OPERA-G
A microkernel for computational grids

Author(s):
Bausch, Win

Publication Date:
2004

Permanent Link:

https://doi.org/10.3929/ethz-a-004687067 →

Diss. ETH No. 15395

OPERA-G : a microkernel for computational grids

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH

for the degree of
Doctor of Technical Sciences

presented by
WIN BAUSCH
Dipl. Informatikingenieur ETH
born May 21, 1974
citizen of Luxembourg

accepted on the recommendation of
Prof. Dr. Gustavo Alonso, examiner
Prof. Dr. Klaus Dittrich, co-examiner

2004

Kurzfassung

Während der letzten Jahre sind wir unentwegt Zeuge eines stetigen Anstiegs des Bedarfs an Rechenressourcen für die Lösung wissenschaftlicher Probleme geworden. Diese Entwicklung wurde nicht nur von traditionell auf grosse Berechnungen ausgerichteten Wissenschaftszweigen vorangetrieben, sondern auch zusätzlich von neu hervorgebrachten Disziplinen, wie z.B. die Genforschung. Der überproportionale Ansteig der benötigten Rechenleistung im Vergleich zu den verfügbaren Ressourcen sowie die zunehmende Interdisziplinarität vieler Forschungsbereiche waren der Grund für die Entwicklung des Grid-Konzeptes. Ähnlich wie das Web, das den mühelosen Zugriff auf eine Fülle von global verteilten Daten vom Schreibtisch aus ermöglicht, wird das Grid global verteilte Rechenressourcen bündeln und jedermann verfügbar machen. Heutzutage gibt es zwar bereits Infrastrukturen die global verteilte Ressourcen bündeln. Solche Infrastrukturen werden jedoch in der Regel im Rahmen eines Abkommens zwischen Forschungspartnern festgelegt und verändern sich während der Dauer eines Projekts nur selten. Das Grid verspricht Abhilfe in diesem Punkt indem es erlaubt sogenannte virtuelle Organisationen, die verteilt vorhandene Hardware, Software und Daten gemeinsam nutzen wollen, transparent und ohne langwierige Abklärungen. Die dafür benötigte Flexibilität sowie die globale Ausdehnung bewirken, dass ein Grid in höchstem Grade heterogen und dynamisch sein wird. Die zum Betrieb eines Grids benötigte Infrastruktur wird komplexer als alles bisher Dagewesene sein. Eines der primären Ziele ist es diese Komplexität wo immer möglich vor dem Benutzer zu kaschieren indem das Grid alle nötigen administrativen Aufgaben die mit der Ausführung einer Anwendung zusammenhängen selbst übernimmt. Der Benutzer sollte nur wenn nötig mit einer Anwendung interagieren müssen. Diese Dissertation untersucht das Potential von Prozessunterstützungssystemen als generische und flexible Basis für den Betrieb eines Grids in dieser Hinsicht.

Der erste Teil der Dissertation charakterisiert die untersuchte Klasse von verteilten Anwendungen anhand einiger typischer Beispiele. Zugleich führen wir eine Reihe von Problemen auf, mit denen man bei der Ausführung solcher Anwendungen in einem Grid rechnen kann. Basierend auf diesen Erfahrungen beschreiben wir die an einen generischen Kernel gestellten Anforderungen. Im folgenden stellen wir zuerst das unserer Plattform zugrunde liegende Programmiermodell vor. Dieses Modell basiert auf der von Prozessunterstützungssystem her bekannten Idee der Prozesse. Wir zeigen auf, wie dieses Programmiermodell in einem Grid zur Anwendung kommen kann und wie es uns erlaubt, den Programmierer bereits zur Entwicklungsszeit eine relativ einfach verständliche und benutzbare Plattform zu bieten. Das Hauptmerkmal hier ist, dass unser Programmiermodell den Entwicklungsprozess einer verteilten Anwendung auf einer sehr hohen Abstraktionsebene unterstützt. Somit kann dieses Modell als Ergänzung bestehender Programmiertechniken betrachtet werden. In der Folge stellen wir eine Architektur für die zuverlässige Ausführung prozessbasierter Anwendungen vor. Ein bedeutendes Element dieser Architektur ist ein Datenmodell in dem Metadaten über die Anwendung gespeichert werden können und was dazu dient Berechnungen sowohl bezüglich der

generierten Resultate als auch im Hinblick auf historische Vorgänge zu interpretieren. Ein anderer zentraler Bestandteil ist die verlässliche Ausführung von Berechnungen sowohl auf der Anwendungs- als auch auf der Plattformebene. Zuverlässigkeit auf der Anwendungsebene wird durch eine Reihe von Fehlerbehandlungsmechanismen auf der Prozessebene garantiert. Auf der Plattformebene wird Fehlertoleranz durch die persistente Speicherung des Applikationszustandes in Kombination mit einem Recovery Mechanismus hergestellt. Allein diese Mechanismen erlauben es bereits robuste verteilte Anwendungen für ein Grid zu entwerfen. Wichtiger aber ist der Umstand, dass diese Mechanismen es uns erlauben einen Schritt weiter zu gehen und autonome Grid Plattformen zu entwerfen. Solche Plattformen passen sich automatisch den gegebenen Bedingungen innerhalb eines Grids an und verändern Anwendungen automatisch indem sie z.B. neue Rechenressourcen ausfindig machen wenn ein Engpass ermittelt wurde, resp. auf alternative Ressourcen umsteigen falls nötig. Solche Fähigkeiten sind auch zentral für die Grid Plattform selbst, da auch sie veränderten Bedingungen ausgesetzt sein kann. Die Plattform sollte wenn möglich solche Situationen ohne Mithilfe eines menschlichen Operators erkennen, z.B. durch Analyse der vorhandenen Metadaten, und die nötigen Gegenmassnahmen treffen können. Unser System wurde als Prototyp implementiert und hat als Kernkomponente innerhalb grösserer Syteme Verwendung gefunden, die in der Dissertation kurz besprochen werden. Der finale Teil der Dissertation widmet sich eingehend einer Reihe von Experimenten die die Fähigkeiten der Plattform illustrieren.

Insgesamt hat diese Dissertation ein Modell, eine Architektur sowie einen Prototypen zur Modellierung und Ausführung verteilter Berechnungen in einer unzuverlässigen Rechenumgebung hervorgebracht. Die Fähigkeiten des Systems erlauben es dem Endbenutzer sich auf den Kern des zu lösenden Problems zu konzentrieren während Aufgaben wie Buchführung über eine Berechnung, effizientes Scheduling, Lastbalancierung verfügbarer Ressourcen, Neuzuordnung von Ressourcen zur Laufzeit, Zugriff auf intermediäre Daten während der Berechnung und Wiederherstellung des Kontextes einer Berechnung bei Rechnerausfällen automatisch ausgeführt werden. Alle diese Fähigkeiten erleichtern es, die Komplexität eines Grids zum Vorteil des Endbenutzers zu maskieren.

Abstract

In the last few years we have witnessed an increasing need for computing power to solve all kinds of scientific problems. This evolution has not only been driven by traditional areas where massive computations are a daily business, but also by the emergence of new fields of application for computers, e.g., genomics. The ever growing need for CPU cycles by ever diversifying user communities gave rise to the idea of the Grid. Similar to the Web offering access to a vast universe of information easily accessible from anybody's desktop, the Grid will allow computational resources to be shared on a global scale. Whereas such sharing already takes place today within well defined and relatively static boundaries that have been determined up front as part of collaboration agreements between partner organizations, the Grid promises to enable the formation so-called virtual organizations that share data, hardware and software in a transparent and straightforward manner. Entities interested in cooperation will ultimately be able to define and adapt the sharing relationships underlying a virtual organization on-the-fly without effort. Due to this flexibility as well as its global scale, the Grid will be a highly dynamic environment. The computational infrastructure to enable this type of distributed computing will be one of the most complex ever built. It follows that one of the prime issues is to effectively hide this complexity from the user and let the Grid take over many of the administrative chores. Users should be able to naturally interact with running applications. In this dissertation we investigate the potential of the notion of *process support system* to build a generic flexible kernel that can be used as foundation for building dependable Grid Computing Environments (GCE).

Throughout the first part of the dissertation we characterize the type of Grid application scenarios we are targeting and account for a variety of problems that may occur when running such applications in a Grid. From these experiences we derive the requirements for a GCE. In what follows, we develop a high-level programming model for Grid applications based on the notion of process as found in process support systems. We show how this programming model lets the programmer concentrate on solving the domain-specific problem at hand and abstracts from the runtime aspects of the application, thus reducing complexity at development time. Our programming model is situated at a high abstraction level and can be seen as complement to traditional programming techniques. Following this, we are proposing an architecture for a GCE that executes processes in a dependable manner. One basic ingredient provided by this architecture is an application metamodel that describes application metadata and can be used to analyze the outcome of a computation with respect to the results that were computed as well as concerning historical aspects. Another basic ingredient is reliable execution both at the application and at the GCE level. We achieve reliability by providing several levels of failure handling on the application level as well as by guaranteeing fault tolerance of the GCE through the use of persistent storage combined with a recovery scheme that allows to resume processing after a failure. While these features by themselves already allow to build robust Grid applications, they above all enable us to go a step further and provide adaptive and

autonomic capabilities. Using these capabilities we can reconfigure active computations by adding, removing or modifying Grid resources to account for changing conditions in the Grid. The same holds for the GCE itself. Autonomic adaptation of the GCE may be necessary in order to guarantee stable operation in response to changing external conditions. Adaptation may happen based on information extracted from the metamodel, e.g., historical data concerning the execution times of past computations. Following the idea of a kernel our system is modular and allows to be extended and customized for specific application scenarios. The architecture proposed in the dissertation has been implemented in a fully functional prototype that has been tailored to different application scenarios, as will be briefly described. The final part of the dissertation will illustrate and qualify the capabilities of our prototype through a set of experiments. All in all, the kernel architecture presented in this dissertation provides a range of basic services that many Grid applications can benefit from. These services include relieving the user from standard bookkeeping tasks, transparently handling issues such as efficient scheduling of jobs, load balancing of available resources, dynamic updating of resource state and configuration, tracking the progress of the computation, recovering from system errors and machine faults, accessing intermediate results as they are computed, automatically aggregating data concerning execution times, and a systematically organizing both intermediate and final results of a computation. Overall, the kernel's functionality enables to build dependable flexibly configurable GCEs that encapsulate the complexity of the Grid environment and provide end-users with an appropriate application-specific interface to the Grid.