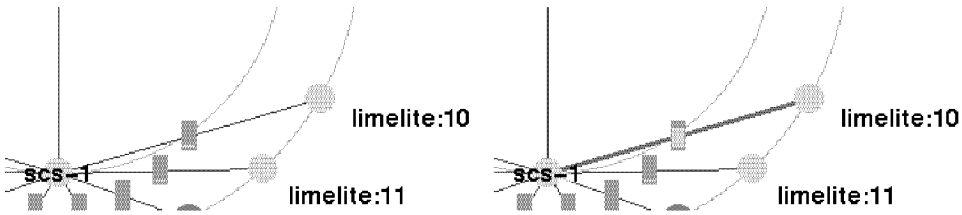


Figure 5.30: Varying line size and gauge fill for low (left) and high value (right)

The **SAN History** tab allows the data about the SAN to be viewed in the history view. This view is very useful for users (e.g. when they want to analyze the communication behavior of their applications) and for administrators (e.g. when they want to analyze error rates in relation to communication performance).

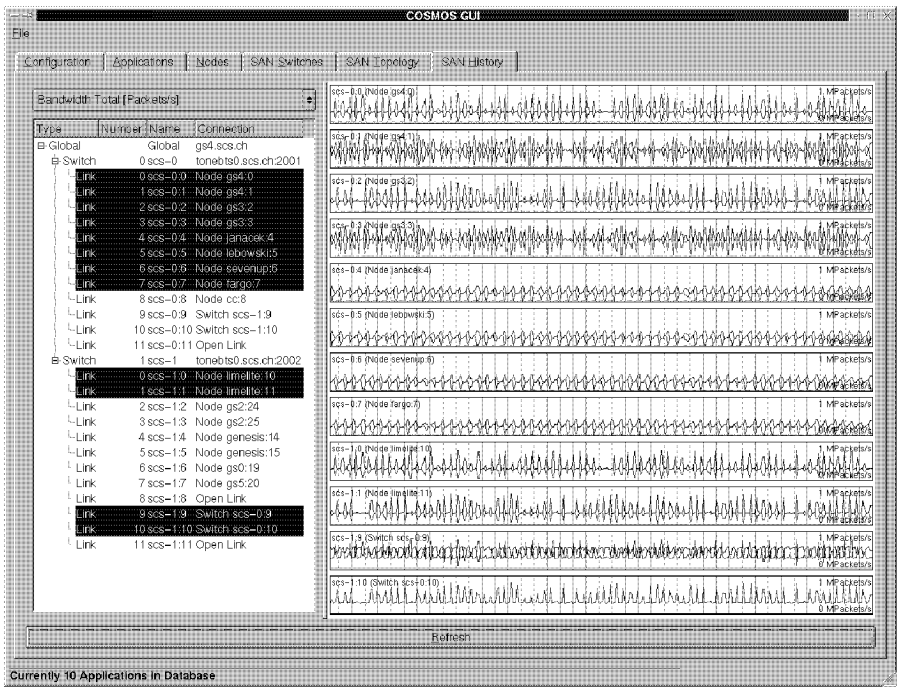


Figure 5.31: SAN History tab

The top selector allows the data set being displayed in the topology plane to be selected. The possible selections are «*Bandwidth In [Packets/s]*», «*Bandwidth Out [Packets/s]*», «*Bandwidth Total [Packets/s]*», «*Bandwidth In [Bytes/s]*», «*Bandwidth Out [Bytes/s]*» and «*Bandwidth Total [Bytes/s]*». The *Refresh* button at the bottom updates the history plane immediately; otherwise the data is updated by the update thread.

On the left-hand side, SAN ports can be selected which are then displayed on the right-hand side. This selection is grouped by the SAN switches and organized as a tree. Every port is described with its connected entity for convenient selection.

On the right-hand side, the history of the selected record set of the selected links is displayed. The charts move from right to left, and show the most current data on the right border. The red lines display data moving into the switch, the blue lines display data moving out of the switch, and the green lines display retransmissions.

The current implementation stores 1000 sample points, which gives a total view of about 10 minutes with a 5-second update rate. Since the COSMOS center MIB does not include archived monitoring data, the content is lost when the GUI is closed.

The topology and history tabs impressed the administrators, developing team members, and many administrators of other supercomputers. These two tabs (plus the application tab) showed the importance and advantage of integrated, comprehensive supercluster management.

The COSMOS GUI has a source tree size of about 500 kBytes (25 000 lines of code). The executable size is about 1.3 MBytes. Because of the local MIB copy with archive, the GUI consumed up to 250 MBytes for the Swiss-T1.

5.4.4 Implementation Summary

Does the implementation match the previously presented design, the specifications, the requirements and the needs of users? The author believes so for the following reasons:

- The implemented software uses the designed center-agent architecture and includes the six software components of the design:
  - The center can be run on either front-end computer.
    - All agents and the user interfaces are connected.
    - It holds the complete MIB and provides it to the user interfaces.
    - It is the only instance of management authority for applications.
    - It detects node faults and aborts applications if processes are lost.
  - The node agents run on all node computers.
    - They only use a few resources of the node computer.
    - They allow the center to fully control all application processes.
    - They detect process faults and abnormal behavior.
    - They monitor the node and application processes.
    - They provide configuration data for the processes.
  - The SAN agent runs on the opposite front-end computer from the center.
    - It provides the collected monitoring data of the T-Net switches to the center.
  - The application library integrates FCI/MPI-based processes into COSMOS.
    - The processes are managed by COSMOS.

- The FCI/MPI implementation is closely coupled to and depends on COSMOS.

➢ The CLI is used for text-based monitoring and application abortions.
- It enables control and monitoring to take place using any low-bandwidth connections.
- It was available from the first day.

➢ The GUI is used for graphical monitoring and application abortions.
- It presents the vital values of the whole supercluster graphically.

❖ Commodity software libraries are used where possible (sockets, Qt, pthreads).
❖ COSMOS is closely coupled to T-Net and FCI, using the same source tree.
❖ Although designed for the Swiss-T1, COSMOS could also have been used on the Swiss-T2 supercluster without and changes being needed.
❖ COSMOS interfaces well with the other management applications.
❖ The development was finished on-time for each milestone.

The finished product was in use without problems or changes until the Swiss-T1 supercluster was taken out of service in late 2003. It matched the needs of the users and administrators who were satisfied due to having information about the SAN, additional information about the applications that GRD/Codine could deliver, and a reliable tool for checking resource availability and aborting applications.

5.5 Qualitative and Quantitative Evaluation

Chapter 3 introduced qualitative and quantitative evaluation tables. The architectures that were presented in the same chapter were evaluated using these tables, as well as the related work in chapter 4. It is therefore only fair to also evaluate COSMOS using these tables.

Criteria	Value	System Size [Nodes] (+ 1 Switch per 4 Nodes)			
		100	1 000	10 000	100 000
Required bandwidth for monitoring (92 B/Node)	1 Hz	9.2 kB/s	92 kB/s	920 kB/s	9.2 MB/s
	1 kHz	9.2 MB/s	92 MB/s	920 MB/s	9.2 GB/s
Duration startup/shutdown*	10 s/Comp.	21 min	3 hrs	1 day	14 days
Duration OS download*	30 s/Node	50 min	8 hrs	3 days	34 days
Duration group creation*	0.1 s/Comp.	13 sec	2 min	21 min	3 hrs
Duration process spawn	0.5 s/Node	1 min	9 min	83 min	14 hrs
Duration fault detection	2 s	2 s	2 s	2 s	2 s
Duration config change*	1 s/Comp.	2 min	21 min	3 hrs	1 day

Table 5.4: Quantitative evaluation table for COSMOS (estimated values have an asterisk)

The monitoring data message that the SAN agent sends to the center has a size of 2 936 Bytes for the Swiss-T1, leading to 92 Bytes per node (32 nodes, 8 T-Net switches). As the T-Net switches can only send one monitoring message per second, there is enough bandwidth (using Fast Ethernet) for up to 100 000 nodes. If node monitoring (with CPU usage and SAN NIC monitoring) were to be implemented, the monitoring data size would grow to 150 Bytes per node, which would limit the system size to some

10 000 nodes, which is equivalent to the size of the current ASCI superclusters. The efficiency of COSMOS monitoring is good.

The spawn mechanism uses a remote shell script that requires 0.5 seconds per node (15 seconds on the Swiss-T1, estimated 1 minute for the Swiss-T2 with 126 nodes). The estimated 9 minutes for the system with 1 000 nodes was already the maximum that the users would accept. Larger systems requiring hours using this mechanism would either need a more sophisticated script that scales, or another mechanism. An experimental system was created that used the node agent for spawning (instead of the remote shell script) by request of the center. Such a mechanism could spawn the application simultaneously, and would have been implemented for the Swiss-T2.

Since COSMOS uses persistent connections, it immediately detects disconnected agents and processes. Faults are detected within a few seconds, independent of system size. The COSMOS connection strategy therefore permits much lower fault detection duration than the standard glueware approach, where components are usually «pinged» sequentially.

The other values in the table have been estimated (marked with an asterisk), since the functionality was not implemented in COSMOS. If the functionality were to be implemented, it would have the following behavior:

- System startup or shutdown would happen through terminal servers, using system console for nodes, and remote power switches for the T-Net switches. Since the communication is text-based, each component would require about 10 seconds for the power control. If the components were controlled serially, the duration would increase linearly with size.
- Downloading a new OS to the nodes (or a new firmware to the T-Net NICs or switches) would be done by saving a new image to the node's storage. This process would take about 30 seconds per component. Since the bandwidth of the center (or the computer where the image is stored) is limited, the number of concurrent downloads would be limited, so the download would be serial and the update time would grow with system size.
- The creation of a group would trigger two different processes in T-Net: The calculation of new routing tables for all switches, and the updating of the «allowed destination ID table» on the NICs. The time for routing table calculation and updating the configuration data would grow with system size.
- Estimating the time for updating the configuration would be difficult, because the number of affected components and duration per component would differ from one update reason to another: One broken node only requires applications to be restarted, but a broken SAN switch may require all routing tables to be updated and many nodes to be shut down.

These values show that COSMOS is only reasonable for superclusters with some hundred nodes. The typical tasks do not scale, but with additional proxies and more features implemented into software (instead of scripts), the tasks would scale well. Assuming one proxy per 100 nodes (or 100 proxies) and additional implemented features, the table would look as follows:

Criteria	Value	System Size [Nodes] / Proxies / Proxy Layers			
		100	1 000	10 000	100 000
		0 / 0	10 / 1	100 / 1	1010 / 2
Required bandwidth for monitoring (92 B/Node)	1 Hz	9.2 kB/s	92 kB/s	920 kB/s	1 MB/s
	1 kHz	1 MB/s	1 MB/s	1 MB/s	1 MB/s
Duration startup/shutdown*	10 s/Comp.	21 min	21 min	21 min	21 min
Duration OS download*	30 s/Node	50 min	52 min	57 min	60 min
Duration group creation*	0.1 s/Comp.	13 sec	13 sec	13 sec	13 sec
Duration process spawn	0.5 s/Node	1 min	1 min	1 min	1 min
Duration fault detection	2 s	2 s	2 s	2 s	2 s
Duration config change*	1 s/Comp.	2 min	2 min	2 min	2 min

Table 5.5: Quantitative evaluation table for COSMOS with assumed proxies

The system with 100 nodes does not need proxies, and systems between 1 000 and 10 000 nodes only need one layer of proxies. The system with 100 000 nodes has one lower layer with 1 000 proxies and one upper layer with 10 proxies. The proxies would collect the monitoring data and only forward statistical data to the center, which would be limited to 1 MBytes/s. The user interfaces would connect to the proxy to get full monitoring data coverage – or at least as much as their network connections can bear.

The spawning mechanism could take advantage of the proxies. The remote shell scripts could be executed by the (lower level) proxies in parallel with 100 nodes per proxy, leading to the one minute spawning time.

In addition the estimated values take advantage of the proxies, since the tasks that are performed by the center are now distributed to the (lower level) proxies which handle their 100 connected nodes:

- Since each proxy has 100 nodes to manage, system startup or shutdown would happen within 18 minutes (with 10 seconds per node).
- Downloading the new OS to the nodes would happen in two phases. First, the OS would be downloaded to the proxies, and then the proxies would download the OS to their agents. With efficient programming, downloading the OS to the proxies would grow in logarithmic scale to the proxy count in each layer. Downloading the image from the (lower level) proxies to the agents would take 50 minutes. The sum of these two phases would be about one hour.
- If the calculation of new routing tables and configurations were to be delegated to the proxies (only asking the center for permission to perform the update in the associated nodes and SAN switches), these tasks could be massively accelerated. If the center was required to calculate and distribute these changes, the proxies could not improve the figures of the previous table.

The quantitative evaluation table shows that COSMOS scales well up to a system size of about 1 000 nodes without proxies. With proxies, it would also be possible to use COSMOS for systems of some 10 000 nodes.

Criteria	Quality	Comments
Monitoring Synchronization	0	Available for node, SAN and application
Monitoring Context	0	Available for node, SAN and application
Development Time & Cost	+	Quickly implemented
Management Overhead	+	No additional hardware and software
Reliability & Availability	–	Center is single point of failure
Scalability	0	Efficient, requires proxies for large systems
Security & Vulnerability	0	Authentication available and stable
Flexibility	–	Supports UNIX/Linux, but T-Net only

Table 5.6: Qualitative evaluation table for COSMOS

Since monitoring only covers SAN, nodes and applications, the synchronization and context is only available for these subsystems. Monitoring of the other subsystems (and performance monitoring of the nodes) is available with other tools, but without context and synchronization to COSMOS. The flexible design of COSMOS would have allowed further coverage, but was not required.

The development efforts for COSMOS were very low. No additional computers were used for COSMOS, because the two front-end computers were required for the storage subsystem. The node agent only consumed few resources on the node, so the overhead caused by COSMOS was very low as well.

The center is the single point of failure – if it crashes, the whole supercluster will be blocked. Although the center of the Swiss-T1 crashed only twice during the full operation time, the limited availability and reliability is the weak point of the design. For larger systems (such as the Swiss-T2) a master/backup strategy with persistent database would have been necessary.

For systems of up to some 100 nodes, the system is efficient enough as shown above, but for larger systems of some thousand nodes, proxies are required, thus increasing the scalability. If proxies were inserted into the design, the user interfaces would not only connect to the center, but also to the proxies to download current monitoring performance, since the center would then only store archived data for statistical reasons.

The design removed much potential vulnerability, making the management safe, but more features would be required if COSMOS were to be turned into a professional product (for example message data encryption, user permissions).

COSMOS was designed for T-Net SAN products exclusively, and implemented for Linux and UNIX. On the one hand, these limitations reduce the flexibility in supported products, but on the other hand, the limitations were selected intentionally because COSMOS is research management software used for research platforms, where no flexibility is required. Of course, this reduction also reduced the development time.

The qualitative evaluation table shows that COSMOS performs a little bit better than the usual glueware design approach, due to the fixed destination platform and available software.

Criteria	Quality	Comments
Design, Installation & Configuration	–	No support
Power Management Implementation	–	Script-based, slow and unreliable
Monitoring Transport to User Interfaces	+	Efficient transport, Cache MIB in GUI
Process Groups / Routing Tables Creation	–	Not implemented
Application & Process Control	+	Integrated in COSMOS
Detection & Handling of Faults	0	Fast detection, but hardcoded handling

Table 5.7: Typical tasks evaluation table for COSMOS

The support for typical management tasks is low, since many different tools are used which are not integrated into one user interface. Design, installation and configuration were fully manual processes without any support from COSMOS and the other management tools.

Power management was slow and unreliable using shell scripts, but this was not a problem because the Swiss-T1 was small and the power was seldom switched on and off. Implementation into software would increase reliability and performance, so in this point, it would be possible to increase the quality grading to a «+».

The problem of transporting monitoring data from the center to the user interfaces is very efficiently solved in COSMOS, because the GUI and CLI each contain partial copies of the system MIB. The user interfaces only fetch the MIB data from the center that is requested by the user. To reduce the network traffic further, the user interfaces only fetch MIB data that is already available locally for a second time, if there is a chance that the data has changed.

The efficient implementation of MPI groups requires adaptations to the routing tables of the SAN switches. Since this feature was not reliably available for the T-Net switches, COSMOS did not support the management of MPI groups and routing tables. If hardware-based multicast is not implemented, the message sender must send the message to each destination individually – which is not a serious problem in small systems such as the Swiss-T1. Because the calculation of new routing tables (in case of faults or the creation of groups) is simple, this feature could be added later which would increase the quality grading to a «+» in this point.

Since COSMOS was built for managing applications and their processes, this functionality was well covered by COSMOS and fully integrated. The detection of faults was also well integrated, but their handling was limited due to the unreliable detection of T-Net faults.

The evaluation table of the typical tasks shows that COSMOS covers the features described by the specifications very well.

5.6 COSMOS Experiences

COSMOS was installed on the following systems:

- The Swiss-T1 supercluster at EPFL with 33 nodes (Compaq Tru64 UNIX).
- The Baby-T1 supercluster at EPFL with 8 nodes (Compaq Tru64 UNIX).
- The T-Net test and development system at SCS with 12 nodes (UNIX and Linux).
- The Intercept test and development system at SCS with 24 nodes (UNIX and Linux).

COSMOS with its user interfaces has some features that were highly appreciated and well used by the users and administrators:

- It is very stable and reliable. While hardware, firmware and the operating system caused the majority of problems at the beginning, lost processes of the resource management troubled the days of the administrators during the productive phase. It was very seldom that the system had to be rebooted due to COSMOS-originating problems.
- COSMOS is very easy and straightforward to use, requiring only a few minutes introduction to the system configuration and the usage of CLI and GUI.
- It facilitates a quick topological view to check whether all nodes and switches are present or not. Faults are immediately detected (within the TCP/IP-based keep-alive functionality) and shown.
- Since the SAN link technology was new, the fact that some cables were weak only showed up under full load with higher retransmission rates than the other links. The monitoring mechanism allowed those bad cables to be detected and they were replaced during scheduled downtimes or unexpected downtimes caused by other sources.
- Because GRD/Codine regularly lost application processes, subsequently spawned applications waited for the blocked slot resources and were aborted after a time-out. Those processes could not be aborted using GRD/Codine. COSMOS allows immediate and reliable abortion of all lost processes within a second – this feature was regularly used on the Swiss-T1.
- COSMOS logs the abort reasons. The administrators could check the application termination code (successful or unsuccessful). If a user had too many abortions, he was forced by the administrators to ask for development support or attend programming courses.
- The strength of COSMOS is that the process is connected to the COSMOS node agent only. All other management attempts of MPI are based on global communication, where all processes have open connections to all other processes of the same application, and leading to a high inter-process communication overhead.
- The execution of the MPI test suite was accelerated as soon as COSMOS became available. The distribution, finalization and abortion process is much faster than the previous FCI-internal solution using global socket-based communication.

In combination with the other management tools of this glueware software architecture, the supercluster was well manageable and efficiently used for many applications, in-

cluding the calculation of the optimal design of the high-performance racing yacht Alinghi (winner of the America's Cup 2003) [Saw02].

Other parallel applications that were executed on the Swiss-T1 include industrial flow simulations [Vos00] based on parallel discrete element methods [SCa99] or smoothed particle hydrodynamics [SCH00], structural mechanics simulations [Vol00], molecular dynamics simulations of modulated crystalline structures [PCB02], and many more.

6 Conclusions

One of the problems of being a pioneer is you always make mistakes and I never, never want to be a pioneer. It's always best to come second when you can look at the mistakes the pioneers made.

Seymour Cray

The successful supercomputer architect Seymour Cray preferred not to be a pioneer. He preferred analyzing existing solutions, improving their advantages, and reducing the effects of their disadvantages. He was not the first to use vector processing units, symmetrical multiprocessors, or massive-parallel architectures – but his supercomputers were always the fastest systems available at the time. The only problem with the above strategy is that existing solutions need to exist and be available. There was no existing solution for Conrad Zuse's problem of repetitive, boring civil engineering calculations and so he unintentionally invented the first computer to be used for scientific calculation.

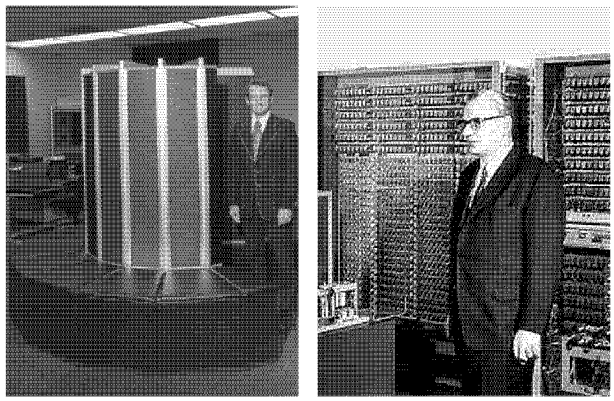


Figure 6.1: Supercomputer architect Seymour Cray (1925–1996) who refused to be a pioneer (left), and civil engineer Conrad Zuse (1910–1995) who invented the modern computer

Pioneering is exploring undiscovered areas: The goal is known, but no roads lead to this goal as yet. The pioneer must select a path without knowing whether it leads to the goal, if it is dangerous, or if there could be more efficient paths. The joy of being a pioneer is the freedom of choice, being first, earning fame. Everybody knows who discovered America, who invented the light bulb, who was the first man on the moon – but few know who was the second, the «optimizer», the successor.

Due to the dissertation project setup, this dissertation presents two pioneering works:

- It describes how to build comprehensive management software for superclusters.
- It presents the first management software that tightly integrates applications.

This chapter not only presents the achievements, the contributions to the supercomputing community, the outlook into the future and some personal experiences, but it should also transmit the author's sense of pride and fun of pioneering from himself to the reader.

6.1 Achievements

The basic achievements of this thesis are the research in supercluster management, presented in chapters 1 to 3, and the creation of COSMOS, presented in chapter 5. Both include important achievements, which are simultaneously contributions to the community.

- A pile of hardware does not make a supercomputer – some «magic» is required to make it behave as one entity. This «magic» is management software as presented in this dissertation, and this dissertation proves that **system management software is mandatory**. The proofs are presented in the sections 1.6, 2.1, and 2.4.
- The thesis is the **first complete research** about comprehensive and scalable computer-aided management of large commodity parts-based supercomputers. No other document is known to the author where this issue of supercomputing has been presented in this amount of detail, and illustrating that management is more than just a «tool» that is nice to have.
- For all those readers who want to create management software for superclusters, chapter 3 contains **blueprints of architectures** with the following characteristics:
  - The architectures are **hierarchical**, with managers (**«centers»**) at the top and managed components (running **«agents»**) at the bottom.
  - The architectures use scalability-enabling devices (**«proxies»**) if the system size or administrative requirements exceed limits. These proxies can be **cascaded** in several layers, and they can also serve as **data aggregation** devices for monitoring data.
  - The architecture can be made **highly available** with master/backup or reliable cluster mechanisms, which are implemented in the centers and proxies.
  - The parallel **application is made manageable** with a library that attaches the application closely to the management.
  - The **user interfaces connect to the center and** – if available – to those **proxies** that can provide the on-line monitoring data that the user wants.
  - The management software is **modular** and allows **all subsystems to be integrated**.
- The supercluster management software **«COSMOS»** which the author created for the Swiss-T1 supercomputer, has the following characteristics:
  - Although COSMOS was developed in 2000/2001, it is **still in use today**, because it is easy to use and maintain, and additional functionality is easy to implement.
  - COSMOS has **only crashed twice during its 3½ years of operation**, once because of a programming error, and once because of an unscheduled OS upgrade. Indeed, COSMOS has been proven to be the most reliable and stable management application available on Swiss-T1.
  - COSMOS is **very efficient**: The resource consumption of the center was low (1 CPU hour per week), and that of the node agent was also very low (about 5 CPU seconds per week).

- The tight integration of the application management allowed for previously unavailable features. Effective **system management** therefore **requires application management**.
- COSMOS was created by the author within **15 months**, including SAN agent and GUI that were added to the finished design within 1 month of being requested.
- The intuitively **selected architecture and design** (center, agents, library, GUI and CLI) **proved to be correct** according to the research presented previously.
- **Only commodity software technologies** were used, except for the GUI that uses qt.
- The topology and history tabs of the GUI include **innovative presentation and utilization**, which allows problems in the supercluster to be seen quickly. Indeed, the GUI was admired by both the administrators and Prof. Gunzinger.

The author is proud to announce that all presented achievements were executed by himself, although influenced by his examining professors, team members, research of other software developers (such as SNMP or supercomputer management software), or other scientific areas (such as human resources, psychology, physics, and arts).

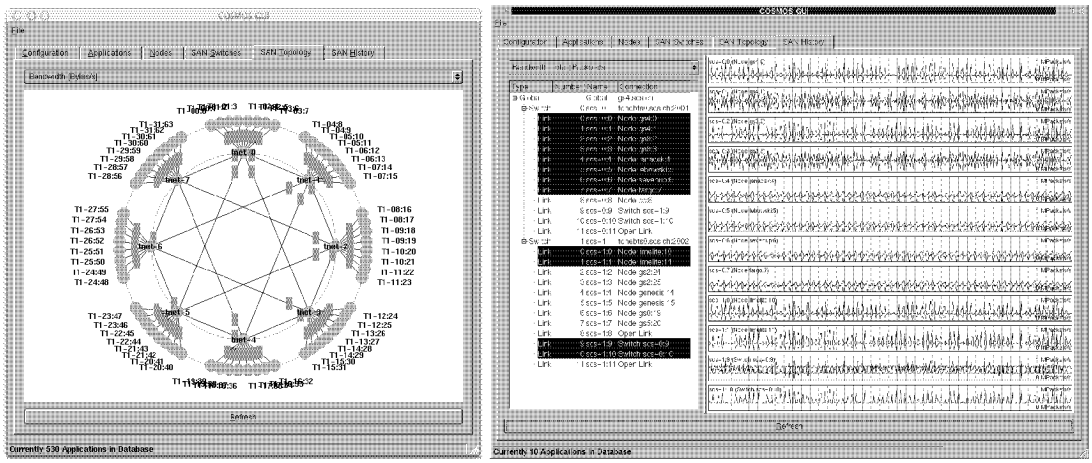


Figure 6.2: The COSMOS GUI with the two favorite monitoring data presentation tabs: The topology tab with all nodes, switches and links with its status and performance (left) and the history tab where the links to be shown can be selected in the left pane (right)

Although COSMOS was previously considered to be a prototype for second-generation management software, it was a proven success and used by many confident administrators and users.

Indeed, all software and hardware that was designed and developed by ETHZ and SCS for the Swiss-Tx supercomputer project was very successful. This fact shows that Switzerland would be a major player in the supercomputing community if the «critical mass» were to be exceeded, if Swiss-based companies could deliver products and technologies all over the world, if commodity supercomputing would really become a «commodity» and the major players in computing (such as SUN, SGI, HP or IBM) would not try to suppress technologies and products of organizations that are not affiliated with their companies.

6.2 Inventions and Contributions

If software solves technical problems and has technical or physical effects, it can be protected with patents, together with the methods that lead to the creation of this software. Integrated management software for superclusters complies with these conditions, and for this reason it can be patented, together with the user interfaces that control the supercluster.

The author believes that the following four inventions (all of which have been presented in this dissertation) would be worth protecting with patents:

- This dissertation presents a method showing how effective and efficient management software for supercomputers that contain mostly commodity off-the shelf components could be created. This method includes the following two discoveries:
  - The «Supercluster Lifecycle» (presented in 2.2) with its three phases «Design», «Installation» and «Operation» shows that there must be at least three software products (each serving one phase) with a common MIB for comprehensive management software.
  - All management functionality has a hierarchy as presented in the «Functionality Pyramid» in 2.3.2, which is inspired by Maslow's pyramid of self-actualization. If functionality of a certain layer is required, all functionality of the layers beneath must be implemented.
- The scalability-enhancing proxy as presented in 3.3.5 is an innovation, since it serves not only as a communication collector/distributor for the center and agents, but also as a (monitoring) data aggregating entity, delivering statistical information to the centers and all information on-demand to UI applications.
- The inverted connection strategy (the agents connect with the center/proxy instead of vice versa) reduces communication overhead in the center/proxy and closes an important security hole: Malicious software acting as managers cannot connect to the agent and create problems that are hard to detect by the center or proxy.
- The integration of the application management into the system management makes the supercomputer management complete. This integration of software into mainly hardware management is new and allows many new features to be accessed by the users and administrators.

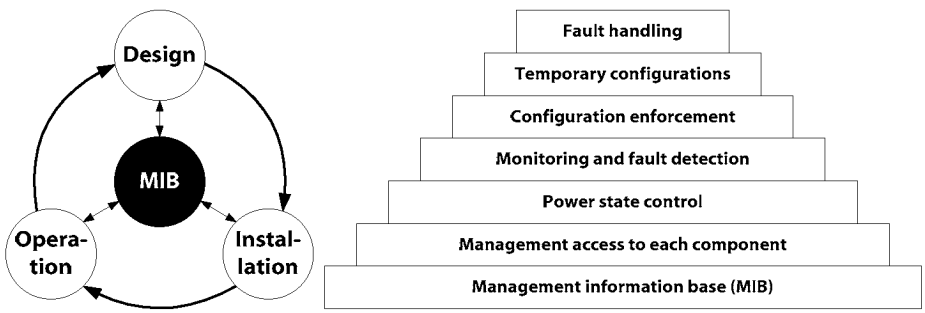


Figure 6.3: Two methods that lead to effective and comprehensive supercluster management: The lifecycle with its three phases (design, installation and operation, left) and the management functionality pyramid with the hierarchy of the provided functionality (right)

As soon as the information about inventions is published, the possibility of protecting them with patents disappeared. Since this dissertation has been published prior to any patenting efforts – and the author is not going to perform any – they have become contributions of the author to the supercomputing community.

The author hopes that these contributions will lead to many management software implementations being run on superclusters worldwide with civil applications.

6.3 Looking to the Future

No crystal ball is necessary to foresee the future of supercomputing:

- More supercomputers containing commodity parts only will become available.
- The system size will increase towards millions of nodes.
- The system performance will reach some PFLOPS in a few years time and some EFLOPS a decade later, with an MTBI of some minutes or seconds.
- There will be commercial providers of supercomputing resources to the public, and users from all over the world will be able to submit their jobs via the internet («GRID»).

With the experience gained through research and COSMOS presented in this dissertation, it can clearly be seen that management is the key: Without system management software, the supercluster is only an inefficient and ineffective pile of hardware, whereas the addition of fully integrated management software gives the supercluster the self-consciousness that it needs.

This required self-consciousness will not come from itself, but will require two efforts:

- The first effort is that of creating the management software itself. This effort can be performed by research teams at universities, open-source programming teams, or developers at supercomputer integrators or manufacturers. This dissertation should give those teams the knowledge of how to create this software.
- The second effort is that of creating hardware and software that can be smoothly integrated into superclusters. This must be done by the component manufacturers, by adding features that enable comprehensive management of the component.

This section therefore contains two parts: The first part shows how COSMOS (or its successors) could be turned into comprehensive system management software, and the second part gives some hints how commodity hardware and software could be enhanced for manageability.

6.3.1 Improving COSMOS

COSMOS is a working prototype – nothing less, but also nothing more. Fredrick P. Brooks once stated «plan to throw one away – you will, anyway» [Bro75]. Although much work and pain was invested in the development of COSMOS, it should be «thrown away», since the gained experience will allow a successor system to be built.

What would the author do today were he asked to create supercluster management software – or what would he suggest to the developers if he were asked for his opinion? The following list summarizes the architecture, design and implementation decisions he would take or suggest.

- **State-of-the art supercluster management software must have the integrated architecture as presented in 3.3.3.** The Swiss-T1 experience shows that currently available management software for individual subsystems is unreliable, hard to use, and does not scale. Non-integrated or glueware architectures will therefore fail.
- **The design must include high-availability mechanisms for managers as presented in 3.3.4.** The supercluster owner loses money if the system management service is unavailable. For small systems, the master/backup mechanism is sufficient and simple to implement. Larger systems require the «reliable cluster» mechanism that balances the workload over all members, which is harder to implement, but scales better and is more reliable.
- **The architecture must include proxies as presented in 3.3.5 if the system size exceeds 500 nodes per center or high-speed monitoring is required.** More nodes per center increases communication overhead and response time, and more than 4 000 nodes per center is not possible with current UNIX implementations without recompiling the kernel.
- **The product must include software for design and installation of the supercluster.** Creating basic configuration data manually is not only inconvenient, but also often a source for inconsistency errors that are hard to find and to solve. One central MIB, which is fed by design software, and is the source for basic configuration data of all components, reduces the setup and installation time by at least one order of magnitude.
- **Integrate the most important subsystems only, but integrate them completely.** The most important subsystems are nodes (including the inserted SAN and LAN NICs, storage, and OS), SAN (switches), resource («batch management»), and application management. The other subsystems (LAN, storage) are often only manageable with proprietary applications of the manufacturers – often protecting business secrets or security mechanisms.
- **Use standardized and open-source products for implementation.** Modularity and communication should be implemented using an object-orientated component library (DCOM or CORBA), visualization should use widget libraries (qt or others), reporting should be implemented using a reporting tool (such as CUBE or Crystal Reports). There are many standardized products and technologies that can be integrated into a professional product. The complexity of integrating these technologies is more than compensated for by the time gained if the functionality need not be implemented by the developers.
- **Create a structured scripting language for describing management actions.** Administrators and users are used to having scripts for various tasks. Fault and trap handling must not be implemented using hard-coded actions, but in a way that allows the standard management actions to be adapted. This is only possible if a script language for describing those actions is available, and the management software includes an interpreter for these scripts.

- **The MIB must be implemented as a SQL-based database.** This has the administrative advantage of the MIB to also be made available offline, when the COSMOS center is down or has crashed, or after the supercluster has been removed. The statistical long-term data would have been of particular interest to the administrators, users – and the Swiss-Tx project managers. The user interfaces are able to access the MIB directly without consuming processing time and/or communication bandwidth to the center.
- **Create GUI and shell commands as user interfaces to the management.** Since human beings are visually driven, the supercluster should be managed graphically and a fancy GUI must be available from the first day. As UNIX users are used to creating and applying shell scripts, the user interface should also include many shell commands, which would also be useful if only slow connections were available. The CLI as for COSMOS is not necessary.

The biggest problem of designing management software for supercomputers is the environment for testing. It should be big enough that scalability and concurrency problems can occur, leading to a test supercluster of at least 64 nodes. COSMOS was tested on a small system with four nodes, which was concurrently being used by the development teams of T-Net and FCI/MPI. In-depth testing was only possible after the installation of the Swiss-T1 supercluster at EPFL, which lead to insomnia for the whole development team between installation and inauguration day. The development team must insist on having exclusive access to a homogenous testing environment on-site.

6.3.2 Work for «the Others»

Management software can only manage what is made manageable. All the hopes that are put into management software concerning increasing MTBI, scalability and functionality, as well as decreasing vulnerability and complexity, depend on the capabilities of the hardware and software that is used in the supercluster. This dissertation assumed that these components allow full manageability, but the reality is different. Many obstacles prevent the fulfillment of these hopes, and it is not in the hands of the management software developers to remove these obstacles.

The manufacturers of the «commodity off-the-shelf components» also have some work ahead of them, to turn commodity supercomputing into a full technological and economical success. The following is an incomplete «wish list» aimed at those manufacturers.

- **Make the components easily and completely manageable.** Do not hide any information, make the component-internal MIB completely accessible to the management agents, allow every single feature and control action to be accessed by the agent. The following list shows some examples where such easy and complete management is not yet possible.
  - If the memory subsystem corrects errors by itself, the error rate must be made available, so that the administrators can replace memory modules during scheduled downtimes if they make too many «non-fatal errors», before the node causes a costly application abortion or the proxy/center causes an expensive supercluster downtime. Fatal faults are almost always preceded by abnormally high «soft error» rates.

- Many computers cannot control their own power status. Low-cost computers cannot even be switched on or off remotely. There must be a permanently available computer-internal management mechanism that allows at least secure power management of the computer, e.g. in the LAN NIC.
- The management software of many hardware parts often accepts telnet connections only, with text-based command line interfaces (such as most UNIX-based servers for power and boot control). These interfaces cause complex codes and much overhead in the system management software. There must be a socket-based message protocol that allows management messages to be sent and received without complicated interpreters.

❖ **Design the hardware for easy maintenance.** Especially hardware for 19" racks must be designed for easy utilization, access and replacement without tools.

- Replacing parts should be as simple as replacing the light bulb in a lamp, nothing more than pulling out a drawer, opening a hood with one finger, grabbing the defective part, putting in the new part, closing the hood and pushing the drawer back into the rack.
- All connectors must be on the same side: On the back. Terminal servers usually have all connectors on the front, whereas the nodes have all connectors on the back. Connecting console cables to nodes is a ridiculous task if many racks are standing side-by-side and the installer has to walk many meters around those racks to switch sides.

❖ **Design operating systems and all software for easy maintenance.** It should be possible to create a complete software image that can be downloaded to the node, without the need for complex configuration adaptation tasks for each individual node.

- Installing an OS upgrade on a supercluster should not include the modification of many configuration files on each node. Download an image, unpack the image on the shadow disk of the node, reboot the node from the shadow, check the operation, duplicate the image from the shadow to the productive disk and reboot from the productive disk. That should be all that needs to be done.
- Installing new software on the supercluster must be non-interactive and non-destructive, must not require configuration file adaptations, and must be easy to de-install. Since software installations happen more often than OS upgrades, they must be faster and easier to perform than the OS installation described before.

❖ **Apply established product and management standards.** Although innovations hardly correspond to existing standards and bright ideas cannot be put into standardized frames, it must be possible to manage everything with standardized management software. Innovative technology cannot be used in superclusters if they cannot be managed by the system management software. As mentioned above, glueware or non-integrated management software architectures are useless, which disqualifies products with non-standard management architectures.

6.4 Personal Experiences

If supercomputing allowed products and technologies for time travel to be created, thus breaking physical laws, and enabling mistakes in global and personal history to be undone, the author would travel back in time and explain the following to his past self:

Tell the Swiss-Tx project managers the importance of system management. They were wrong to expect system management software to be a small tool to manage highly sophisticated and autonomous hardware. Since integrated management for superclusters was new, they could not know any better. The development time for T-Net and FCI would have been much shorter if COSMOS had been developed as an equal partner in parallel with a larger team of 2 to 3 people. Global management features would have been solved in COSMOS instead of in the hardware, which would have led to more features and reliability in less time.

Time invested in specification and design is not lost, but re-couped 10-fold. Every minute saved in the design process is paid for with one additional hour of implementation and testing time. Although there was a need for working prototypes and not enough time for the design process, many weekends and nights of work could have been spent on other tasks. On the other hand, many of the most creative and unorthodox solutions were the result of this pressure.

There is no such thing as a single solution, and if there is thought to be, it will turn out to be the wrong solution. Heading in a direction where multiple solutions are expected is a safe bet, since one of them will certainly work. Immediately implementing the only expected solution is dangerous, since it will surely be found to be wrong at a later stage, such as during implementation or testing. A spiral path around possible solutions, evaluating them and selecting the most promising one, is less efficient but more effective, whereas a direct selection of the «only solution» always turns out to be more efficient but less effective.

Use available commodity products that simplify design and implementation. It is OK to use standard products in an academic project in order to shorten development time. It may take more time to integrate these products into the design than creating an own solution, but the gain is made by using established and working modules. This is especially true for databases, transaction-like communication and thread-safe data handling in queues and lists. Using qt for the GUI allowed its development time to be massively reduced.

Implement a graphical user interface from day one. Since people are visually driven, graphical user interfaces are very important. The confidence in COSMOS – and the author – massively increased after the GUI was finished. Command-line based user interfaces are nice, but a colorful window with flashing lines, boxes and circles, with performance meters and many other bells and whistles is very important.

Linux and Tru64 UNIX are not completely compatible. Many hours were lost because fundamental data types and libraries look the same, but are not the same. Socket communication and pthreads multithreading in particular are differently implemented.

There are additional challenges in Linux: Some functionality has not been completely implemented (such as timeout and watermark values for sockets), and some other functionality has been simplified so much that it cannot be used as intended (such as sending signals between threads).

Experience comes from making mistakes, analyzing them and learning how to do it better next time. Since pioneers are allowed to make mistakes – and the goal of research is to learn how to do something, how to make things better – the project created a wealth of experience that is printed in this dissertation. And the author hopes that this document will help developers to create state-of-the-art comprehensive system management software for superclusters.

When management software gains control over hardware, when the components react immediately at the fingertips of the GUI user, when flashing boxes in windows show the performance of a supercluster, when plugging in cables or switching off the power causes automated physical reactions, and when the supercluster with many independent parts gains self-consciousness through using the management software, the same amount of fun is provided as when watching a child growing up. This fully compensates for all the pain and effort that was needed in the preceding process.

A Performance Simulation

The supercluster lifecycle presented in section 2.2 includes the simulation of the design phase, where the potential performance of supercluster designs is calculated. This appendix presents the two main estimation techniques, Amdahl's law and the calculation-to-bandwidth ratio γ.

A.1 Extended Amdahl's Law

Amdahl's law [Amd67] describes how application execution time is reduced and calculation performance is increased («speed-up») by using accelerating technologies. Amdahl's law was introduced when vector supercomputers were established and the calculation on vector-type data (vectors or matrices) could be accelerated using vector pipelines, registers and data units.

Amdahl's law divides the application code into the serial code that cannot be accelerated, and the vector code that can be accelerated using the accelerating technology. For parallel supercomputers, the vector code is called parallel code that is divided between the nodes – if an application has a loop that operates on 10 000 data items, the loop will operate on 100 data items on 100 nodes.

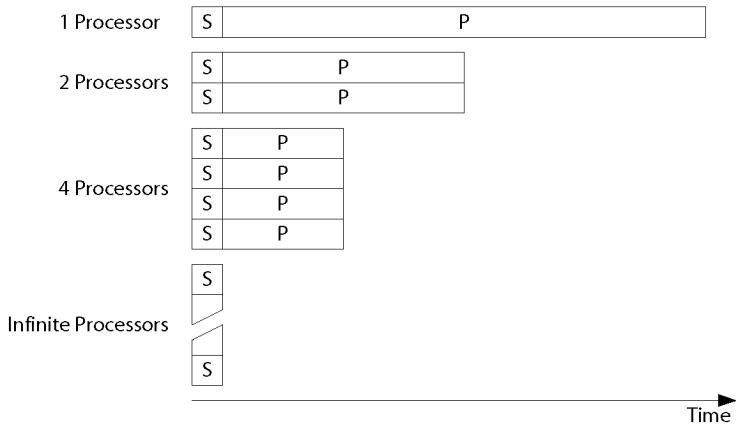


Figure A.1: Amdahl's Law with serial and parallel code only

This serial code limits the maximum speed-up because it builds a socket of instructions that cannot be parallelized.

Amdahl's law remains correct for vector processors, because either there is no communication overhead or it is part of the serial code. For parallel computers, Amdahl's law must be corrected because the communication overhead usually increases when using more nodes.

Figure A.2: Amdahl's law with serial, parallel and communication code

The above illustration shows that the communication overhead compromises the gain of parallel execution (same performance of 4 and 8 processors). The following figure shows the speed-up behavior for various distributions of the serial, parallel and communication codes of an application.

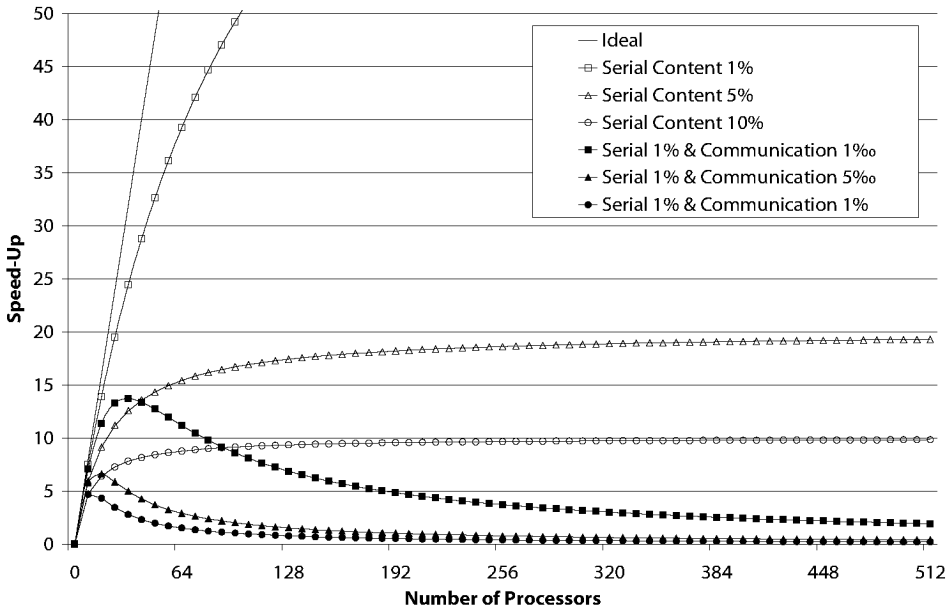


Figure A.3: Speed-up over size with variable scalar, parallel and communication parts

The above figure demonstrates dramatically, that the application performance and speed-up is limited by the application itself and by the architectures of the used node and network technologies:

- The parallel application defines the shape of the speed-up curve and, depending on the algorithm used, the communication patterns and application design. A good algorithm has a low serial content which moves the maximum speed-up upwards.
- The node architecture defines the performance measured in FLOPS. Multiplied by the expected speed-up, the performance can be estimated for a specific number of processors.
- The SAN architecture defines how much the communication code affects the speed-up. A smart SAN architecture reduces the overhead and moves the speed-up maximum toward a higher node number, which allows a higher speed-up.

The speed-up can be calculated using the following equation, which also allows the number of processors where the maximum speed-up is reached to be estimated.

$$S(p) = \frac{1}{c_0 + \frac{c_1}{p} + c_2 f(p)}$$

Equation A.1: Speed-up formula for parallel applications

- The value p is the number of processors executing the application.
- The value c_0 ($0 \le c_0 \le 1$) is the part of code that cannot be parallelized and must be executed on each node separately, such as the application start-up code.
- The value c_1 ($0 \le c_1 \le 1$) is the part of code that can be parallelized and that takes (theoretically) no time even if executed on infinite nodes.
- The value c_2 ($0 \le c_2 \le 1$, $c_1 + c_2 + c_3 = 1$) is the part of code that contains the instructions for the data transport between nodes and synchronization.
- The formula f(p) describes the communication behavior of the application depending on the number of participating nodes – it can be fixed (if neighbor-to-neighbor communication is used), linear (if global broadcasts are often used) or have any other form.

This speed-up equation (or, to be more precise, its first derivative to the number of processors) is used to estimate the optimal number of nodes for the application, where the maximum speed-up is achieved. To simplify the calculation, the communication behavior is assumed to be linear, described by f(p) = p.

$$\hat{p} = \sqrt{\frac{c_1}{c_2}}$$

Equation A.2: Optimal number of processors for parallel applications

If 99% of the code can be accelerated and 1‰ of the code is executed for communication with each partner, the optimal number of processors is about 32. Reducing c_2 to 0.1‰ leads to an optimal processor number of about 100. Increasing the number of processors beyond this value will decrease the speed-up.

A.2 Communication-to-Calculation Ratio γ

Supercomputers in general (and superclusters in particular) are often limited in bandwidth; between the network and the memory, between the storage and the memory, between the memory and the CPU and between the subsystems of the CPU (registers, cache, functional units). The bandwidth is the most limiting factor in supercomputers, since it has not increased to the same scale as the potential processing performance. The data cannot be loaded or stored fast enough to efficiently fill the pipelines of the CPUs, or the data cannot be moved fast enough through the SAN or storage subsystem to keep the CPUs in the node busy.

The instructions of an application need to be able to load operators from memory for calculation and also store the results into memory. Some of the results are transferred to storage devices, and some are communicated over the network to other nodes. Depending on the application, the amount of data moved within the node and supercluster varies.

$$\gamma = \frac{Bandwidth}{Performance} \frac{[Bytes/s]}{[Instructions/s]}$$

Equation A.3: Communication-to-calculation ratio γ

The communication-to-calculation ratio γ (measured in Bytes per instruction) describes how much bandwidth is required – or available – per instruction, how much is needed by the application, and how much is offered by the system. There are various γ values, describing the bandwidth at various places in the supercluster.

- The value γ_A describes the need of the application, how many bytes of data are to be moved per floating point instruction from or to the cache, memory, storage, LAN and SAN). These values are given by the application architecture and design.
- The value γ_M describes what the supercluster can offer to the application (how many bytes of data the node can move between CPU and the cache, main memory, storage, LAN and SAN). The values are given not only by the node and SAN architectures, but also by the selected topology of the supercluster.

The following table shows the γ values that need to be calculated for each application and each product used in the supercluster designs. The ratio $\Gamma = \gamma_M \div \gamma_A$ shows what is available compared to what is needed on each level.

Interface	Machine γ_M	Application γ_A	Ratio $\Gamma = \gamma_M \div \gamma_A$
CPU / Cache	$\gamma_{M/C}$	$\gamma_{A/C}$	Γ_C
CPU / Memory	$\gamma_{M/M}$	$\gamma_{A/M}$	Γ_M
SAN / Memory	$\gamma_{M/N}$	$\gamma_{A/N}$	Γ_N
Storage / Memory	$\gamma_{M/S}$	$\gamma_{A/S}$	Γ_S
LAN / Memory	$\gamma_{M/L}$	$\gamma_{A/L}$	Γ_L

Table A.1: The γ values in supercluster and application

$$\Gamma = \frac{Available\ in\ System}{Needed\ by\ Application}$$

Equation A.4: Comparison value Γ showing how much is available of what is needed

After these γ values have been calculated for every application and supercluster design, the results show whether the calculation or the bandwidth is limiting the computation performance. The least relation $\Gamma = \gamma_M \div \gamma_A$ of the respective values for the supercluster and the application show the maximum performance that is achievable for the system.

The least value Γ is the limit for the performance, calculated as follows:

$$p = \hat{p} \times \min(1, \Gamma_C, \Gamma_M, \Gamma_N, \Gamma_S, \Gamma_L)$$

Equation A.5: Relation between performance, peak performance and Γ

The following example is based on the following supercluster and application algorithm:

The supercluster is an 8-node system of 8 COMPAQ AlphaServer DS20 nodes with 2 Alpha 21164 microprocessors (500 MHz, 1 GFLOPS, 4 GBytes/s cache bandwidth, 2 GBytes/s memory bandwidth) and a fully connected T-Net SAN with an average bandwidth of 100 MBytes/s per node.

The application algorithm is a matrix-vector multiplication that is commonly used for the iterative solution of linear equation systems. The vectors have a size of 1024 elements and the matrix has a size of 1024×1024 elements.

$$\mathbf{x}_{i+1} = \mathrm{A} \cdot \mathbf{x}_i + \mathbf{b}$$

Equation A.6: Iterative solution of linear equation systems

The supercluster-based γ values are as follows:

- The connection between CPU and cache is capable of transporting 4 GBytes/s and the CPU can process 1 GFLOPS, resulting in a value $\gamma_{M/C}$ of 4 Bytes/instruction.
- The connection between both CPUs and memory transports 2 GBytes/s and both CPUs process 2 GFLOPS, resulting in a value $\gamma_{M/M}$ of 1 Bytes/instruction.
- The SAN NIC can transport 100 MBytes/s while both CPUs that share the SAN NIC process 1 GFLOPS, leading to a value $\gamma_{M/N}$ of 0.05 Bytes/instruction.

The application-based γ values are as follows:

- The vector might match the cache size, but since external caches are usually direct-mapped and internal caches are usually too small, the vector trashes the matrix rows, making the data cache completely useless. The $\gamma_{A/C}$ value is therefore identical to the $\gamma_{A/M}$ value.

- For each resulting vector element, 1024 multiplications and 1024 additions are performed, needing 2048 operands read from memory. This results in a value $\gamma_{A/M}$ of 8 Bytes/instruction.
- The resulting vector elements must be globally communicated. Each of the 16 CPUs calculates 64 elements and sends them to the other 15 CPUs, leading to a value $\gamma_{A/N}$ of about 0.059 Bytes/instruction (64 elements × 8 Bytes/element × 15 repetitions= 7.5 kBytes, 64×2048 = 128k operations).

Interface	Machine γ_M		Application γ_A		Γ	
	Name	Value	Name	Value	Name	Value
CPU / Cache	$\gamma_{M/C}$	4.000	$\gamma_{A/C}$	0.500	Γ_C	8.000
CPU / Memory	$\gamma_{M/M}$	1.000	$\gamma_{A/M}$	0.125	Γ_M	8.000
SAN / Memory	$\gamma_{M/N}$	0.050	$\gamma_{A/N}$	0.847	Γ_N	0.059

Table A.2: The Γ values of the presented example

If the least Γ is 0.059 – only 6% of the required bandwidth between SAN NIC and memory can be delivered – the maximum computation performance is 6% of the peak performance. There is no way to get faster with this algorithm and the selected components of the supercluster. The peak performance of this system is 6% of the theoretical peak performance, in this case less than 1 GFLOPS of the 16 GFLOPS available. The rest is wasted because the SAN cannot keep the memory subsystem busy.

This model is very simplified and holds for applications, where the communication tasks are equally used and performed in parallel. If the application is highly optimized for execution on the supercluster, it works on as much data as fits in the cache, later accesses the memory only to fill the cache again, thus reducing the effect of bandwidth bottlenecks. The real performance formula could look as follows, but further research and work is required:

$$p = \hat{p} \times \sum_{i \in \{C,M,N,S,L\}} (w_i \times \Gamma_i)$$

Equation A.7: Corrected, weighted relation between performance, peak performance and Γ

In this formula, all Γ values are included with their relative weight.

B Sample MIB

This appendix describes a sample management information base (MIB) for supercluster management software and the COSMOS MIB itself at the end. The tables not only describe the data in the MIB, but also the source and cause of the data with codes. They have the following meaning:

- Configuration data (in column «C») defines the behavior of the component.
  - The basic configuration («B») is loaded after start-up or reset and is entered by the administrators. There may be multiple basic configurations depending on the desired supercluster mode, selected by the administrators or the scheduling service.
  - During operation, some configuration data is temporarily created («T») and later removed from the devices (e.g. routing tables in SAN switches). This data is also removed if the supercluster is restarted.
- Monitoring data (in column «M») shows current performance.
  - Some values are sampled slowly or only once after start-up («S»), since they do not change rapidly (e.g. temperature) or not at all (e.g. serial number).
  - Other values are sampled quickly («Q»), since they change quickly and are used to check the current usage and performance (e.g. link bandwidth).
  - Some values are stored as an event happens («E») (e.g. application start-up). These values are saved with their context (e.g. causing application or user).
- Monitored values can be protected through the trap mechanism (in column «T»). If a trap is triggered, the associated action handling this trap is executed.
  - Some values are protected using a pair of limits («L»). An action is triggered if the limits are broken. Some values may even have two pairs of limits. Possible values are node temperatures, where exceeding 40 °C may cause the air conditioning to be increased and exceeding 50 °C may cause an emergency system shutdown.
  - Some values are compared («C») with values of like components. The mechanism reacts if the difference between these values is too big. Possible values are the CPU load of nodes running processes of the same application. If a process consumes almost 100% of the CPU power and all others almost 0%, this may indicate a hanging process and the application must be aborted to save lost processing time.
- The fault mechanism (in column «F») detects faults and unexpected changes, and triggers the associated actions for handling the fault.

Since this appendix only covers a sample MIB for supercluster management, the list is not intended to be complete, because the length and coverage depend on the intentions, capabilities and requirements.

B.1 Structure of the MIB

The MIB data is organized as a tree of identifiers. Every identifier is a scalar value, an array, a structured data type or again a tree of identifiers. The structure of the MIB is similar to the structure of the SNMP MIB [Ros96]. Connected to the root identifier, the subsystems are the next layer of identifiers, followed by the types of components (if heterogeneous) followed by the instance number, followed by further identifiers.

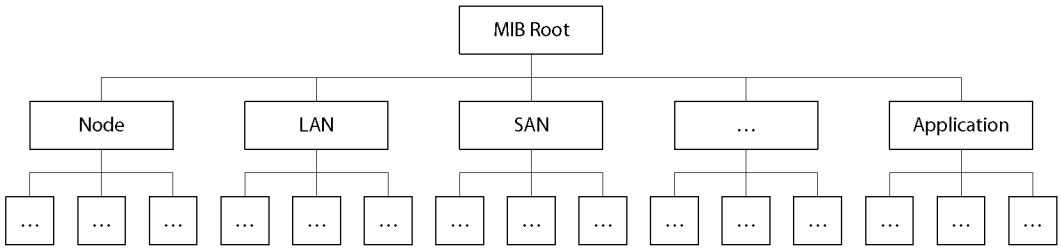


Figure B.1: The MIB structure – a tree of identifiers

Every value in the MIB is identified by a series of identifiers and numbers. The identifiers specify the value type of interest, and the numbers specify the respective entity. The following EBNF notation describes this identification.

```
ManagementModule { ( "." Entity ) | ( "." Identifier ) }
```

The next illustration shows an example of how to access the output rate in Bytes/s of link 4 of the SAN switch 2 of the SAN subsystem. The textual notation to identify this value for this example is `SAN.Switch.2.Link.4.SendRateOut`.

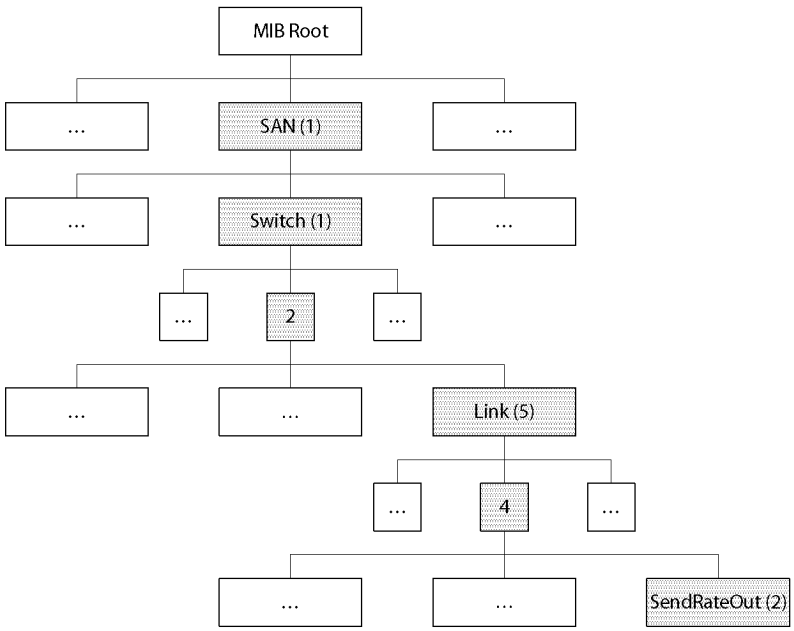


Figure B.2: MIB value identification example

Of course the text identifiers are backed by numerical values, so in the above example they can be accessed using the identifier `1.1.2.5.4.2` instead.

In the above example, the requested data is the current value. Monitored values are usually stored in an array, which is implemented as a separate table of the database, and contains all sampled values. To access a certain value, the access algorithm translates a request of a value of a certain moment into the right index of the value array. This means that the output rate of 20:14:55 hours from November 11th 2000 may be translated as the 162734th value in the table.

To simplify these accesses, there are two strategies:

- The administrators define a moment zero (e.g. January 1st 2000 at 00:00:00 hours) and a heartbeat (e.g. one heartbeat per 5 seconds). Values are then stored in the right index (e.g. January 1st 2002 at 00:00:05 hours in index 1, January 1st 2002 at 00:00:10 hours in index 2 etc.). The index is simple to calculate and the tables contain explicit values, but this strategy creates huge tables, and any changes to moment zero or heartbeat create problems.
- Every value contains not only the value itself, but also the sampling moment. To further compress the tables, no values are added if the change is too small compared to the value already stored. This strategy creates smaller tables, but index calculation of a defined moment is hard.

The information which is stored in the MIB and managed by the center application can be divided into the following three major groups:

- Static information such as descriptions or serial numbers that is entered manually by the administrators or by automated information retrieving mechanisms during installation.
- Configuration data which is loaded from configuration files and entered on-line by administrators through the user interfaces or – for agents and proxies – sent from the centers.
- Dynamic information (called monitored data) generated by the agents such as data rates, resource usage or temperature. This data is stored in a way that supports the value retrieval of various moments in history.

The following data of managed components is stored in the MIB:

- Information about the managed component, such as description, serial number, contact information (IP address, TCP port) or location in the building.
- Configuration data for the managed component, including limits for selected monitored values for the trap mechanism.
- Description of executed actions in case of traps and faults such as scripts.
- Sampled monitoring data from the managed components as provided by the agents such as temperature, data transfer rates or resource usage.
- Statistical data from the monitoring data for instant access without calculation, such as maximum, minimum and average values for each monitored value within a timeframe.

The application and resource management include the following data in the MIB:

- Information about executed applications including the processes.
- Information about applications waiting in queues scheduled for execution or already being executed on the supercluster nodes.
- Configuration of the application and resource management, including scripts executed before and after the application execution and actions executed in case of traps and faults.
- Configuration of the scheduling mechanism and job queues of the resource management.
- Sampled monitoring data from the processes provided by the application and node agents, as well as statistical data derived from the monitoring data such as minimum, maximum and average values – after a certain time, the summaries are stored and the sampled data can be deleted to save space and time.
- Accounting information to enable users, projects and departments to be charged at a later time.

Which values are stored in the MIB and how they are saved and represented depends on the design and implementation of the management system as well as on the selected platforms and products used in the supercluster.

B.2 Node Management

The node management stores the following information for each node in the MIB:

Name	Value Description	Unit	Example	C	M	T	F
Description	Description of the node		COMPAQ DS-20e				
SerialNumber	Serial number of the node		SN0815-42				
HostName	Host name (alias) of the node		cluster-01-001	B			
Room	Room in supercomputing center		AGD 23.0				
Rack	Rack of the node in the room		D3				
Position	Position of node within rack		0				
OsName	Name of the node OS		OSF1		S	C	
OsVersion	Version of the node OS		V5.1a		S	C	
Status	Power status of the node		Ready		E		×
Temperature	Max. temperature of all sensors	°C	21		S	L	
RamSize	Main memory size of the node	MBytes	1024		E	C	
MemorySize	Currently used virtual memory	MBytes	2432		S	C	
CPU	Tree describing the CPUs						
Storage	Tree describing the storage						
LanNIC	Tree describing the LAN NICs						
SanNIC	Tree describing the SAN NICs						
Slot	Tree describing the slots						

Table B.1: The MIB node values

Every CPU of a node has its own record containing the following fields.

Name	Value Description	Unit	Example	C	M	T	F
Description	Description of the CPU		Alpha EV67		S	C	
Revision	Revision code of the CPU		2		S	C	
Clock	Clock frequency of the CPU	MHz	667		S	C	
Status	Current status of the CPU		Ready		E		×
Load	Current load of the CPU	%	75		Q	C	
AppLoad	Current application load	%	12		Q	C	

Table B.2: The MIB node values for the CPU tree

Every storage device of a node has its own record which contains the following fields.

Name	Value Description	Unit	Example	C	M	T	F
Size	Size of the storage unit	MBytes	32123		S		
Used	Currently used size	MBytes	12123		S	C	

Table B.3: The MIB node values for the storage description tree

A slot of a node is a resource unit that can be consumed by application processes. Every slot of a node has its own record which contains the following fields.

Value Name	Value Description	Unit	Example	C	M	T	F
SanID	SAN ID of the associated NIC		1	B			
ChannelID	Associated SAN NIC channel		0	T			
DeviceID	Associated SAN NIC device		0	T			
AppID	Application ID of the process		42	T	S	C	
Rank	Application rank of the process		0		S		
Status	Status of the slot		Started		Q	C	×

Table B.4: The MIB node values for the slot description tree

The field *Status* defines the current slot status. The slot status can be either *Unused* if the slot is currently free, *Reserved* if an application is about to start, *Started* if the application has started and the process is running, *Finalizing* if the process has finished its job and is about to end, or *Aborting* if the process or the application has detected a problem and is about to be terminated.

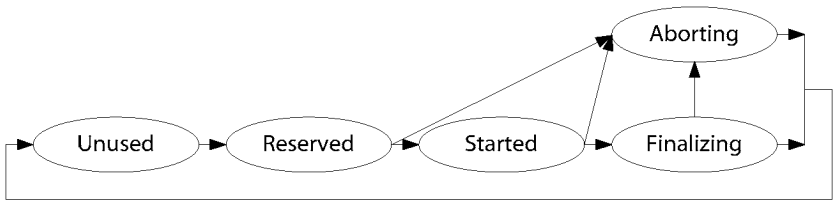


Figure B.3: Slot states and transitions

B.3 LAN Management Data

The LAN management stores the following information for each LAN switch in the MIB:

Value Name	Value Description	Unit	Example	C	M	T	F
Description	Description of the switch		Giga-Net Switch				
SerialNumber	Serial number of the switch		123-2365-42				
SwitchName	Switch name (alias) of the switch		cluster-01-l1	B			
HostName	Host name of the switch		cluster-01-l1.t1.ch	B			
HostPort	TCP port to connect the switch		1059	B			
Rack	Rack of the switch in the room		E2				
Position	Position of switch within rack		3				
Version	Version of the switch firmware		V2.02		S	C	
Status	Power status of the switch		Ready		E		×
Temperature	Max. temperature of all sensors	°C	21		S	L	
Port	Tree describing the port						

Table B.5: The MIB LAN switch values

Every port of a LAN switch has its own record which contains the following fields.

Value Name	Value Description	Unit	Example	C	M	T	F
Status	Status of the port		Up		E		×
SndRateB	Send rate of the port	Bytes/s	98123432		Q	C	
RcvRateB	Receive rate of the port	Bytes/s	12765987		Q	C	
Type	Type of the connection partner		Switch	B	S	C	
Partner	Number of the partner		1	B	S	C	
Port	Number of the partner port		1	B	S	C	

Table B.6: The MIB LAN switch port values

The fields *Type*, *Partner* and *Port* describe which location this port is connected to.

Every LAN NIC of a node has its own record which contains the following fields.

Name	Value Description	Unit	Example	C	M	T	F
Description	Description of the LAN NIC		COMPAQ DE500		S	C	
LanID	Tree describing the LAN NIC IDs						

Table B.7: The MIB node values for the LAN NIC description tree

Every *LanID* of a LAN NIC of a node has its own record which contains the following fields.

Value Name	Value Description	Unit	Example	C	M	T	F
IpNumber	IP Number of the LAN NIC ID		192.168.1.1	B			
HostName	Associated host name		cluster-01-01.t1.ch	B			
Type	Type of the connection partner		Switch	B	S	C	
Partner	Number of the partner		1	B	S	C	
Port	Number of the partner port		1	B	S	C	

Table B.8: The MIB node values for the LAN NIC ID description tree

The fields *Type*, *Partner* and *Port* describe which location this port is connected to.

B.4 SAN Management Data

The SAN management stores the following information for each SAN switch in the MIB:

Value Name	Value Description	Unit	Example	C	M	T	F
Description	Description of the switch		SCS T-Net Switch				
SerialNumber	Serial number of the switch		202-012-010-0001				
SwitchName	Switch name (alias) of the switch		cluster-01-s01	B			
SwitchNumber	Number of the switch		1	B			
HostName	Host name of the switch		cluster-01-s1.t1.ch	B			
HostPort	TCP port to connect the switch		2001	B			
Rack	Rack of the switch in the room		D4				
Position	Position of switch within rack		0				
Version	Version of the switch firmware		V23.12.1		S	C	
Status	Power status of the switch		Ready		E		×
Temperature	Max. temperature of all sensors	°C	21		S	L	
Port	Tree describing the port						

Table B.9: The MIB SAN switch values

Every port of a SAN switch has its own record which contains the following fields.

Value Name	Value Description	Unit	Example	C	M	T	F
Status	Status of the port		Up		E		×
FatalErrors	Number of fatal errors		0		E		×
SndRateB	Send rate of the port	Bytes/s	98123432		Q	C	
RcvRateB	Receive rate of the port	Bytes/s	12765987		Q	C	
ErrRate	Error rate of the port	Errors/s	12		Q	L	
SndRateP	Send rate of the port	Pkt./s	987234		Q	C	
RcvRateP	Receive rate of the port	Pkt./s	123765		Q	C	
Type	Type of the connection partner		Switch	B	S	C	
Partner	Number of the partner		1	B	S	C	
Port	Number of the partner port		1	B	S	C	

Table B.10: The MIB SAN switch port values

The fields *Type*, *Partner* and *Port* describe which location this port is connected to.

Every SAN NIC of a node has its own record which contains the following fields.

Value Name	Value Description	Unit	Example	C	M	T	F
Description	Description of the SAN NIC		SCS T-Net 32/33		S	C	
SerialNumber	Serial number of the SAN NIC		202-011-010-0042		S		
Version	Version of the SAN NIC firmware		V5.42		S	C	
BoardID	Unique ID of the SAN NIC		1	B			
Status	Power status of the SAN NIC		Ready		E		×
Port	Tree describing the ports						

Table B.11: The MIB node values for the SAN NIC description tree

Every port of a SAN NIC of a node has its own record which contains the following fields.

Value Name	Value Description	Unit	Example	C	M	T	F
Status	Status of the link		Up		E		×
FatalErrors	Number of fatal errors		0		E		×
SndRateB	Send rate of the link	Bytes/s	98123432		Q	C	
RcvRateB	Receive rate of the link	Bytes/s	12765987		Q	C	
ErrRate	Error rate of the link	Errors/s	12		Q	L	
SndRateP	Send rate of the link	Pkt./s	987234		Q	C	
RcvRateP	Receive rate of the link	Pkt./s	123765		Q	C	
Type	Type of the connection partner		Switch	B	S	C	
Partner	Number of the partner		1	B	S	C	
Port	Number of the partner port		1	B	S	C	

Table B.12: The MIB node values for the SAN NIC link description tree

The fields *Type*, *Partner* and *Port* describe which location this port is connected to.

B.5 Application Management Data

The application management stores the following data for each application in the MIB:

Value Name	Value Description	Unit	Example	C	M	T	F
AppID	Cluster-unique application ID		42 (0/42)				
AppName	Application name (command)		a.out				
AppPath	Application path		/users/nemecek/				
UserID	User ID		10083				
UserName	User name		nemecek				
BarrierCount	Number of barriers		2				
TimeCreated	Time stamp of creation		01-04-01 18:00:01		E	C	
TimeStarted	Time stamp of application start		01-04-01 18:00:10		E	C	
TimeLBarrier	Time stamp of last barrier		01-04-01 19:05:19		E	C	
TimeFinished	Time stamp of termination		01-04-01 19:05:20		E	C	
Status	Status of the application		Finalized		E		×
Processes	Tree describing the processes						

Table B.13: The MIB application values

The field *Status* contains the current application status. After the application record has been created, its status is *Uninitialized*. When all record entries have been initialized, the status changes to *Registered* and waits until all processes connect to the node agents. Once all the processes are ready, the status changes to *Started*. After the first process has sent its finalize message, the status changes to *Finalizing* and later to *Finalized* if all processes have finished. Troubled applications are aborted and their status is changed to *Aborting* and later *Aborted* if all processes have stopped.

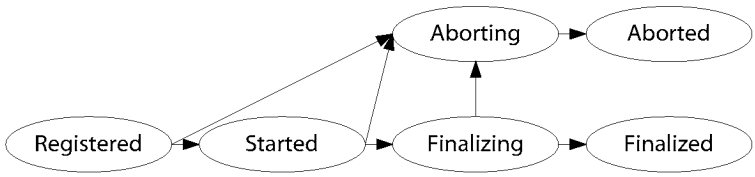


Figure B.4: Application states and transitions

The field *Processes* contains an array which contains information about all the processes belonging to this application. These process records contain the following fields.

Value Name	Value Description	Unit	Example	C	M	T	F
HostAlias	Node the process runs on		cluster-01-001				
HostSlot	Slot the process runs in		0				
CpuLoad	CPU Load with this process	%	12		Q	C	
MemSize	Memory usage of the process	MBytes	123		Q	C	
ProcessID	Process ID on the node		135975		E		
TimeRegistered	Time stamp of registration		01-04-01 18:00:02		E		
TimeLBarrier	Time stamp of last barrier		01-04-01 19:05:18		E		
TimeFinished	Time stamp of termination		01-04-01 19:05:19		E		
Status	Status of the process		Finalized		E		×
AbortCause	Cause of abortion (if aborted)				E		

Table B.14: The MIB application process values

The field *Status* contains the current process status. After creation of the process, the status is *Uninitialized* and it waits for the process to connect. After the process has connected and has sent the start-up message, its status changes to *Registered*. Once all the processes are ready, the status changes to *Started*. When the process has finished its operation, its status changes to *Finalizing* and after all application processes have finished, it changes to *Finalized*. If the process detects a problem, its status changes to *Aborting*, and after it has disconnected from the node agent, it changes to *Aborted*.

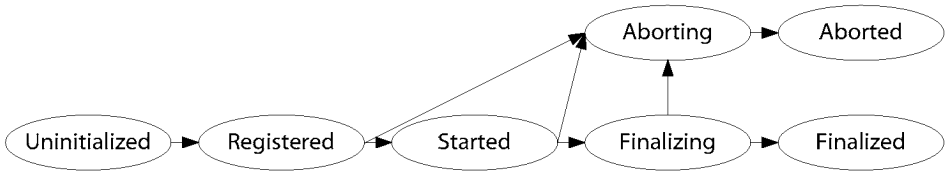


Figure B.5: Process states and transitions

If the application has been aborted, the field *AbortCause* will contain a description of the reason why the process was terminated. The process that caused the abortion will contain its reason; all other processes are aborted by the node agents. The following table summarizes the possible values of this field.

Abort Cause	Description
AbortCalled	The process detected a problem and called the abortion routine
Crashed	The process has disconnected from the node agent (crashed)
Signal_HUP	The process received the signal HUP and terminated
Signal_INT	The process received the signal INT (Ctrl-C) and terminated
Signal_QUIT	The process received the signal QUIT and terminated
Signal_ILL	The process received the signal ILL (illegal instruction) and terminated
Signal_TRAP	The process received the signal TRAP and terminated
Signal_ABORT	The process received the signal ABRT and terminated
Signal_FPE	The process received the signal FPE (FP exception) and terminated

Abort Cause	Description
Signal_BUS	The process received the signal BUS (bus error) and terminated
Signal_SEGV	The process received the signal SEGV (access violation) and terminated
Signal_PIPE	The process received the signal PIPE (broken pipe)
Signal_ALRM	The process received the signal ALRM (timer alarm) and terminated
Signal_TERM	The process received the signal TERM and terminated immediately
Signal_XCPU	The process received the signal XCPU (time is up) and terminated
Signal_XFSZ	The process received the signal XFSZ and terminated
Signal_VTALRM	The process received the signal VTALRM and terminated
Signal_PROF	The process received the signal PROF (profiler) and terminated
Signal_USR1	The process received the signal USR1 (user signal 1) and terminated
Signal	The process received an unidentified signal and terminated
Died	The process has terminated before starting up correctly
Killed	The application has been killed manually by the administrator
TimedOut	The process has timed out in sending a message

Table B.15: The MIB abort causes of a process

B.6 User Management Data

The user management stores the following information for each user in the MIB:

Value Name	Value Description	Unit	Example	C	M	T	F
UserID	ID of the user (used by OS)		10083				
UserName	Username (used by OS)		nemecek				
LastName	Last name of the User		Nemecek				
FirstName	First name of the user		Josef				
Department	Department of the user		Development				
eMail	e-Mail address		nemecek@epfl.ch				
Address	Business Address		CAPA H5, EPFL				
Phone	Phone number		24527				
Pager	Pager number		324527				
AvCredits	Available credits	CRE	424		S	L	
UsedCredits	Consumed credits	CRE	201				
BurnedCredits	«Burned» (wasted) credits	CRE	10		S	L	
CreditAddRate	Credits to add per week	CRE/w	20	B			
CreditBurnRate	Credits wasted this week	CRE/w	0		S	L	
Quota	Allowed storage consumption	MBytes	2048	B	S	L	
SubApps	Applications submitted		225				
AbortedApps	Applications aborted		2		S	L	
Permissions	User Permissions			B			

Table B.16: The MIB user values

This sample user MIB is based on the assumption, that the resource management uses credits to charge the resource usage. The usage history can be stored separately for statistical purposes.

B.7 Resource Management Data

The resource management stores the following values for each queue in the MIB:

Value Name	Value Description	Unit	Example	C	M	T	F
QName	Queue name		Weekday	B			
Users	Users allowed to submit			B			
Availability	Queue Availability		Mo–Fr 0600–1800	B			
Credit	Credits cost per CPU and hour	CRE/C/h	1	B			
Status	Current queue status		Open		E		
Queue	Tree with queued applications						
Active	Tree with running applications						
Finished	Tree with finished applications						

Table B.17: The MIB queue values

For each application, the data as described in the application management is stored in the MIB.

B.8 COSMOS MIB

The COSMOS MIB described next has the same structure as the MIB presented in this appendix, but the coverage is not as complete: Some MIB values are reserved, but not implemented in COSMOS and they are written in the following tables using *italic letters*.

B.8.1 COSMOS MIB Branches

COSMOS only covers three management modules, described by three main branches in the MIB:

Name	MIB Main Branch
0	Node Management
1	SAN Management
2	Application Management

Table B.18: Main branches of the COSMOS MIB

Further names are reserved, but not used by COSMOS.

B.8.2 COSMOS Node Management MIB Branch

The node management branch of the MIB is identified by names starting with «0». Requesting the value for «0» returns the number of nodes in the supercluster. The second value «n» in the name is the node number (e.g. «0.4» for the node number 4). The third value in the name represents information about the node. Some names have sub-trees (such as CPUs or LAN NICs) with the same logic as sketched here.

Name	Description	Unit	Example	C	M	T	F
0	Number of Nodes		32	B			
0.n.0	*Node Name*		*COMPAQ DS-20e*				
0.n.1	*Serial Number*		*SN0815-42*				
0.n.2	Host Name (Alias)		Swiss-T1 Node 12	B			
0.n.3	*Room Number*		*AGD 23.0*				
0.n.4	*Rack Number*		*42*				
0.n.5	*Position in Rack*		*4*				
0.n.6	OS Name		OSF1		S	C	
0.n.7	OS Version		V5.0		S	C	
0.n.8	Status		Ready		E		×
0.n.9	*Temperature*	*°C*	*35*		*S*	*L*	
0.n.10	*RAM Size*	*MBytes*	*1024*		*S*	*C*	
0.n.11	*RAM Used*	*MBytes*	*744*		*S*	*L*	
0.n.12	Processors (separate tree)		2				
0.n.12.c.0	*CPU Name*		*Alpha EV67*		*S*	*C*	
0.n.12.c.1	*Revision*		*2*		*S*	*C*	
0.n.12.c.2	*Clock Frequency*	*MHz*	*667*		*S*	*C*	
0.n.12.c.3	*Load*	*%*	*75*		*Q*	*C*	
0.n.12.c.4	*Productive Application Load*	*%*	*73*		*Q*	*C*	
0.n.13	*Storage Devices (separate tree)*		*2*				
0.n.13.s.0	*Capacity*	*MBytes*	*30000*		*S*	*C*	
0.n.13.s.1	*Used Space*	*MBytes*	*12345*		*S*	*C*	
0.n.14	LAN NICs (separate tree)		1				
0.n.14.l.0	*NIC Description*		*DE-500E*		*S*	*C*	
0.n.14.l.1	LAN Identities (separate trees)		1				
0.n.14.l.1.i.0	IP Number		192.168.100.102				
0.n.14.l.1.i.1	Host Name		t1-12.epfl.ch				
0.n.15	SAN NICs (separate tree)		2				
0.n.15.s.0	*NIC Description*		*SCS T-Net*				
0.n.15.s.1	*Serial Number*		*202-PCI-011-0042*		*S*		
0.n.15.s.2	*Version*		*2.54*		*S*		
0.n.15.s.3	Board ID		42		S		
0.n.15.s.4	Status		Ready		E		×
0.n.15.s.5	Links (separate tree)		1				
0.n.15.s.5.l.0	*Status*		*Up*		*E*		×
0.n.15.s.5.l.1	*Fatal Errors*		*0*		*E*		×

Name	Description	Unit	Example	C	M	T	F
0.n.15.s.5.l.2	*MIA Status*		*N/A*		*E*		×
0.n.15.s.5.l.3	*Send Rate*	*MB/s*	*12*		*Q*	*C*	
0.n.15.s.5.l.4	*Receive Rate*	*MB/s*	*10*		*Q*	*C*	
0.n.15.s.5.l.5	*Error Rate*	*Errors/s*	*40*		*Q*	*C*	

Table B.19: Node management branches and sub-trees

SAN NIC monitoring was not implemented in COSMOS, because the T-Net SAN NIC had no monitoring registers implemented. The space of the FPGA chips on the NIC was completely consumed by the logic for productive work. COSMOS was only able to retrieve the static information from a PROM chip, provided by the SAN NIC driver, which had been extended with custom PROM programming code provided by the author.

B.8.3 COSMOS SAN Switch Management MIB Branch

The SAN switch management branch of the MIB is identified by names starting with «1». Requesting the value for «1» returns the number of SAN switches in the supercluster. The second value «s» in the name is the switch number (e.g. «1.4» for the switch number 4). The third value in the name represents information about the switch. One name has a sub-tree (SAN links) with the same logic as sketched here.

Value	Description	Unit	Example	C	M	T	F
1	Number of SAN Switches		8				
1.s.0	*Description*		*SCS T-Net Switch*				
1.s.1	*Serial Number*		*202-SWI-011-00001*		*S*	*C*	
1.s.2	Switch Name (Alias)		Swiss-T1 Switch 1	B			
1.s.3	Switch Number		1	B			
1.s.4	*Room Number*		*AGD 23.0*				
1.s.5	*Rack Number*		*42*				
1.s.6	*Position in Rack*		*4*				
1.s.7	*Version*		*V23.12.1*		*S*	*C*	
1.s.8	Status		Ready		E		×
1.s.9	Temperature	°C	35		S	L	
1.s.10	Links (separate tree)		12				
1.s.10.l.0	Status		Up		E		×
1.s.10.l.1	Fatal Errors		0		E		×
1.s.10.l.2	*MIA Status*		*N/A*		*E*		×
1.s.10.l.3	*Send Rate*	*MBytes/s*	*12*		*Q*	*C*	
1.s.10.l.4	Receive Rate	MBytes/s	10		Q	C	
1.s.10.l.5	Error Rate	Errors/s	40		Q	C	

Table B.20: SAN management branches and sub-trees

The monitoring capabilities of the SAN switch were limited, because the development and production of the T-Net switch ran out of time and space (as the T-Net SAN NIC). The developers of the T-Net NIC and switch were busy fixing bugs and had no time to add monitoring functionality as required for reliable system management software.

B.8.4 COSMOS Application Management MIB Branch

The application management branch of the MIB is identified by names starting with «2». Requesting the value for «2» returns the number of applications stored in the MIB. The second value «a» in the name is the application number (e.g. «2.4» for the application 4), but not for the application ID, since the ID can be re-used after finalization or abortion. The third value in the name represents information about the application. One name has a sub-tree (processes) with the same logic as sketched here.

Value	Description	Unit	Example	C	M	T	F
2	Number of Applications in MIB		200				
2.a.0	Application ID		42 (0/42)		S	C	
2.a.1	Name		a.out		S	C	
2.a.2	Path		/usr/epfl/test/		S	C	
2.a.3	Owner		100 (nemecek)		S	C	
2.a.4	Creation Time		01.08.00 12:34:56		S	C	
2.a.5	Start Time		01.08.00 12:35:00		S	C	
2.a.9	Last Barrier Time		01.08.00 12:35:00		S	C	
2.a.10	Barrier Count		1		S	C	
2.a.6	End Time		00.00.00 00:00:00		S	C	
2.a.7	Status		2 (Started)		E	C	×
2.a.8	Processes (separate tree)						
2.a.8.p.0	Node		Swiss-T1 Node 1		S	C	
2.a.8.p.1	Slot		0		S	C	
*2.a.8.p.2**	*Load*	*%*	*75*		*S*	*C*	
*2.a.8.p.3**	*Memory Size*	*MBytes*	*756*		*S*	*C*	
2.a.8.p.4	Process ID (PID)		4212		S		
2.a.8.p.5	Register Time		01.08.00 12:34:56		S		
2.a.8.p.6	Current Barrier Time		00.00.00 00:00:00		S		
2.a.8.p.7	Finish Time		00.00.00 00:00:00		S		
2.a.8.p.8	Abort Cause		0 (Normal)		E		×
2.a.8.p.9	Status		2 (Started)		S		
2.a.11	Abort						×

Table B.21: Application management branches and sub-trees

The application management MIB branch is of course the most developed part of the MIB, since it is the novelty in system management.

The name «2.a.11» is the only name that represents an action in the COSMOS MIB: When the user interfaces write the application ID into this name, COSMOS triggers an immediate abortion of the application, which takes about 1 second. This functionality was implemented after some problems had arisen, where the resource management software GRD started new applications while processes of a previously lost application blocked the reserved resources. With this feature, the application could be wiped out from the system and GRD was thus able to work correctly again.

C Management Functionality

This appendix contains the functionality of the supercluster management. The functionality is grouped into management areas and subsystems. The functionality described here is referenced first in section 2.3. The tables also contain the distribution of the functionality between the components center, proxy and agents – as described in section 3.4.

C.1 Configuration

The configuration functionality has a direct effect on the MIB: These actions change the component status, the behavior and configuration of the component.

C.1.1 Node Subsystem

Functionality	Covered by		
	Center	Proxy	Agent
Power Control	Power-On/Off, Boot	Power-On/Off, Boot	Reset, Shutdown
OS / Firmware Image Control	Creation, Distribution	Distribution	Installation
Application / Service Control	Decision, Distribution	Distribution	Execution

Table C.1: Node Subsystem Configuration Functionality

The *Power Control* functionality manages the node status. Current operating systems only allow the node to be shut down or reset. Powering the node on and off or selecting the right image to boot from is done using the console server. The centers or proxies use the terminal server connections to perform these tasks.

Upgrades of the OS, libraries or software packages are performed through the *OS / Firmware Image Control* functionality. The image that is installed on the nodes by the node agent is created on the centers and distributed by the centers and proxies.

The execution of local applications or services is controlled by the *Application / Service Control* functionality. All services and applications that are required on the nodes are defined by the administrators, managed by the centers, distributed by the centers and proxies, and executed on the nodes. These services include OS services (FTP, libraries etc.) and the execution of applications (e.g. updates of local files or applications).

C.1.2 SAN Subsystem

The SAN subsystem manages the SAN switches (managed by the SAN agent) and the SAN NICs (managed by the node agent). There is therefore one table which describes the functionality for the SAN switch and one which describes the SAN NICs.

Functionality	Covered by		
	Center	Proxy	Agent
Power Control	Power-On/Off	Power-On/Off	Boot, Reset, Shutdown
Firmware Image Control	Creation, Distribution	Distribution	Installation
Routing Table Control	Calc., Distribution	Distribution	Installation
Basic Configuration Control	Creation, Distribution	Distribution	Installation

Table C.2: SAN Subsystem Configuration Functionality for SAN Switches

The *Power Control* functionality manages the SAN switch status. Under the assumption that the SAN switch agent cannot power on or off the switch hardware, the centers or proxies perform these tasks through terminal servers or remote power switches.

The SAN switch firmware, routing tables and individual basic configuration are all created by the centers, since they have the system knowledge to create the mandatory configuration data and files. This data is locally applied by the SAN switch agents.

The SAN switch agent can be run directly in the SAN switch using the management protocol. If the SAN switch uses a different management protocol, the SAN switch agent runs outside the switch, and translates the management protocols.

Functionality	Covered by		
	Center	Proxy	Agent
Firmware Image Control	Creation, Distribution	Distribution	Installation
Routing Table Control	Calculation, Distrib.	Distribution	Installation
Basic Configuration Control	Creation, Distribution	Distribution	Installation

Table C.3: SAN Subsystem Configuration Functionality for SAN NICs

The SAN NIC firmware, routing tables and individual basic configuration are all created by the centers, since they have the system knowledge to create the mandatory configuration data. This data is locally applied by the node agents, using the SAN NIC driver.

C.1.3 LAN Subsystem

The LAN subsystem manages the LAN switches (managed by the LAN agent) and the LAN NICs (managed by the node agent). There is therefore one table which describes the functionality for the LAN switch and one which describes the LAN NICs.

Functionality	Covered by		
	Center	Proxy	Agent
Power Control	Power-On/Off	Power-On/Off	Boot, Reset, Shutdown
Firmware Image Control	Creation, Distribution	Distribution	Installation
Routing Table Control	Calc., Distribution	Distribution	Installation
Basic Configuration Control	Creation, Distribution	Distribution	Installation

Table C.4: LAN Subsystem Configuration Functionality for LAN Switches

The *Power Control* functionality manages the status of the LAN switch. Under the assumption that the LAN switch agent cannot power on or off the switch hardware, the centers or proxies perform these tasks through terminal servers or power switches.

The LAN switch firmware, routing tables and individual basic configuration are all created by the centers, since they have the system knowledge to create the mandatory configuration data and files. This data is locally applied by the LAN switch agents.

Common LAN switches are usually managed with SNMP [Ros96]. The centers and proxies can either communicate using SNMP with the LAN switches, or a switch-external LAN agent translates the management protocol to SNMP.

Functionality	Covered by		
	Center	Proxy	Agent
Firmware Image Control	Creation, Distribution	Distribution	Installation
Routing Table Control	Calc., Distribution	Distribution	Installation
Basic Configuration Control	Creation, Distribution	Distribution	Installation

Table C.5: LAN Subsystem Configuration Functionality for LAN NICs

The LAN NIC firmware, routing tables and individual basic configuration are all created by the centers, since they have the system knowledge to create the mandatory configuration data and files. This data is locally applied by the node agents, using the LAN NIC management functionality of the OS.

C.1.4 Storage Subsystem

Functionality	Covered by		
	Center	Proxy	Agent
Power Control	Power-On/Off, Boot	Power-On/Off, Boot	Reset, Shutdown
OS / Firmware Image Control	Creation, Distribution	Distribution	Installation
Basic Configuration Control	Creation, Distribution	Distribution	Installation

Table C.6: Storage Subsystem Configuration Functionality

The storage subsystem can be implemented as a separate subsystem with dedicated computers and towers of storage devices, connected via LAN or SAN to the supercluster components. Another approach is to integrate the storage subsystem into the nodes.

Therefore the *Power Control* functionality only makes sense if dedicated hardware is used for the storage subsystem.

As for the nodes, the centers decide the basic configuration and OS or firmware of the storage devices. They and the proxies then distribute this data which is enforced by the agents.

C.1.5 Application Subsystem

Functionality	Covered by		
	Center	Proxy	Agent
Spawning / Application Start	Distribution	Distribution	Spawn («fork»)
Signal / Application Control	Distribution	Distribution	Execution
Basic Configuration Control	Creation, Distribution	Distribution	Installation
Checkpointing	Decision	Distribution	Execution

Table C.7: Application Subsystem Configuration Functionality

The application is started using the *Spawning* mechanism. The simplest implementation of this is the creation of named pipes for input/output redirection (stdin, stdout, stderr) and management information exchange (communication channel for the application management library described in 3.1.5), forking away a child process (duplication of a process) that is closing any unused connections, changing the user, and overloading itself with the application image.

The application management requires signals to be sent to the application: A signal to indicate when the reserved resources have been consumed (XCPU), a signal that aborts the application (TERM), a signal that suspends or resumes the application and many more. Applications can also request abortions (if an application-internal fault has been detected).

The application management also associates each application process with node resources. The slot (usually one slot per CPU, one SAN NIC channel per slot and one process per slot) is either configured statically, or the resources are assigned dynamically at application startup.

The checkpointing mechanism regularly stores synchronized images of all processes to persistent storage. These images allow the application to be restarted from this synchronized point in the case where one or more processes need to be migrated to another node.

C.1.6 Resource Subsystem

The resource subsystem only consists of the center module, since it is the «planning unit» that manages the application subsystem and consumes monitoring data and fault/trap messages of all the other subsystems.

Functionality	Covered by		
	Center	Proxy	Agent
Queue Handling	Management	–	–
System Configuration	Management	–	–

Table C.8: Resource Subsystem Configuration Functionality

The users enter requests to start applications into queues. Scheduling mechanisms decide which applications are to start as soon as enough resources become available. The selected applications are passed to the application management module in the center.

Each queue has a profile that includes the following:

- Open to certain users, projects, or departments (e.g. queue for students)
- Open to applications of a certain size or type (e.g. queue for batch-type applications of at most 16 CPUs or queue for debugging applications of at most 32 nodes)
- Scheduling of applications only during certain hours (e.g. night queue, where applications are only scheduled for startup between 1800 and 0800 hours)
- Scheduling on nodes of a certain type (e.g. interactive nodes with fast network and specialized graphics hardware or nodes of specially protected partitions)

Besides the queues, the resource management also includes the system configuration planning which in turn includes the following jobs:

- System (or partition) power-on, boot, reset, shutdown and power-off
- Application of a set of basic configurations (e.g. red/black partitioning of the ASCI Red)
- Queue opening or closing

All these jobs are performed on behalf of the administrators, using the functionality of all other subsystem management modules available.

C.1.7 Management Subsystem

Functionality	Covered by		
	Center	Proxy	Agent
Power Control (Center)	Reset, Shutdown	–	–
Power Control (Proxy)	Power-On/Off, Boot	Reset, Shutdown	–
Basic Configuration Control	Creation, Distribution, Installation	Distribution, Installation	Installation

Table C.9: Management Subsystem Configuration Functionality

Also the management subsystem needs to be managed. Booting the management is the trickiest process, since the centers are not working. The administrators have to start one of the centers, which in turn starts the other centers, which turn on the proxies. If the whole supercluster management is powered up, they power up all components. Shutting down the management works in reverse order, where all components are turned off, followed by the proxies, followed by centers. The last center is turned off by the administrators.

The basic configuration is created by the administrators and stored in the MIB. Changes in the management configuration are calculated by the centers and distributed to the proxies, which then distribute the configuration on to the agents.

C.2 Monitoring

Monitoring has the following tasks:

- Deliver the current performance data at a convenient update rate.
- Provide historical data at a convenient resolution.
- Allow special monitoring data gathering for development and debugging.

All tasks require some smart implementation ideas.

C.2.1 Gathering Current Performance Data

The current performance at a convenient update rate is limited by the available bandwidth between user interface applications (consumers) and the centers and proxies (producers). The proxies hold the current (and historical) performance data, reducing the network load in the management subsystem for the monitoring functionality.

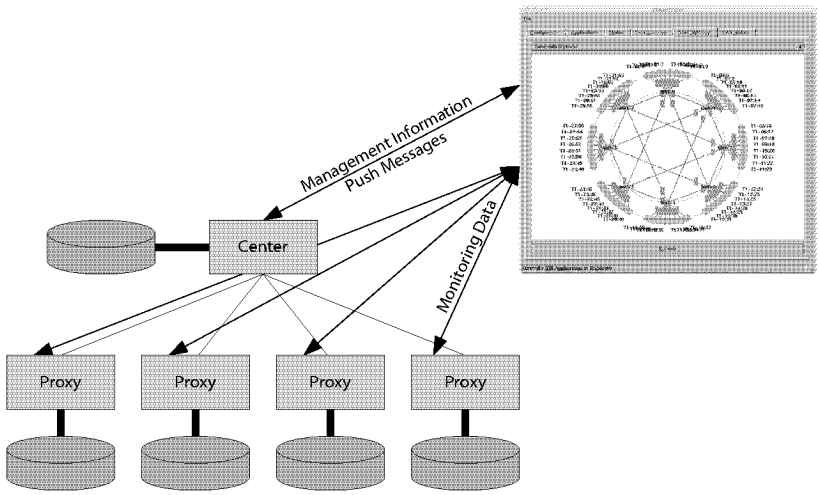


Figure C.1: Connections for monitoring between GUI, centers and proxies

With this architecture, the monitoring data update rate only depends on the available rate between the management (center and proxies) and the user interfaces.

A supercluster consisting of 10 000 nodes (1 CPU and 1 SAN port for one process) and 1 000 switches (16 ports, 1 switch per 10 nodes) produces about 1 MBytes of data per monitoring data sample if not too many additional monitoring features are needed.

One of these «additional monitoring features» may include separating SAN traffic into applications and processes for debugging and profiling purposes. This can be achieved by using «virtual channels» with one set of registers per channel. 49 channels for the separation (and one for the rest) allow separation of the traffic into 3 applications of 16 processes each (or one application with 49 processes). This functionality increases the sample size to 14 MBytes. This rate saturates a fast Ethernet connection to the workstation running the GUI.

With 4 channels only, the sample size is 1 545 kBytes, saturating the LAN link at a rate of 10 Hz.

C.2.2 Storing Historical and Statistical Data

Monitoring data cannot be stored forever – at least not in the initial full sampling rate. The monitoring data from the proxies is statistically reduced and forwarded to the centers. They reduce the monitoring data further as the data ages.

For statistical reasons, taking the average value for the history only is too coarse. It is better to take the average value, plus the minimum and the maximum value per period.

The following table assumes a sample size of 1 MBytes (see above).

Stored Time	Storage	Sampling Rate	Est. Size	Req. Bandwidth
1 Hour	Proxy	1 Second	10.5 GBytes	3 MBytes/s
1 Day	Proxy / Center	10 Seconds	25.3 GBytes	300 kBytes/s
1 Week	Center	1 Minute	29.5 GBytes	52 kBytes/s
1 Month	Center	5 Minutes	26.2 GBytes	10 kBytes/s
1 Year	Center	1 Hour	25.7 GBytes	1 kBytes/s
Total			**117.2 GBytes**	

Table C.10: Required storage for archiving monitoring data for a 10 000 nodes system

The data amount handled by the center is still impressive. The rightmost column shows the required network bandwidth if all data is sent to the center.

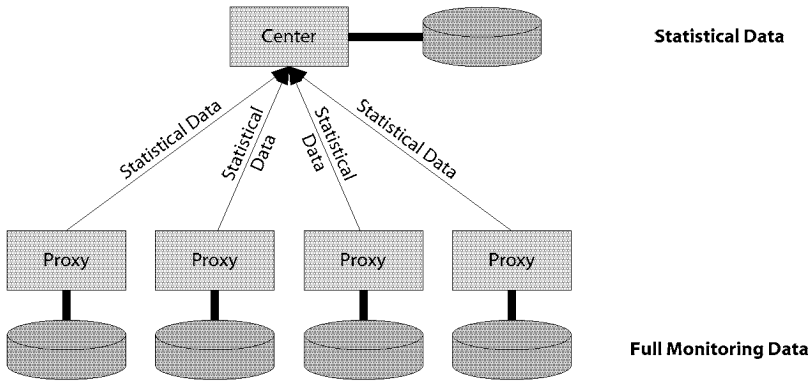


Figure C.2: Monitoring data in proxies and statistical data in center

A sampling rate of one second for the current performance data is usually high enough. Higher rates are only required if detailed information is needed, e.g. for debugging or profiling.

C.2.3 Special Monitoring Features

Another point is the extended monitoring for debugging: The application developer needs support in developing, debugging and testing parallel applications – print messages and syslog entries are not precise enough. The following table summarizes some debugging functionalities and the estimated required size for a 100-process job (with 10 SAN switches) running for one hour. The «amount» is the total number of all events of all nodes during the entire execution time.

Functionality	Size	Amount	Estimated Size
MPI call logging with timestamp	6 Bytes	14 400 000	90 MBytes
Debug prints with timestamp	40 Bytes	1 000 000	40 MBytes
Message calls with timestamp	8 Bytes	3 600 000	30 MBytes
Message trace logging with timestamp	10 Bytes	10 800 000	108 MBytes
Node load	2 Bytes	36 000 000	72 MBytes
SAN Switch load	160 Bytes	3 600 000	576 MBytes
Total			**916 MBytes**

Table C.11: Required data for detailed monitoring of a 100-process job for 1 hour

MPI call logging stores the command and timestamp of each MPI call (2 Bytes for the function ID, 4 Bytes for the timestamp). In addition, MPI message calls take the information of the target process and tag (an additional Byte each). Each process calls 40 MPI commands and 10 MPI message calls per second.

The most common debugging method is to print supporting information at current program positions. The printed line usually includes current variable values and status data (36 Bytes for the string and 4 Bytes for the timestamp). Each process prints 3 lines per second.

The most interesting part is precise message tracing: Which message, from which source, going to which target, with which tag, entered or left which component, through which port at what time. A 100-nodes system has 10 switches with 16 ports each, leading to 260 ports (100 NIC ports and 160 switches ports). Tracing a unicast message passes 6 ports, each of which generates one log entry (4 Bytes timestamp, 2 Bytes port, 4 Bytes source/destination/tag information). Each node sends 5 MPI messages per second.

The node and SAN switch load is taken 100 times per second. The node delivers the current CPU and memory usage in percent (each 1 Byte). The SAN switch delivers the current bandwidth and error rate (1 Byte for errors and 4 Bytes for bandwidth per direction) for each of the 16 ports.

The total amount of data produced during this one hour task is 916 MBytes, which requires about 250 kBytes/s of bandwidth.

The result is a global log of all processes and debugging-relevant events.

C.3 Fault Handling

Murphy once said, that everything that can fail, will. And if there is a combination of failures that lead to a disaster, this combination will happen when it is least welcome, e.g. during an official demonstration day with many governmental officials and journalists present[8].

[8] The most famous catastrophe of this nature was the baggage claim system inauguration day at Denver airport in December 1994, where baggage was cut into pieces, and smashed by switches – everything observed by invited journalists and VIPs.

The fault handling needs definitions of faults and how to handle them. There are some principal strategies in identifying faults and finding procedures to cure them:

- Be precise in fault definitions, since the general cure may create too much overhead (e.g. turning off a SAN switch with bad SAN links instead of turning off the bad ports only).
- Do not over-specify faults if many of them are handled identically (e.g. if the NIC is defective, you do not need to find out what is wrong, it will be replaced anyway).
- Do not check for faults or handle them if it is easier for the personnel to fix the problem (e.g. a broken LAN link between node and switch is replaced as it is detected by the personnel and there is no need for the management to try to contact the node through other communication links such as terminal server or SAN).

The following tables list some faults, show by whom they are detected and how they can be cured by the management or with human help. The lists are – of course – not complete.

C.3.1 Node Management

The node crash is simplest fault: The node simply disappears, and the LAN connection (and often also all other communication channels) is just dropped. The node is taken out of the pool of nodes that are available for processing and anything that was running on this node needs to be restarted. Since the SAN may contain packets of or for this node, these applications' packets need to be dropped before they block buffers and channels.

Fault	Detected by	Handling
Node crashed	Proxy / Center Framework	Remove node from pool Power-off node effectively Abort affected applications on nodes Flush SAN from applications' messages Restart applications on new pool
Node-internal fault	Node Agent Framework	As «node crashed»
Bad communication	Proxy / Center Framework	As «node crashed»

Table C.12: Faults of the node subsystem

Node-internal faults include faults of any subsystem that is not managed by a management module of the framework (CPU, CPU fan, power supply, power fan, system storage, memory, operating system) or the management itself (lost demon, shared memory region etc.). The fault cannot usually be solved by the software and requires administrative help. Until this help becomes available, the cause is stored in the MIB and the node is turned off.

Bad communication is usually a sign of a crashed management agent that requires the node agent to be restarted (and therefore the whole node).

Although the cure is always the same, the differentiation into these categories makes sense, since the administrator must find the problem cause manually.

C.3.2 SAN Management

The node has potentially more than one SAN NIC inserted into it and the SAN NIC may have more than one port. If one of the ports (or one of the NICs) has failed, only the associated slot is removed from the pool and the applications are aborted and restarted. If the only port (or the last remaining port) has failed, the node consequently becomes useless and requires power-off.

Fault	Detected by	Handling
Switch crashed	Proxy / Center Framework	Remove connected nodes from pool Distribute new routing tables Power-off nodes and switch Abort affected applications on nodes Flush SAN from messages to switch Restart applications on new pool
Port fault (node)	SAN Agent Framework	Remove node from pool Power-off node Abort affected applications on nodes Flush SAN from applications' messages Restart applications on new pool
Port fault (switch)	SAN Agent Framework	Distribute new routing tables Re-route existing messages
Switch-internal fault	SAN Agent Framework	As «switch crashed»
Bad communication	Proxy / Center Framework	As «switch crashed»

Table C.13: Faults of the SAN subsystem (SAN switch)

Fault	Detected by	Handling
Port or NIC fault (one NIC / port per node)	SAN Module (Node Agent)	Remove node from pool Power-off node Abort affected applications on nodes Flush SAN from applications' messages Restart applications on new pool
Port or NIC fault (more than one NIC / port per node)	SAN Module (Node Agent)	Remove slot from pool Abort affected applications on nodes Flush SAN from applications' messages Restart applications on new pool
Bad message received	SAN Module (Node Agent)	Send report to center (firmware bug?) Check / update routing tables Abort source application Flush SAN from application's messages Restart application

Table C.14: Faults of the SAN subsystem (SAN NIC)

If a SAN switch crashes, all connected nodes become unavailable, are taken out of the pool and powered off. The crashed switch is powered off and all switches (at least the surrounding ones) receive new routing tables. The applications currently running on those nodes are aborted and restarted on a new set of nodes.

If only one port breaks (hardware defect, broken cable), the reaction depends on the nature of its connected partner. A connected node becomes unavailable and must be removed from the pool (as in «node crashed» in table C.12), whereas a disconnected SAN switch only requires new routing tables and re-routing packets in port buffers.

Switch-internal faults and bad communication is handled as in the node fault handling.

Modern SAN technologies use per-link retransmission protocols. Invalid messages are sent again automatically if not acknowledged by the receiver, which simplifies the implementation of parallel environments (e.g. MPI) and management software.

If a message that is not expected is received for one of the currently running processes (bad application ID or process rank, invalid tag etc.), the message header will be sent to the center which then decides what to do. The center usually checks and re-distributes the routing tables and aborts the source application (or sends the message to the waiting destination process and keeps the application running – as long as there are not too many misrouted messages).

C.3.3 LAN Management

Fault	Detected by	Handling
Switch crashed	Proxy / Center Framework	Remove connected nodes from pool Distribute new routing tables Power-off nodes and switch Abort affected applications on nodes Restart applications on new pool
Port fault (node)	LAN Agent Framework	Remove node from pool Power-off node Abort affected applications on nodes Restart applications on new pool
Port fault (switch)	LAN Agent Framework	Distribute new routing tables
Switch-internal fault	LAN Agent Framework	As «switch crashed»
Bad communication	Proxy / Center Framework	As «switch crashed»

Table C.15: Faults of the LAN subsystem (LAN switch)

The node has potentially more than one LAN NIC inserted into it and the LAN NIC may have more than one port. If one of the ports (or one of the NICs) has failed, only the routing tables need to be adapted to use alternative LAN connections for the same jobs. If the only port (or the last remaining port) has failed, the node becomes useless and requires power-off.

Because of the similar structures of SAN and LAN, the faults and their cures are also similar, except for the different routing behaviors (automatic retransmissions after timeout, dropped messages after exceeding the message-internal TTL (time to live)).

Fault	Detected by	Handling
Port or NIC fault (one NIC / port per node)	LAN Module (Node Agent)	Remove node from pool Power-off node Abort affected applications on nodes Restart applications on new pool
Port or NIC fault (more than one NIC / port per node)	LAN Module (Node Agent)	Distribute new routing tables

Table C.16: Faults of the LAN subsystem (LAN NIC)

Besides standard Ethernet-based communication, terminal server communication is also found in the LAN subsystem. Since there is no way to handle faults of terminal servers, only the administrator is notified.

C.3.4 Storage Management

Full storage devices quickly block the entire supercluster operation. The trap mechanism should prevent this situation by sending warning messages to the administrators when the capacity has almost been used. If the storage devices are filled with data, the oldest data will be moved onto slower storage space (e.g. tape devices, reserve storage) and the operation will continue. If this is not possible, the applications of users that have accounts on these devices will be suspended or not scheduled, since they consume and produce data on these storage devices.

Fault	Detected by	Handling
Storage crashed	Proxy / Center Framework	Remove devices from pool Suspend / Disable applications of users that have accounts on affected devices
Disk full	Storage Module (Agent)	Move old data to slower storage Notify administrators and users Stop scheduling or suspend applications of users on affected devices
Read / Write error	Storage Module (Agent)	Repeat access Remove disk from storage pool Suspend / Disable applications of users that have accounts on affected disk

Table C.17: Faults of the storage subsystem

With current technology, read/write errors are seldom and always an indicator for replacing storage devices. The access is usually repeated, and if the problem persists, the device is considered defective and taken out of the pool. Applications of users that have accounts on this device are stopped from being scheduled or suspended.

If the storage subsystem (or parts of it) becomes unavailable, operation must be (partially) suspended as application input cannot be gathered and output cannot be stored. At least those applications of users that have accounts on the affected devices must be stopped.

C.3.5 Application Management

The application is aborted in three cases: One of the processes has failed and just «disappeared», one of the processes encountered an error and requested an abortion, or one of the processes performed an illegal operation and has been aborted by the management. The abortion is followed by the removal of all messages of this application in the SAN. If the fault was not caused by the application itself, it can be restarted from the last checkpoint.

Fault	Detected by	Handling
Process abortion	Node Agent Framework	Abort application globally Flush SAN of application's messages Eventually restart application
Process abort request	Application Module (Node Agent)	Same as «Process abortion»
Spawn failure	Node Agent Framework	Repeat once Abortion in case of failure
Illegal operation	Application Module (Node Agent or Center)	Same as «Process abortion»

Table C.18: Faults of the application subsystem

The application can be aborted before it has been started. If one of the processes failed to spawn, all other processes must be aborted before the application has started.

C.3.6 Supercluster Management

The management subsystem can also be hit by a fault. Minor faults can be prevented using redundancy, resulting in performance degradation during the healing phase. Some fatal faults may still remain which cannot be resolved automatically and require administrative interaction.

Fault	Detected by	Handling
Center crashed	Other Frameworks	Wait until center has been restarted or re-distribute work to remaining centers
Proxy crashed	Other Frameworks	Wait until proxy has been restarted or re-distribute work to remaining proxies
Node Agent crashed	Proxy / Center Framework	Same as «Node crashed»
SAN Agent crashed	Proxy / Center Framework	Same as «Switch crashed»
LAN Agent crashed	Proxy / Center Framework	Same as «Switch crashed»
Storage Agent crashed	Proxy / Center Framework	Same as «Storage crashed»
MIB storage unavailable	Center / Proxy Framework	Use mirrored storage Suspend operation
Module malfunction	Framework	Restart module (if possible) Restart framework

Table C.19: Faults of the management subsystem

The effect of a proxy or center crash depends on the availability mechanism. In a reliable cluster, the workload will be distributed over the remaining cluster members (as long as the quorum is reached). If there is no high-availability mechanism present (or the quorum has been lost), the management service of the affected supercluster part will be suspended.

A crashed agent (Node, SAN, LAN, and Storage) is handled in the same way as if the component had crashed. The resource is removed from the pool, and applications are aborted or suspended, and components are re-configured or powered off.

The database that holds the MIB is essential for the operation. For high availability, it makes sense to mirror this database storage. If the storage is unavailable, the operation will be suspended and the database must be recovered by the administrators.

If the module of an agent, a proxy or a center malfunctions, the framework tries to reset the module (if possible – this depends on the design and implementation). If this fails, the framework will reset itself. If this fails as well, the framework will shut itself down that triggers the respective fault mechanism.

C.4 Trap Handling

The monitored values are observed by the management software. If the value breaks administrator-set limits, management actions are triggered automatically. Every monitored value can be protected by using a mechanism with two limits – a lower and an upper limit. These define the bandwidth in which the value is expected to be. A second pair of limits can be used to trigger further, more extreme actions. This situation is explained in Figure 2.17.

Not only the value itself is checked, but also the other components' values are checked. If the temperature of a node is 10 °C higher than its neighboring node (but lower than the limit), something might be wrong as well. If such a difference is found, an administrative message is sent, but no action is taken.

The idea of the trap management is to enable the detection of faults before they actually occur. Suspicious components can be replaced during scheduled downtime, before they cause a fatal fault resulting in unscheduled downtime that consumes time and money.

The following tables define the monitoring values that are usually protected by the trap handling mechanism, together with the actions that are usually triggered if a limit is broken.

C.4.1 Node Management

If the node temperature exceeds the first limit, the management tries to decrease the temperature by increasing fan speeds or air conditioning power. If the second limit is exceeded, the node is powered off to protect the hardware.

Value	Sample Limit(s)	Sample Actions
Node temperature	40 °C 60 °C	Increase / decrease air conditioning Switch off node
ECC memory error rate	1 Correction / s	Administrative alert
Storage error rate	1 Correction / s	Administrative alert
CPU usage	Idle: 1% / 10% Busy: 40% / 100%	Administrative alert
Memory usage	Idle: 5% / 20% Busy: 30% / 100%	Administrative alert
Storage usage	80%	Administrative alert

Table C.20: Node monitoring values protected by trap handling

Some node-internal subsystems have error correction mechanisms, such as the main memory or storage devices. The management must have access to the monitoring values of the error correction mechanisms, because the device must be replaced during scheduled downtime if the error rate is too high (compared to the other nodes' values).

The CPU and memory usage must not exceed certain limits when idle: There is something wrong if one idle node is busier (or less busy) than other idle nodes (virus, defect, crashed service). Also a different memory usage shows that there is a problem. Also differences between busy nodes can indicate problems on the node.

If local storage becomes full, the node must suspend operation. To prevent this situation, the administrator receives an alert and unused data can be cleaned up.

C.4.2 SAN Management

As with the nodes, the SAN switch hardware must also be protected if the temperature exceeds the predefined limits. The air conditioning performance is increased, the performance of the SAN switch is decreased or – if all else fails – the switch is powered off.

Value	Sample Limit(s)	Sample Actions
Switch temperature	40 °C 60 °C	Increase / decrease air conditioning Switch off SAN switch
«Soft» memory error rate	1 Correction / s	Administrative alert
Port error rate	1% 15%	Administrative alert Disable port

Table C.21: SAN monitoring values protected by trap handling (SAN switch)

Recoverable errors can also occur in the SAN NIC and switch, where error-correcting mechanisms hide «soft errors» and increase reliability. These non-fatal errors must be detected and – if they are too high – the component must be replaced during the next scheduled downtime.

High-speed networking technologies usually work close to their physical limits. Errors occur repeatedly, but are corrected by the hardware and software. If the error rate of a port is higher than the rate of other ports, the administrator receives a message that

allows him to replace the cable (usually the reason for bad communication performance) or other components. If the error rate is too high, the port is disabled.

Value	Sample Limit(s)	Sample Actions
Misrouted messages	10 / hour	Administrative alert
«Soft» memory error rate	1 Correction / s	Administrative alert
Port error rate	1% 15%	Administrative alert Disable port

Table C.22: SAN monitoring values protected by trap handling (SAN NIC)

C.4.3 LAN Subsystem

The LAN subsystem has the same trap mechanisms as the SAN subsystem.

Value	Sample Limit(s)	Sample Actions
Switch temperature	10 °C / 40 °C 0 °C / 60 °C	Increase / decrease air conditioning Switch off LAN switch
«Soft» memory error rate	1 Correction / s	Administrative alert
Port error rate	1% 15%	Administrative alert Disable port

Table C.23: LAN monitoring values protected by trap handling (LAN switch)

Value	Sample Limit(s)	Sample Actions
Misrouted messages	10 / hour	Administrative alert
«Soft» memory error rate	1 Correction / s	Administrative alert
Port error rate	1% 15%	Administrative alert Disable port

Table C.24: LAN monitoring values protected by trap handling (LAN NIC)

C.4.4 Storage Subsystem

Also the storage subsystem hardware is protected by the temperature trap, which increases the air conditioning or powers off the storage devices.

Value	Sample Limit(s)	Sample Actions
Storage temperature	40 °C 60 °C	Increase / decrease air conditioning Switch off node
Storage error rate	1 Correction / s	Administrative alert
Storage usage	80%	Administrative alert
Quota	80% 100%	User alert Stop scheduling user's applications

Table C.25: Storage monitoring values protected by trap handling

If non-fatal errors are detected at a higher rate than those of other devices, the administrator receives an alert identifying the device and can replace it during the next scheduled downtime.

If the storage usage exceeds a limit, the administrator receives an alert to archive old data or add more storage capacity as soon as possible.

Users with accounts on the storage subsystems have limited capacities reserved. If this quota is close to being consumed, the user receives an alert to clean up his account or request more space. If the quota is exceeded, the management stops scheduling the user's applications.

C.4.5 Application Management

The life of an application is the spawning (distribution of all processes to nodes), some barriers for global synchronization, and finalization (or abortion) at the end. Depending on the application and system size, spawning should not take more than 5 minutes – if it does, something has gone wrong and the application will be aborted. Barriers are used for global synchronization and all processes should enter this barrier in about the same time – if the processes have to wait for more than 5 minutes to continue, one process might be left hanging and the application will be aborted. If the application is still running five minutes after the first process has entered the finalization phase, some processes will be hanging and the application may be aborted. And finally, if the abortion of an application takes more than 5 minutes, the application is swept out of the slots.

Value	Sample Limit(s)	Sample Actions
Spawning time	1 min 5 min	Administrative & user alert Abortion
Barrier time	1 min 5 min	Administrative & user alert Abortion
Finalization time	1 min 5 min	Administrative & user alert Abortion
Abortion time	1 min 5 min	Administrative & user alert Forced sweep
CPU utilization	<50% for 5 min <5% for 5 min	User alert Abortion
Memory utilization	>100% of reserved space >125% of reserved space	User alert Abortion
Processing time	>100% of reserved time >125% of reserved time	User alert Abortion

Table C.26: Application monitoring values protected by trap handling

The processes are expected to use the available processing power. If the CPU usage drops below 50% for more than 5 minutes, the efficiency is too low and something might be wrong and the user is alerted. If the CPU usage is less than 5% for more than 5 minutes, the application seems to be blocked and is aborted.

Each application process receives a certain CPU time and main memory amount. If the memory consumption reaches 100% or no more CPU time is left, the user receives an alert. If the memory usage or CPU time reaches 125% of the reserved amount, the application is aborted.

C.4.6 Management Subsystem

Reliable clusters only operate if a quorum (50% + 1 member) is available. Only members within this quorum are allowed to operate, all others must stop. If the quorum is lost, operation must be suspended until either another member joins the cluster, or the administrator manually enables the minority to continue with the operation.

Value	Sample Limit(s)	Sample Actions
Quorum	50% + 1 member	Administrative alert
Channel Quality	1 Error / minute	Administrative alert
	1 Error / second	Disconnection & restart

Table C.27: Application monitoring values protected by trap handling

Communication channels between management components should be secured, since invalid messages can disturb the operation. If there is one invalid message per minute, the administrator will receive an alert. If there one bad message per second, the connection is closed and the error-generating partner is asked to reset.

C.4.7 User Subsystem

Users connect the management applications using usernames and passwords from trusted workstations. If the authentication mechanism rejects a user, one try per user per day is accepted, but if a user continues to use the wrong username and password, he will be blocked until the administrator re-enables the account after checking the reason for the rejection (intruder, bad memory).

Value	Sample Limit(s)	Sample Actions
Authentication Failed	1 try / day	Administrative alert
	10^{th} try of the day	Block user
Aborted Applications due to Programming Errors	1 Abort / day	Administrative alert
	10 Aborts / day	Stop scheduling user's applications
Credit Consumption	90%	Administrative & user alert
	100%	Stop scheduling user's applications
GUI Resource Consumption	20%	Administrative & user alert
	50%	Reject further GUI applications of user

Table C.28: User monitoring values protected by trap handling

Applications can abort for various reasons. The most unwanted reason for abortions are application bugs (dangling pointers, infinite loops, deadlocks). Applications are usually tested on small superclusters before they are allowed to run on the productive system. If applications continue to abort and are continuously queued by their users, scheduling

of their applications will be stopped. The administrator can address the problem by insisting that the users take some programming courses or requiring a programming expert to debug his applications.

Users consume resources, which may be counted with credits. If 90% of his daily or weekly credits have been consumed, the user receives an alert. When 100% of his credits have been used, his applications are suspended and scheduling of new applications for this user is stopped.

User interfaces consume bandwidth within the management LAN. If too many user interfaces are connected to the management, the management functionality may be compromised. The amount of data communicated between the management computers and the user interface is limited. If the limit is reached, the user interfaces receive data at a slower rate and connection requests of new user interfaces are blocked.

D Manual of the COSMOS CLI

The CLI is a command-line-based application that accepts commands with a specified syntax. The executable `cosmos_cli` starts the COSMOS CLI application.

Synopsis

```
cosmos_cli [ -c center ] [ -p port ]
```

The COSMOS CLI connects to the `center`. If none is given, the local host is taken. The COSMOS CLI connects to the `port`. If none is given, the default port 4242 is used.

Application Commands

Entering the sole command `app` returns the number of applications stored in the MIB.

```
COSMOS: app
Number of applications: 42
```

Entering the command `app` followed by a number returns the application information.

```
COSMOS: app 0
Application   0:       ID  180388626474 (42/42)
Application   0:     Name  ./mpitest9
Application   0:    Owner  11067 (nemecek)
Application   0:  Created  13.03.01 16:44:23
Application   0:  Started  13.03.01 16:44:24
Application   0: Finished  --.--.-- --:--:--
Application   0:   Status  2 (Started)
Application   0:     Size  2
```

Entering `app` followed by a number and `overview` and a number returns an overview, beginning from the specified application, containing the number of applications.

```
COSMOS: app 0 overview 2
Number of applications: 2 (from 0)
App#  Created           Started           Finished          Status      Size  User
   0  13.03.01 16:44:23  13.03.01 16:44:24  --.--.-- --:--:--  Started        2  11067
   1  13.03.01 16:44:34  13.03.01 16:44:34  --.--.-- --:--:--  Started        2  11067
```

Entering the command `app` followed by a number and abort and another two numbers aborts the application specified by the first number. The latter two numbers are needed for security and must be the two parts of the application number.

```
COSMOS: app 0 abort 42 42
Application aborted successfully
```

Node Commands

Entering the sole command `node` returns the number of nodes stored in the MIB.

```
COSMOS: node
Number of nodes: 16
```

Entering the command `node` followed by a number returns either the information about the node or an error message if the node does not exist.

```
COSMOS: node 2
Node   2:              Alias gs3
Node   2:              Status Ready
Node   2:              Number of slots: 2
Node   2         Slot  SAN ID  Status  Rank  Application ID
Node   2            0       2    Used     0  180388626474 (42/42)
Node   2            1       3    Used     0  180388626475 (42/43)
```

SAN Commands

Entering the sole command `san` returns the number of SAN switches stored in the MIB.

```
COSMOS: san
Number of SAN Switches: 2
```

Entering the command `san` followed by a number and `static` returns the static information of the SAN switch.

```
COSMOS: san 0 static
Switch   0:          Alias  scs-0
Switch   0:          Name   tonebts0.scs.ch
Switch   0:          Port   2001
Switch   0:          Serial 202-SWI-011-002-00007
Switch   0:          Status Up
Switch   0     Link  Type   Partner  Number
Switch   0        0    Node       9       0
Switch   0        1    Node       9       1
Switch   0        2    Node       2       2
Switch   0        3    Node       2       3
Switch   0        4    Node       4       4
Switch   0        5    Node       5       5
Switch   0        6    Node       6       6
Switch   0        7    Node       7       7
Switch   0        8    Node       8       8
Switch   0        9  Switch       1       9
Switch   0       10  Switch       1      10
Switch   0       11       -       0       0
```

Entering the command `san` followed by a number and `dynamic` returns the dynamic information of the SAN switch.

```
COSMOS: san 0 dynamic
Switch   0:          Temperature 35
Switch   0:          Uptime      1735 sec
Switch   0:          Status Up
Switch   0     Link  P/s in P/s out  Byte/s out  Ret/s  Status
Switch   0        0  519741  420055   53761192      60       0
Switch   0        1  478735  465218   59543524      58       0
Switch   0        2  420049  519743   66522048      60       0
Switch   0        3  465218  478740   61273785      59       0
Switch   0        4       0       0          0       0       1
Switch   0        5       0       0          0       0       1
Switch   0        6       0       0          0       0       1
```

E Bibliography

[Amd67] G. M. Amdahl: Validity of the single processor approach to achieving large scale computing capabilities; Proc. AFIPS Spring Joint Computer Conference 30, pages 483–485; April 1967.

[ATEL] ATEL Aare Tessin AG für Elektrizität Website; http://www.atel.ch/

[BB01] Victor R. Basili, Barry Boehm: COTS-Based Systems Top 10 List; IEEE Computer, Vol. 34, No. 5, pages 91–93; 2001.

[Bel92] Gordon Bell: Ultracomputers – A Teraflop Before Its Time; Communications of the ACM, Vol. 35, No. 8, pages 27–47; August 1992.

[BLF+99] Stephan Brauss, Martin Lienhard, Martin Frey, Martin Heimlicher, Andreas Huber, Patrick Müller, Martin Näf, Josef Nemecek, Roland Paul, Anton Gunzinger: An Efficient Communication Architecture for Commodity Supercomputing; International Conference of High-Performance Computing and Communications (SC99); 1999.

[Bra00] Stephan Brauss: A New Communication Architecture for Workstation Clusters; Ph. D. Thesis; Hartung-Gorre Series in Microelectronics, Vol. 108; 2000.

[Bro75] Frederick P. Brooks: The Mythical Man-Month – Essays on Software Engineering; Addison Wesley Longman Inc.; 1975.

[CERN] CERN Event Computing Homepage, see http://www.cern.ch/.

[CPQPF] Compaq Computer Corporation Internet Homepage; Information for Alpha Microprocessors and technologies such as Distributed File Systems (Petal/Frangipani) and others.

[CSG98] D. E. Culler, J. P. Singh, A. Gupta: Parallel Computer Architecture – A Hardware/Software Approach; Morgan Kaufmann Publishers; 1998.

[DMI] DMI Specification; http://www.dmtf.org/

[Don94] Jack J. Dongrarra: Performance of various Computers using standard linear Equations Software; Technical Report, CS-89-85, Computer Science Department, University of Tennessee; 1994.

[EPF00] Swiss-Tx Supercluster Home Page on the EPFL Website; http://capawww.epfl.ch/swiss-tx/

[Fly72] Michael J. Flynn: Some Computer Organizations and their Effectiveness; IEEE Transactions on Computers, Vol. C-21, No. 9, pages 948–960; 1972.

[GD+98] Ralf Gruber, Yves Dubois-Pèlerin, Swiss-Tx Group: Swiss Tx – First Experiences on the Swiss-T0 System; EPFL Supercomputing Review, No. 10, pages 19–23; 1998.

[GG97] Ralf Gruber, Anton Gunzinger: The Swiss-Tx Supercomputer Project; EPFL Supercomputing Review, No. 9, pages 21–23; 1997.

[Gon03] Jean Gonnord: HPC at CEA/DAM; Presentation at 7th SOS Workshop on Distributed Supercomputing; 2003.

[GRD] Codine/GRD Product Page of Sun Microsystems; http://www.sun.com/

[HPOV] Hewlett-Packard OpenView Information Page; http://www.openview.hp.com/

[Hüs97] René Hüsler: Intelligent Communication – A new Paradigm for Data-Parallel Programming; Dissertation ETH 12035; Hartung-Gorre Series in Microelectronics, Vol. 67, pages 57–63; 1997.

[IBM] International Business Machines (IBM) Internet Homepage; Information for Power Microprocessors and Technologies such as AltiVec.

[INTEL] Intel Internet Homepage; Information for Intel Microprocessors and Technologies such as MMX (Multi-Media Extension).

{Jaf84] Arthur Jaffe: Ordering the Universe: The role of mathematics; SIAM Review, Vol. 26, page 478; 1984.

[JM99] M. Jovanovic, V. Milutinovic: An Overview of Reflective Memory Systems; IEEE Concurrency, Vol. 7, No. 2, pages 56–64; 1999.

[KG99] Pierre Kuonen, Ralf Gruber: Parallel Computer Architectures for Commodity Computing and the Swiss-T1 Machine; EPFL Supercomputing Review, No. 11, pages 3–11; 1999.

[Lie00] Martin Lienhard: Concept and Implementation of an Efficient Communication Network for Commodity Supercomputing; Ph. D. Thesis; Hartung-Gorre Series in Microelectronics, Vol. 103; 2000.

[LSF] LSF Product Page of Platform Computing Inc.; http://www.platform.com/

{Mas71] Abraham Harold Maslow: The Farther Reaches of Human Nature; 1971.

[Mea96] G. Meares: Reflective Memory Network; Real-Time Magazine, Vol. 2, pages 51–62; 1996.

[MH98] Timothy G. Mattson, Greg Henry: An Overview of the Intel TFLOPS Supercomputer; Intel Technology Journal, 1st Quarter 1998.

[Mit98] Bradley Mitchell: Scalable Platform Services on the Intel TFLOPS Supercomputer; Intel Technology Journal, 1st Quarter 1998.

[Moo65] Gordon E. Moore: Cramming more Components onto Integrated Circuits; Electronics Magazine, Vol. 38, No. 8, pages 114–117; 1965.

[MPI95] Message Passing Interface Forum: MPI – A Message-Passing Interface Standard; University of Tennessee; 1995.

[MPI97] Message Passing Interface Forum: MPI-2 – Extensions to the Message Passing Interface; University of Tennessee; 1997.

[Mor03] John Morrison: The ASCI Q System at Los Alamos; Presentation at 7th SOS Workshop on Distributed Supercomputing; LA-UR-03-0541; 2003.

[Nem99] J. Nemecek: Network Topologies – Theoretical Study of Common Topologies; ETH Zürich and SCS; 1999.

[OPENMP] OpenMP Project Homepage; http://www.openmp.org/

[PBS] PBS Pro Product Page of Veridian Inc.; http://www.pbspro.com/

[PCB02] Yuansheng Pan, Gervais Chapuis, David Brown: Molecular Dynamics Simulations of Modulated Crystalline Structures; Supercomputing Review 13; EPFL; 2002.

[PTM96] J. Protic, M. Tomasevic, V. Milutinovic: Distributed Shared Memory – Concepts and Systems; IEEE Parallel and Distributed Technology; 1996.

[Qua00] RMS Reference Manual; Quadrics Supercomputers World Ltd.; Document Version 4-AA-RLAZA-TE; June 2000.

[Ros96] Marshall T. Rose: The Simple Book – An Introduction to Networking Management; Prentice-Hall; ISBN 0-13-451659-1; 1996.

[San96] ASCI Red Home Page on the Sandia National Laboratories Website; http://www.sandia.gov/ASCI/Red/

[Vos00] Mark Sawley: Numerical Flow Simulation for the America's Cup; Supercomputing Review 13; EPFL; 2002.

[SCa99] Mark L. Sawley, Paul W. Cleary: A Parallel Discrete Element Method for Industrial Granular Flow Simulation; Supercomputing Review 11; EPFL; 1999.

[SCH00] Mark L. Sawley, Paul W. Cleary, Joseph Ha: Using smoothed particle hydrodynamics for Industrial Flow Simulation; Supercomputing Review 12; EPFL; 2000.

[Sei92] Armin Seiler: Marketing – Erfolgreiche Umsetzung in die Praxis; Orell Füssli; 1992.

[SGL03] Thomas L. Sterling, William Gropp, Ewing Lusk: Beowulf Cluster Computing with Linux; MIT Press; ISBN 0-262-69292-9; 2003.

[SETI] Homepage of the SETI institute; http://www.seti.org/

[SGP01] Sahra Sedigh-Ali, Arif Ghafoor, Raymond A. Paul: Software Engineering Metrics for COTS-Based Systems; IEEE Computer, Vol. 34, No. 5, pages 44–50; 2001.

[SMF00] Rajesh Subramanyan, José Miguel-Alonso, José A. B. Fortes: A scalable SNMP-based distributed monitoring system for heterogeneous network computing; Technical Paper Supercomputing 2000; 2000.

[SSB+99] Thomas L. Sterling, John Salmon, Donald J. Becker, Daniel F. Savarese: How to Build a Beowulf; MIT Press; ISBN 0-262-69218-X; 1999.

[Ste98] W. Richard Stevens: UNIX Network Programming Volume 1 & 2; Prentice-Hall; ISBN 0-13-490012-X and 0-13-081081-9; 1998 and 1999.

[TKR+91] Andrew S. Tanenbaum, M. Frans Kaashoek, Robbert van Renesse, Henri E. Bal: The Amoeba Distributed Operating System – A Status Report; Computer Communications, Vol. 14, pages 324–335; 1991.

[Top500] Supercomputer Top500 List; Joint Project of the Universities of Tennessee (USA) and Mannheim (Germany); http://www.top500.org/

[Vol00] Pieter Volgers: HPC in Computational Structural Mechanics; Supercomputing Review 12; EPFL; 2000.

[Vos00] Jan Vos: Flow Simulations on High Performance Computers using the NSMB flow solver; Supercomputing Review 12; EPFL; 2000.

F Definition of Terms

«War is Peace», «Freedom is Slavery», «Ignorance is Strength».

George Orwell, «Inner Party» slogans from the novel «1984»

The following is an alphabetical list of the technical terms used throughout this document plus a brief explanation of their meanings.

Adaptive Routing
: Routing algorithm, where multiple ports for the same destination can be used, depending on current link load, status or other dynamic values. Opposite: Static routing.

Agent
: Software run on a managed component, managing that component.

API
: Application Programming Interface. Description and definition of calls provided by an application or library allowing modular programming.

Application (or Parallel Application)
: An application for superclusters is a set of executables that are distributed over the nodes. Every executable (called *Process*) knows the application to which it belongs (through the *Application ID*) and which *Rank* it has within the application. The processes exchange information through the SAN or LAN. All executables are handled as one entity that are started and stopped simultaneously on the supercluster.

Application ID
: The Application ID identifies the application within the supercluster.

ASCI Project
: Advanced Simulation and Computing Initiative. US-founded project to accelerate the computation performance of supercomputers thus allowing simulations of nuclear warhead arsenal.

ASIC (Application-Specific Integrated Circuit)
: Electronic chips in computers that perform specialized actions, the opposite of discrete parts. ASICs are used for microprocessors, I/O device controllers etc.

Authentication

Authentication is a part of the Security. It guarantees other communication entities the correctness and reliability of the presented identity. Authentication can be achieved as username/password mechanisms or as knowledge mechanisms, where a correct answer to a question must be found.

Availability

Availability is the percentage of time a specific service is fully available to a consumer. An availability of 100% electrical power for a computer can be achieved using uninterruptible power supplies (UPS) that hold the power for a certain time. A high availability of services can be achieved using Redundancy.

Barrier Synchronization

Synchronization mechanism, where all participating members are required to be present. This mechanism is used between iterations of an algorithm, after (or before) the results are distributed to all other nodes.

BLACS (Basic Linear Algebra Communication Subprograms)

Software package for linear algebra. The library is portable and available for parallel supercomputer architectures.

Blade Server

Computer architecture, where the computer parts (CPU, memory, some I/O) and the supporting parts (power, LAN, storage) are separated into blades (computers) and case backplane (supporting parts), For supercomputing, the blades can be interpreted as processing elements (PE), the backplane with the blades inserted as node.

Broadcast

Network communication pattern, where a message is multiplied (either by the network or by the source node) and sent to all possible destinations.

Center

Software running on a central computer, managing the agents on the managed component, potentially through proxies.

CLI (Command Line Interface)

User interface application, where information is shown as text and entered over the keyboard. Opposite of GUI.

Commodity [parts-based] Supercomputers

Supercomputers that are built from commodity off-the-shelf parts, designed for non-supercomputing use. These parts include processors, memories, I/O devices and many others. The opposite is *Custom [parts-based] Supercomputers*.

COW (Cluster of Workstations)

A COW system consists of a number of standard workstations that are interconnected by a fast network (such as T-Net or MyriNet). Special software enables the COW to be used as a supercomputer (job queue, inter-process communication, math libraries). Only dedicated computers can act as a node of a supercomputer.

Custom Supercomputers (or Full-Custom Supercomputers)
Supercomputers that are built from custom parts, only designed for supercomputing use. These parts include processors, memories, I/O devices and many others. The opposite is *Commodity [parts-based] Supercomputers*.

Debugging
Removing programming errors from applications.

EFLOPS
Short for Exa FLOPS (Floating Point Operations per Second), equal to one quintillion (1 000 000 000 000 000 000 = 10^{18}) operations per second.

Encryption
Encryption is a part of the Security. Data in encrypted into unreadable data that can only be decrypted back into readable data by the other communication entity. This is achieved by encryption algorithms that are either symmetric (encryption and decryption use the same keys and algorithms) or asymmetric (encryption and decryption use different keys and/or algorithms).

Event Synchronization
Synchronization mechanism, where one member indicates to all other members that something happened. This mechanism is used in algorithms, where all members search for a result and the first member to find the result stops the search.

FLOPS
Short for Floating Point Operations per Second.

GFLOPS
Short for Giga FLOPS (Floating Point Operations per Second), equal to one billion (1 000 000 000 = 10^{9}) operations per second.

GRID Computing
A supercomputing philosophy, where computing resources are decoupled from their consumers, and compared to water or electrical power. Computing plants provide their services to users, no matter where they are. The consumption is measured and accounted.

GUI (Graphical User Interface)
User interface application, where information is shown and entered graphically, using graphs, gauges, meters and many more. Opposite of CLI.

Integrity
Integrity is part of the Security. It guarantees the receiving communication entity that the data it has received is identical to the data that was sent – no data has been modified, deleted or added to the data during transmission. This is achieved by using an algorithm that adds some kind of checksum that is created by the sender and afterwards checked by the receiver.

LAN (Local Area Network)

The Local Area Network (LAN) interconnects all components of the supercluster and is used for socket-based communication such as NFS, FTP, Telnet and more. The commonly used technologies are Ethernet and Token Ring.

LINPACK

Software package for linear algebra, created by Jack Dongarra [Don94]. The library is portable and available for various supercomputer architectures, no matter whether scalar, parallel or vector-type.

Master/Backup System

Reliability mechanism, where the master system is in operation and a backup system takes over the operation as soon as an arbiter decides that the master system is unavailable.

MFLOPS

Short for Mega FLOPS (Floating Point Operations per Second), equal to one million (1 000 000 = 10^6) operations per second.

MIB (Management Information Base)

The MIB contains a hierarchical representation of management values that can be read, written and created. Most network hardware allows access to the MIB using the *Simple Network Management Protocol (SNMP)*.

MIMD (Multiple Instruction [Streams], Multiple Data [Streams])

Architecture for processors or computers [Fly72]. The processing unit executes instructions from multiple instruction streams and operates on data from multiple data streams. This architecture is used in multiprocessor architectures.

MISD (Multiple Instruction [Streams], Single Data [Stream])

Architecture for processors or computers [Fly72]. The processing unit executes instructions from multiple instruction streams and operates on data from one data stream. This architecture is used in architectures for audiovisual processing.

MPI (Message Passing Interface)

Standardized library for communication between processes of parallel applications. The current version with many hundred calls supports – besides simple message passing – parallel I/O and storage access.

MPICH

MPI implementation available for various platforms, e.g. Windows using Ethernet. Very portable but not very efficient on high-speed SAN products.

MPP (Massive Parallel Processing)

Computing architecture where a huge amount of processors participate.

MTBI (Mean Time Between Interruptions)

The Mean Time Between Interruptions (MTBI) is the average time that elapses before a component or system fails or requires repair. If a device has an MTBI of 1000 hours, the device will, on average, operate for 1000 hours before a failure occurs.

Multicast

Network communication pattern, where a message is multiplied (either by the network or by the source node) and sent to multiple destinations. Multicasts are used for group-internal broadcast messages.

Multicomputer

Parallel supercomputer architecture with many independent computers that are interconnected with a fast network. Usual architecture of NOW and COW systems.

Multiprocessor

Parallel supercomputer architecture with a fast network that interconnects memory, I/O, and processors. Usual architecture for SMP and NUMA systems.

Mutual Exclusion

Mechanism that allows for multiple processes using shared resources, e.g. for exclusive access or reader/writer algorithms.

NIC (Network Interface Card)

The Network Interface Card (NIC) connects the computer to a network. The NIC sends and receives data to and from the network. The driver and operating systems allow applications to exchange data between computers.

Node

Computer containing hardware and software for participating in parallel supercomputing applications. A node contains one or more processing elements, storage devices and network interfaces.

NOW (Network of Workstations)

A NOW system consists of a number of standard workstations that are interconnected with a standard network (such as Ethernet). Special software enables the NOW to be used as a supercomputer (job queue, inter-process communication, math libraries). Every computer in a LAN can be used as a node of a supercomputer.

NUMA (Non-Uniform Memory Access)

In NUMA-based computers, every processor has its own memory that is accessed directly. Every processor can also access the memory of all other processors, but this access is handled by special hardware, using a network. Accessing distant memory takes more time and has a lower bandwidth than accessing the own memory.

PE (Processing Element)

Physical unit for calculation, including one processor and memory. A node can contain multiple processing elements.

Peak Performance

Performance level which the manufacturer guarantees cannot be exceeded.

PFLOPS

Short for Peta FLOPS (Floating Point Operations per Second), equal to one quadrillion (1 000 000 000 000 000 = 10^{15}) operations per second.

Process

A Process is a part of a (parallel) *Application* started on the *Supercluster*. It is identified by its *Application ID*, its individual *Rank* within the application, and the executable and user information. The process is started on a node, it then allocates resources and communicates to other processes of the same application. If a process fails, the other processes of the same application will also be terminated.

Profiling

Analyzing (parallel) applications to enable faster execution.

Proxy

Software running on computers, and managing the agents on the managed components on behalf of the centers. Behaves as agent toward centers and as a center toward agents.

PVM (Parallel Virtual Machine)

Standardized communication library that creates a virtual parallel machine. The communication between computers of various platforms is simplified, from Windows-based laptops up to supercomputers. The result is a heterogeneous supercomputer.

Queue

List of applications that are consequently being executed.

Rank

The Rank identifies the process within the same (parallel) application. If the size of the application is 16 nodes (16 *Processes* belong to the application), the processes receive individual rank numbers from 0 to 15. Every process knows its individual rank.

Redundancy

Redundancy means that a service or component that is mission-critical is backed by additional components that are activated in case of break-down. Redundancy is used to achieve high reliability and availability of services.

Reliability

Reliability means that a consumer can depend on the service because it always reacts in the same way and produces feedback in every case or demonstrates how the goods or services may be obtained in case of a system or component failure.

Reliable Cluster

Reliability mechanism, where the entities of the cluster solve a problem together. The cluster is allowed to operate as long as a quorum of all members is available (usually one more than half of them). The workload is distributed to all present members.

Resource Matrix

Matrix with two dimensions «processing element» and «time». Applications are rectangular blocks in this matrix. Parallel version of the Queue.

RISC (Reduced Instruction Set Computer)

Processor with a reduced set of instruction and addressing modes. The reduction allows simpler silicon implementations and higher clock rates, leading to higher performance.

SAN (System Area Network)

The System Area Network (SAN) interconnects the nodes of a supercluster and is used for fast communication between distributed processes on the nodes. The key features of the SAN are high bandwidth and low latency, combined with a rich set of functionality such as multicast and broadcast mechanisms. The commonly used technologies are Myricom MyriNet, SCS T-Net or Quadrics QNet.

Scalability

Scalability is the behavior of output values of a function when the input values are increased or decreased. A function where the output values change in, at most, a linear relationship to the input values is called «to scale well». Functions where the output values change in more than a linear relationship to the input values are called «not to scale».

Security

Security is the combination of Authentication, Encryption and Integrity. The combination of these three areas allows reliable communication between two entities.

SHMEM (Shared Memory)

Standardized communication library that allows shared-memory-like computing in multicomputer environments.

SIMD (Single Instruction [Stream], Multiple Data [Streams])

Architecture for processors or computers [Fly72]. The processing unit executes instructions from one instruction stream and operates on data from multiple data streams. This architecture is used in vector-processor architectures.

SISD (Single Instruction [Stream], Single Data [Stream])

Architecture for processors or computers [Fly72]. The processing unit executes instructions from one instruction stream and operates on data from one data stream. This architecture is used in microprocessor architectures.

SMP (Symmetrical Multiprocessing, Shared Memory Processing)

In SMP-based computers, all processors are connected to a shared memory. All processors can access this memory with the same time lapse. The connection between the processors and the memory is usually the bottleneck, since the available bandwidth is limited and divided by the number of processors.

SNMP (Simple Network Management Protocol)

The SNMP allows the network nodes to be managed using a simple de-facto standard created by Marshall T. Rose and Jeffrey D. Case. SNMP allows network nodes to be configured and enough freedom to be given to the network hardware manufacturers to implement the access only to the most vital parts or to all parts of the hardware. Besides SNMP, a *MIB* needs to be implemented which the SNMP has access to.

Speed-Up

Gain factor of an accelerated system (e.g. parallel computer) over the original system.

Static Routing

Routing algorithm, where only one port is defined for a destination. The routing algorithm must re-calculate the routing tables of all network routers and distribute those tables to all routers. Opposite: Adaptive routing.

Supercluster

Cluster-based supercomputer, commonly used for commodity supercomputers.

TCP/IP (Transmission Control Protocol / Internet Protocol)

The TCP/IP allows «reliable» transmission between two communicating partners. Connections are actively created and closed, data is transmitted using an acknowledged mechanism and timeouts allow the detection of failures. TCP/IP only allows the creation of point-to-point connections between an accepting server and a connecting client. TCP/IP connections are used for applications using standardized sockets interface.

TFLOPS

Short for Tera FLOPS (Floating Point Operations per Second), equal to one trillion (1 000 000 000 000 = 10^{12}) operations per second.

Top500

Ranking list of the fastest supercomputers of the world, updated twice a year in June and November during supercomputing conferences in Germany and the US [Top500].

Vector Processor

Processor that does not calculate on single («scalar») values, but on multiple values concurrently. These values are structured in arrays of identical data type, called «vectors».

Acknowledgements

I would like to express my sincere gratitude to my advisors, Prof. Dr. Anton Gunzinger and Prof. Dr. Bernhard Plattner for their support and inspiring discussions. Toni Gunzinger has made many interesting supercomputing projects possible and I would like to thank him for his exceptional support and confidence in me and in my work.

Special thanks go to my dear colleagues and co-workers of the supercomputing groups at ETHZ and SCS: Stephan Brauss, Martin Lienhard, Martin Näf, Martin Frey, Martin Heimlicher, Andreas Huber, Patrick Müller, Peter Severa and Martin Uehli. In particular the fruitful and motivating discussions and work with Stephan Brauss, Martin Lienhard and Martin Frey were very inspiring, and lead to lean and creative solutions.

I would also like to thank the Head of the Electronics Laboratory of ETHZ, Prof. Dr. Gerhard Tröster, who supported my work during the last phase and put enough pressure on me to effectively and efficiently bring this dissertation to a successful end.

My further thanks go to my proof-reader, Nicola Auf der Maur, who did an excellent job in translating this dissertation from Swiss English into British English. All remaining mistakes and errors were added after proofreading.

The «Swiss-Tx» project was funded by the *Swiss Commission for Technology and Innovation* (CTI) and I would like to express my gratitude for their support of supercomputing research projects.

Zurich in November 2005 Josef Nemecek

Curriculum Vitæ

Personal Information

Josef Nemecek
born March 31, 1973 in Wädenswil (Switzerland)
citizen of Wädenswil (Switzerland) and Czech Republic

Education

1997–2005 Ph. D. student at the Electronics Laboratory (IFE) of the Swiss Federal Institute of Technology in Zurich (ETHZ)

1992–1997 M. S. student in Computer Sciences at the Swiss Federal Institute of Technology in Zurich (ETHZ), subsidiary subject Business Economics and Marketing

Work

1997–2001 Research and teaching assistant in the High-Performance Computing Group at the Electronics Laboratory of the Swiss Federal Institute of Technology in Zurich (ETHZ)

1996 Internship at Supercomputing Systems AG in Zurich, developing the Windows NT driver for a custom FC-based NIC for the digital audio mixing console Studer D950.

Publications as responsible author or co-author

2000 «Specification of a Second-Generation Management Software for Superclusters»; for the High-Performance Computing Group at Supercomputing Systems Ltd., Switzerland.

2000 «Management of Superclusters», SOS Workshop presentation.

2000 «COSMOS – Management of the Swiss-T1»; Posters for the inauguration of the Swiss-T1 supercluster and Swiss-Tx final presentation; Internal report of the Electronics Laboratory of the ETH Zurich.

1999 «An Efficient Communication Architecture for Commodity Supercomputing»; International Conference of High-Performance Computing and Communications (SC99).

1999 «Computer-Aided Management of Commodity Supercomputers»; Institute colloquium of the Electronics Laboratory of the ETH Zurich.

1999 «Network Topologies, Theoretical Study of Common Topologies»; internal publication for the Swiss-Tx project management and COMPAQ Computer Corporation.

1998 «The Alpha 21264 Microprocessor»; Product presentation and outlook at the COMPAQ Technology forum in Switzerland.

1998 «Supercomputing using Windows NT»; Institute colloquium of the Electronics Laboratory of the ETH Zurich.

1998 «Supercomputing using Commodity Components»; Research report for lectures in technical computing at the ETH Zurich.

1998 «Supercomputing auf Workstation-Basis»; Informatik 3/1998 (publication of the Swiss computer science organizations), 3–6

1997 «MARS – Multiprocessor Architecture RISC-based Supercomputer»; Diploma thesis at the Computing Department of the ETH Zurich.

1996 «The Alpha 21164 Microprocessor»; Colloquium at the Computing Department of the ETH Zurich.

Developments

1999–2001 COSMOS – Management Software for the Swiss-T1 and other Superclusters; Available for Red Hat Linux and Compaq Tru64 UNIX on Intel and Alpha Processors. Design and Development.

1998–2000 FCI – Fast Communication Interface; Available for Linux, Compaq Tru64 UNIX and Windows NT on Intel and Alpha Processors. Co-Design (with Stephan Brauss) and Co-Development (with Stephan Brauss and Martin Frey) on Windows NT.

1998–2000 Windows NT Device Driver, Supporting and Testing Software for EasyNet and T-Net network interfaces on Windows NT. Co-Design and Co-Development (with Stephan Brauss, Martin Frey, Martin Heimlicher, Andreas Huber, Martin Lienhard and Patrick Müller)

1996 Windows NT Device Driver, Supporting and Testing Software for MemNet, the FC-based NIC of the digital mixing console Studer D950.