Backup and process migration mechanisms in process support systems

Backup and Process Migration Mechanisms in Process Support Systems[*]

Technical Report No. 304
ETH Zürich, Institute of Information Systems

Claus Hagen **Gustavo Alonso**
Information and Communication Systems Research Group
Institute of Information Systems, Swiss Federal Institute of Technology (ETH)
ETH Zentrum, CH-8092 Zürich, Switzerland
{hagen,alonso}@inf.ethz.ch

August, 1998

Abstract

Availability in Process Support Systems (PSS) can be achieved by using standby mechanisms that allow a backup server to take over in case a primary server fails. These mechanisms, resembling the *process pair* approach used in operating systems, require the primary to send information about state changes to the backup on a regular basis. In PSS where all relevant state information is stored in a database, there are two principal strategies for synchronizing a primary–backup pair. One is to use the replication mechanisms provided by the DBMS. Another is to implement a message mechanism to exchange state information between servers above the database level. For both approaches, several variants exist that allow to trade run-time performance for failover time. This paper discusses the possible strategies and evaluates their performance based on an implementation within the OPERA process support kernel. Moreover, it is shown how the mechanisms can be used as the basis for implementing process migration in a distributed setting.

1 Introduction

Process support systems (PSS) are metaprogramming environments that allow the specification, execution, and monitoring of complex sequences of application invocations (*processes*) in a distributed environment composed of heterogeneous, autonomous applications (*composite* system). PSS are increasingly gaining importance as integral building blocks of corporate IT infrastructures, software development environments, and scientific computing since they provide the services necessary to integrate distributed, heterogeneous programs into coherent applications. Today, PSS are used for tasks such as supporting enterprises in streamlining and automating their business processes [GHS95, LA94], supporting health care organizations in coordinating immunization campaigns [SKM^{+}96], helping scientists perform and analyze experiments [MVW96, BSR96, AH97], and allowing software development teams organize joint work on shared resources [BK94, TKP94]. With the proliferation of PSS technology, the requirements in terms of availability increase but, unfortunately, current commercial systems do not provide satisfactory solutions in this area [AAAM97, GHS95, KR96, AM97]. A possible solution to this problem is to use standby techniques similar to database backup techniques [GR93] or *process pair* concepts in operating systems [GR93, Bar81]. In practice, since the PSS is built on top of a database, the backup mechanisms can either be based on the

[*]Part of this work has been funded by the Swiss National Science Foundation under the project TRAMs (Transactions and Active Database Mechanisms for Workflow Management).

replication capabilities of the underlying DBMS, or implemented at the application level using semantic knowledge of the application. The latter has been suggested as the best approach [KGAM96] due to its added flexibility but, so far, no effort has been made to evaluate these different strategies and check their applicability in a real setting.

This paper presents a comprehensive study of backup mechanisms in the context of process support systems. Several algorithms are proposed and their performance and effectiveness studied in detail within the OPERA kernel [AHST97, AH97, HA98]. In particular, we show that backup can be performed at a high level, without having to rely on the database idiosyncrasies. This approach has significant advantages. First, it allows to use different databases as primary and backup (in the current implementation, Oracle and ObjectStore are both used as backup for each other), which was suggested in [KGAM96] as one of the advantages of semantic backup, although without any empirical evidence. To some degree this argument is losing strength, given that there are now products on the market that allow to replicate data between heterogeneous DBMS [Kno97, IBM97]. These products do not yet provide, however, the synchronous replication needed for 2-safe backup strategies. In this regard, the solution proposed here could be seen as a way to circumvent the limitations of the state of the art in current products. Second, the performance overhead of high level or semantic backup has been minimized through a careful and in depth optimization of the algorithms used, as well as through a well tuned implementation. As the results presented in here show, the overhead introduced by our solutions is comparable to that incurred by a commercial database. Third, and perhaps more interestingly, we show how these same backup mechanisms can be used as the basis for a sophisticated process migration mechanism that is both transparent and efficient. We consider that an important contribution of the paper is the implementation effort and the insights gained from it in terms of real system experience. We expect these results and the discussion in the paper to be of significant use to anybody interested in developing a solution for semantic backup or process migration in distributed systems.

The paper is organized as follows. Section 2 surveys the possible backup strategies that could be adopted in the context of a PSS. Section 3 describes the solution proposed. Section 4 discusses how to use the backup solution to implement process migration. Section 5 discusses the results of our analysis and implementation aspects. Section 6 concludes the paper.

2 Backup strategies

A number of techniques exist that could be used in the context of a PSS. In this section, these techniques are briefly discussed and their advantages and shortcomings, from the point of view of a PSS, analyzed in some detail. We will base our discussion on a very general system model that suits most existing systems. Consider a PSS as a set of *servers* acting as interpreters for metaprograms (*process models*). There is a one-to-many relation between servers and process instances, i.e., each instance is assigned to one server (which may change in time) that stores the process state persistently in form of a *process image* containing all relevant state information. The key to PSS backup mechanisms is thus the redundant storage of the process images on failure-independent sites. More detailed coverage of general PSS concepts can be found in the literature [AHST97, AH97, GHS95, Hsu95].

2.1 Low level backup mechanisms

Backup mechanisms ensure continuous availability and can be found in most database management systems as well as in specialized operating systems like Tandem's Guardian System [Bar81]. As an example of the basic idea, Guardian is based on *process pairs*, with one process (*primary*) performing the actual work, and the other (*backup*) serving as a standby that takes over should the primary become unavailable. This mechanism requires a dedicated backup and the notification of the backup of all significant changes in the primary's state in order to guarantee *exactly once* semantics. Although the techniques used in this paper are similar in nature, we differ from this approach in that we do not require a dedicated backup. In fact, we have made great efforts to minimize the impact of the backup mechanism on the secondary site in order to be able to use it as well as primary server for other processes. We envision a system configuration in which a PSS server offers backup services as part of its normal functionality and can perform such services concurrently with those of a normal PSS server. For cost and feasibility reasons, we do not want to resort to dedicated backup solutions like those used in operating systems or hardware architectures.

Databases use similar ideas to implement backup strategies. The discussion is often centered not so much on the unit of backup (often changes on a per transaction basis) but on the level of consistency to be achieved. This is typical of the constant trade-off in databases between consistency and performance. Thus, in databases, several approaches are used: hot-standby (the backup can take over immediately), cold-standby (the backup needs to be initialized and updated before it can take over), 1-safe (some changes may be lost when failures occur since the backup is not necessarily completely up to date), and 2-safe (no changes are lost when failures occur since the backup and the primary are synchronized by means of a 2 Phase Commit protocol) [GR93]. The main problem with database backup tools is that they are, necessarily, database specific. These tools are not conceived as open systems and do not allow to use different DBMSs in the primary and backup. Moreover, these tools leave almost no room for controlling the backup process. We have put a significant effort in making the backup mechanism of a PSS user controlled in order to leave to the user the decision of when a backup is necessary. Similarly, database backup tools would not have allowed to develop the process migration mechanism we provide as an extension to the backup architecture.

2.2 Database replication as the basis for backup mechanisms

Backup and replication techniques are fairly similar. In principle, it is possible to use replication techniques to implement a backup strategy. In practice, the state of the art in database replication does not really allow it. Most DBMS vendors implement *primary-copy*, *asynchronous* replication where only one replica of a data item is updatable while the others are read-only and not necessarily up to date at all times [GHOS96]. This approach could be used to implement 1-safe strategies, but this is very much depending on the way the DBMS implements replication. With a DBMS using a push model, like Sybase [Kno97], this could be a possibility. There, the server propagates changes soon after they have occured. With a DBMS implementing a pull model, like IBM Data Replicator [IBM97], it is not realistic to use replication as the basis for the backup mechanism since here the primary server is polled by the backup site which has no knowledge of when changes occur. In addition, these replication mechanisms, being designed primarily for warehousing applications, are geared towards a homogeneous environment where the copies are all managed by the same type of DBMS and would force us, in most cases, to have a dedicated backup. From a practical standpoint, and to our knowledge, only Oracle's *Advanced (Symmetric) Replication* mechanisms [Ora95, Ora97] provide enough flexibility to use replication as the basis for a backup. Oracle8 Server offers various replication techniques, of which only one is really suited to the purposes of backup: Synchronous multimaster replication, where all copies of a data item are fully updatable and changes are propagated to all sites within the same transaction. Multimaster replication is based on *master groups* that are sets of base tables. Once a master group is defined (using PL/SQL or a graphical user interface), it can be replicated on multiple Oracle databases by generating special PL/SQL packages (modules containing stored procedures) on all databases. For each replicated table, two packages are generated that contain procedures for change propagation and for conflict resolution (if asynchronous replication is used). The replication mechanisms are tuple-based. Every time a row of a replicated table is changed, a trigger invokes the appropriate procedure in the table's propagation package. In contrast to Oracle7, Oracle8 uses internal triggers that are not part of the DML and are optimized for performance.

Process backup is implemented over Oracle's multimaster replication by defining a master group that contains all the tables for the process images. The main drawback of this replication scheme is that in practice, it is very inflexible. Once one of the replication sites becomes unavailable, the remaining DBMS allow no updates to the replicated tables until the failure is repaired or the replication scheme changed. This requires removing the failed database from the set of replication sites and adding a new site that acts as new backup. The process of restructuring the replication scheme is complex since the PL/SQL packages have to be changed and the new site has to be synchronized by sending all data. This procedure consumes a considerable amount of time. In practice, using the replication mechanisms of a database for back-up can thus only be used for cold stand-by approaches that tolerate a significant delay until execution can be resumed. Hot stand-by configurations with their demand for quick take-over do not seem to be feasible in this environment. These problems are not particular to the way Oracle implements replication, but are typical of commercial replication techniques if used as the basis for backup strategies.

2.3 Semantic backup

In the same way that it is possible to implement replication relying on semantic information (e.g., reconciliation techniques [Ora95, Ora97], although in this case semantic information is used to resolve inconsistencies), a backup mechanism can also use semantic information about the application in order to reduce overhead. Briefly described, the approach is based on synchronizing the primary and backup at the end of each transaction. All changes to the state of a process are stored in a *change buffer*, which is sent from the primary to the backup. The transaction can commit when the changes have been applied successfully on both sites ("to apply" has different meanings, depending on the strategy used). Key to this approach is the ability to represent the changes performed at the primary in a concise way. This requires to represent the process state in a format that is independent of the database representation. This format is used as the semantic information that allows to implement the backup mechanism at a higher level, bypassing the underlying database. This idea was suggested as a theoretical possibility in [KGAM96] but, to our knowledge, this approach has never been implemented in practice. From the descriptions provided in [KGAM96], it is not clear whether the approach performs better than standard backup techniques or those based on database replication mechanisms. In addition, only failures are considered and not scheduled outages for maintenance and system configuration changes. In many environments, the latter can be much more frequent than the former and the backup strategy is incomplete if it can not cope with such situations. Finally, the ideas in [KGAM96] are presented in the context of FlowMark, a workflow management system of IBM, and may not apply to generic process support systems.

3 A flexible backup strategy

The solutions we propose follow the ideas outlined in [KGAM96] although with many significant changes. Among others, the solution is generalized to a PSS instead of reducing its applicability to a workflow management system and a number of important changes and optimizations in the algorithms were necessary to make them feasible in practice.

3.1 Backup in a Process Support Systems

The functionality of a PSS revolves, to a great extent, around the way processes are defined and represented in the system. After the designer has defined the process using a graphical language, the process is translated into an internal representation more suitable for execution. This internal representation can be further processed to produce a third format used to store the process persistently in a database. For our purposes here, it is the internal representation, the one used for execution, that is of interest. The details of the internal representation are not relevant for the discussion of the backup subsystem and are omitted here. Interested readers can find more information in [AHST97, AH97]. The internal representation summarizes the state of a process: which tasks have been executed, data values passed to and returned from tasks, and additional information like the person that executed a task or the site that was chosen to execute a program. Every update in the process state takes place through a transaction, which executes what is usually referred to as a *navigation step* in the process.

Semantic backup is performed by exchanging between primary and secondary information casted in terms of the internal representation (instead of pages, or database transactions). In our implementation, at the end of each navigation step and as part of the same transaction, all changes to the process state are saved in a *change buffer* at the primary and sent to the backup server for further treatment. Obviously, the change buffers play a crucial role in the performance of the backup system since they are exchanged at the end of each navigation step and stored at both the primary and backup. In the algorithms of [KGAM96], the change buffers are stored in persistent hash tables on both the primary and the backup. This adds significant, unnecessary overhead in practice. In our implementation, the change buffers are kept directly in the database, thereby greatly simplifying the backup mechanism. Our experiments revealed that storing the change buffers in the databases requires only between 40 and 80 ms per buffer, depending on the database size. In order to speed up the time needed to store and retrieve the buffers, a separate tablespace with a very large extent size (1 GB) was used, thereby reducing the overhead of frequently allocating new extents. The change buffers were stored as character arrays with a maximum size of 4 KB. The average size of the change buffers in our experiments was 300 Byte (significantly smaller than the information exchanged by database replication strategies).

In a PSS environment, only 2-safe techniques make sense [KGAM96]. Information lost when a secondary takes over may lead to having to repeat activities already completed, which is not only time consuming but also confusing and may lead to inconsistencies. This implies that storing the change buffer persistently at the primary and backup must be done in a synchronous manner using 2 Phase Commit (2PC) [BHG87]. To alleviate the cost, it is possible to minimize the time it takes the secondary to reply by storing the buffer, but not actually performing the changes. This leads to two possible strategies: a hot-standby and a cold-standby backup. Thus, as in [KGAM96], we define three types of processes: *critical*, *important* and *normal*, depending on whether they use a hot-standby, a cold-standby, or no backup at all. The different availability levels allow to trade runtime overhead for recovery cost. Critical processes can proceed immediately after the takeover of the backup, but the synchronous update of the database increases their navigation cost at run-time. Important processes, on the other hand, have a reduced overhead during normal operation, but they have to be installed in the database during takeover, which leads to a longer recovery phase.

Since 2-safe is used, the backup mechanisms we propose could suffer from the high overhead of 2PC. We can alleviate this cost by taking advantage of the fact that only two sites are participating in each distributed transaction. We use a "degenerated" form of linear two-phase commit [BHG87] in which the primary sends the change buffer to the backup, the backup commits locally and then sends an acknowledgment back to the primary, which commits. The decision for the commit of the distributed transaction is taken at the moment the backup transaction commits. Correctness is guaranteed since if the primary fails after the backup's decision, but before its own commit, the backup will take over navigation anyway and continue with the valid process state. If the backup does not take over (this can be the case when the down-time of the primary is too short), the primary has to inquire about the process state during recovery. To this end, each server uses a *reconfiguration protocol* at startup during which it determines the actual state of all processes that are registered in its database by querying the backup servers. If a backup server has taken over a process, it can be removed from the old primary's database since the new primary has elected a new backup during fail-over. If the backup has not yet taken over, it simply sends the actual state of the process to the primary which updates the process image and continues navigation.

To increase the performance of the backup system even further, it is possible to execute the navigation steps partially in parallel. We do so by using multiple threads to speed up the processing of navigation steps. Figure 1 shows the structure of a transaction as it occurs when the hot standby mechanism is used. First, the process image is fetched from the database. It is cached in main memory for the duration of the transaction. Navigation takes then place over the cached information, which will also act as the change buffer. After navigation is completed, the transaction forks into two threads. The first thread applies the changes to the process image at the primary, while the second sends the change buffer to the backup. At the backup a transaction is started that applies the change buffer to the process image. After its commit an acknowledgment is sent to the primary, and then the primary commits.

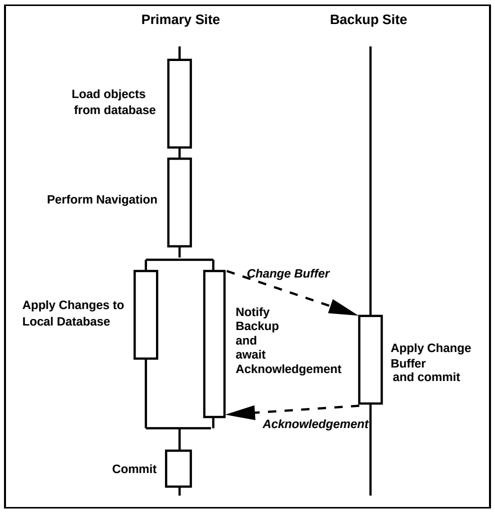

Primary Site
Backup Site
Load objects from database
Perform Navigation
Change Buffer
Apply Changes to Local Database
Notify Backup and await Acknowledgement
Apply Change Buffer and commit
Acknowledgement
Commit

Figure 1: Structure of a distributed transaction (hot standby)

3.2 Alternative backup strategies

For comparison purposes as an alternative to our backup strategies, we have implemented a solution based on Oracle's multimaster replication. This is done by defining a master group that contains all the tables for the process images. This data is then replicated between two databases, where each one is used by one server. The process engines operated on the two databases, and changes of a process image were immediately propagated to the other site by the replication mechanisms. Note that with this scheme, there is no notion of a private database per server. Because of this, a flag has to be stored with each process image to indicate which server is its primary. As pointed out before, this mechanism is only suitable for implementing cold standby strategies. We use it, nevertheless, as a reference to measure the overhead of doing the backup outside the context of the database.

4 Process migration

The mechanisms described above can be used not only to achieve continuous availability but also as building blocks for implementing *process migration*. Such functionality can be of great help to solve scalability problems and balancing the load on a large PSS [AAAM97]. The backup mechanisms proposed so far help only in the event of sudden server failures and only for important and critical processes. New problems arise, however, if a server needs to be shut down deliberately, either for maintenance reasons or because of system reconfigurations, and the processes were classified as normal. It was to address this issue that we initially implemented process migration. Later on the idea was generalized to all process types.

The basic idea behind our process migration solution is to use the same information the backup system uses. For simplicity, we assume the process belongs to the category of normal processes. If this is not the case, some of the steps described below are not necessary since they are already performed by the backup system. The migration algorithm (Figure 2) is based on shortly activating the backup of a process by upgrading it to critical. Since this implies that, via the backup system, the process image is automatically copied from the current server to the target server, it is then possible to switch primary and backup and afterwards degradate the process to *normal* state again. The result is that the process now runs on the target server. Applying this algorithm to all processes running on a given server allows to shut it down without affecting any ongoing activities. Note that it is possible to select the new server on a per-process basis, which allows to distribute the load of a terminating server evenly to the remaining ones.

The advantage of this scheme is that it allows the safe migration of processes without the need for addi-

Current server:		**Target server:**
Switch P to critical mode	*(Start process migration)*	
Create change buffer for P		
Select a target server		
Send buffer to target		Receive change buffer
		Create process P in instance space
		Install change buffer for P
	(Take over)	
P is deactivated		P is activated
	(P back to normal*)*	
Garbage collection on P		Resume execution of P

Figure 2: Migrating a process, P, from a server to another (target) server.

tional hardware or software components. The crucial point of server migration is ensuring the atomicity of the migration process, so that neither a process gets lost nor ends up being stored on two servers. Achieving this in an environment without backup mechanisms would require the deployment of transactional middleware such as persistent queues or TP-Monitors [BN97]. Thus, the advantages of our approach are threefold: First, we preserve platform independence since we do not have to rely on DBMS providing middleware interfaces (e.g., the XA interface needed for interaction with most TP monitors) – many object-oriented DBMS, for example, do not provide these mechanisms. Second, system complexity is kept bounded (we do not need yet another middleware component that has to be bought, installed, and maintained), which fosters the portability of a PSS and simplifies new installations. Finally, and perhaps most important, the solution is performant. As the experimental results given in section 5 reveal, backup mechanisms provide a suitable base for fast process migration which allows to use it even for the transport of large process sets.

Server migration is initiated by a special PSS component, the *planner*, that is responsible for controlling and changing the system configuration. To this end, it communicates with the servers in the system in order to gather information about their current load. The planner can then intitiate a reconfiguration of the system, either automatically (if certain tresholds for system and server load are reached) or if a reconfiguration is triggered from a system administrator, which may be the case if a server has to be disconnected from the system. The planner is designed to react to two situations of interest: Low load, in which case single servers can be stopped (and their machines be used for other tasks), and server overload, in which case new servers have to be started and existing processes migrated to them. The planner uses a configuration database in order to store information about the system structure and the load of the various servers. Note that for availability reasons, the planner can be implemented as a distributed component with a replicated configuration database. This is, however, not mandatory since a failure of the planner has no direct impact on the functionality of the system. In case of a failure, it is possible to start a new planner with an empty configuration database that is then populated by querying all servers during the initialization phase.

Using the planner, it is possible to design a system that dynamically adapts to varying loads of the environment. Server machines can then be used for other, low priority jobs whenever they are not needed for the PSS. In the case of a server shutdown, the planner selects a set of available servers out of the remaining ones (based on the load information it has collected) and then initiates the migration of all the processes to the new servers by sending a `migrate_all` message. This same procedure is initiated also on explicit demand of a system administrator, when a server has to be stopped for maintenance. Administrators use the planner to monitor and change a distributed PSS environment.

5 Experimental results

In this section we discuss the implementation of the algorithms in terms of both technical details regarding the concrete design decisions and performance results.

5.1 System Configuration

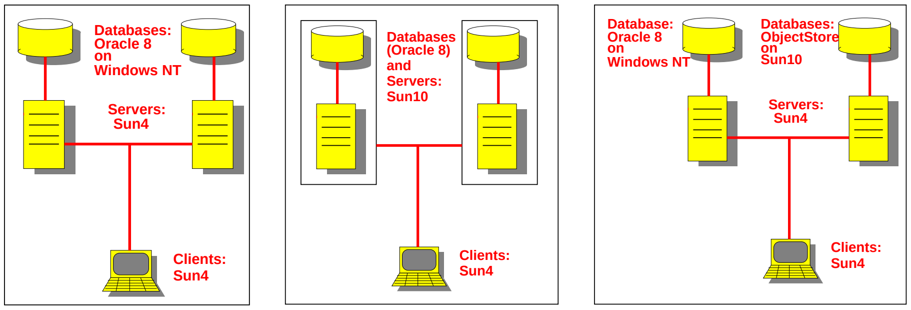


Figure 3: Configurations used in the experiments

The configurations used, a cluster of workstations of different types connected by a switched Ethernet, resemble many real environments. The concrete machines involved in the experiment are listed in Table 4, and an overview of the configurations used is given in Figure 3. In most experiments we used configuration A, in which the Sun4 workstations were running the workflow engines and the PCs were used as database servers running Oracle8 DBMS. Servers and clients were located on Sun4 workstations in the standard experiments. In order to determine the impact of communication overhead, in some experiments we used configuration B, where database and server are located on the same workstation. Here we used the Sun10 workstations for databases (Oracle8) and servers. Finally, for the evaluation of mixed environments consisting of relational and object-oriented databases, we used configuration C with an ObjectStore database located on a Sparc10 server, and Oracle8 on a PC.

For all experiments we used the same process model consisting of 10 activities, and with a process image size of about 50 KB. For performance reasons, the clustering facilities of Oracle8 were used in order to group together related tuples. All significant tables were grouped in one hashed cluster with the instance ID as cluster key, ensuring that the components of each process image are stored next to each other.

5.2 Base line results: no backup

As base line, we use the performance measurements for normal processes using configuration A. To study the impact of database size on navigation performance, the measurements were made with a varying number of concurrent process instances which resulted in different database sizes. The process images had an average size of 50 KB, which lead to database sizes between 500 KB and 50 MB. Figure 5 gives the average execution time for various types of transactions occuring during navigation. We distinguish between three transaction types used in the servers: *Process Instantiation (INST)* involves creating a new copy of a process template and identifying those tasks that are immediately executable. *Activity start (START)* is called whenever the server is notified that an activity has started. It is the simplest transaction type, since it only requires the change of the activity's state in the process image. *Activity termination (TERM)* comprises updating the terminating activity's state and performing navigation (evaluating the metaprogram in order to identify the activities that have become executable), and updating the states of the executable activities. The results show that, above a certain database size, the time per transaction stays almost constant. Databases with sizes below this threshold fit in main memory, thus the reduced execution time.

5.3 Backup based on DBMS replication

The backup mechanism implemented on top of Oracle's replication, as explained above, lead to an overhead, with respect to the base line, of approximately 200 % for process instantiation (because new database

Machine type	Processor	Main memory	disk space
SPARCstation 10	Model 30 SuperSparc	96 MB	322 MB
SPARCstation 4	Sparc 60 Mhz	64 MB	667 MB
Dell GXM 166	Intel Pentium 166 Mhz	32 MB	80MB

Figure 4: The machine types used in the experiments

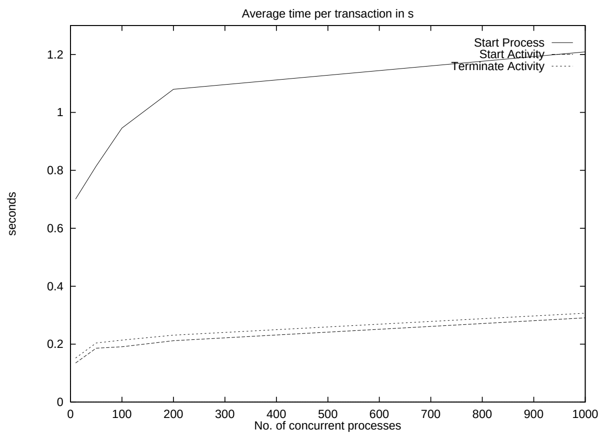


Figure 5: Time per transaction without backup

objects have to be inserted in the remote database) and an encouraging 30 % for the much more frequent navigation transactions (figure Figure 6). These response times are acceptable, especially if we take into account the absolute numbers, which stay below 4 seconds for process instantiation and take less than 0.2 seconds for navigation. A strange behaviour was observed, however, in the case of process termination, where the process image has to be removed from the database. The cost of removing one instance grew constantly with the database size, with an average duration of 15 minutes when 1000 concurrent processes existed. The reason for this can partly be explained by Oracle's row-based change propagation, where each changed tuple is propagated separately to the remote sites. This does not, however, explain the reason for the significant overhead. We believe that this is an implementation bug that will be solved in future versions of the DBMS (we used the very early version 8.0.3). This particular point aside, we believe the other results to be representative. From a practical stand point, the current behaviour creates significant problems in real scenarios. Process support systems may execute a large number of processes. Although current systems tend to keep the data for completed processes in the database, the trend is to remove these processes from the on-line database and store them in a data warehouse for analysis. This is one of the reasons why we use separate storage spaces [AHST97]. In particular, the history space plays the role of data warehouse for completed processes so that this data can be manipulated and pre-processed without interfering with the on-line operations of processes being executed. In our system, we automatically remove a terminated process from the instance space into the history space by copying it (if change buffers are available, it will use them to avoid having to access the primary) and then deleting it from the instance space. Obviously, this operation can become quite expensive if the problem persists.

5.4 Semantic replication, cold standby

To mirror actual working conditions, all experiments were conducted with all servers acting both as primary and backup, with processes evenly distributed between the servers. In cold standby configurations, the biggest source of overhead is sending the change buffer to the backup and wait until it is persistently stored in the database. Figure 7 shows the performance of process instance replication for cold standby mode. The results show that the cold standby mechanism leads to an overhead, with respect to the base line, that is

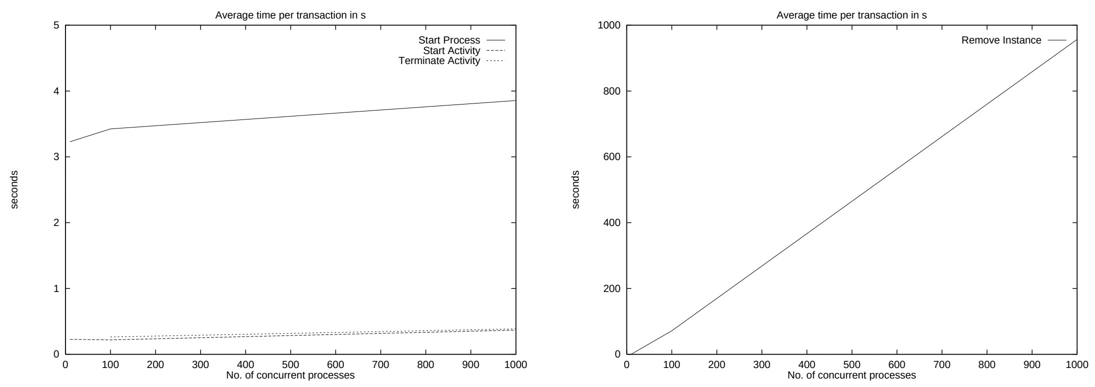


Figure 6: Performance of backup based on Oracle's replication mechanisms.

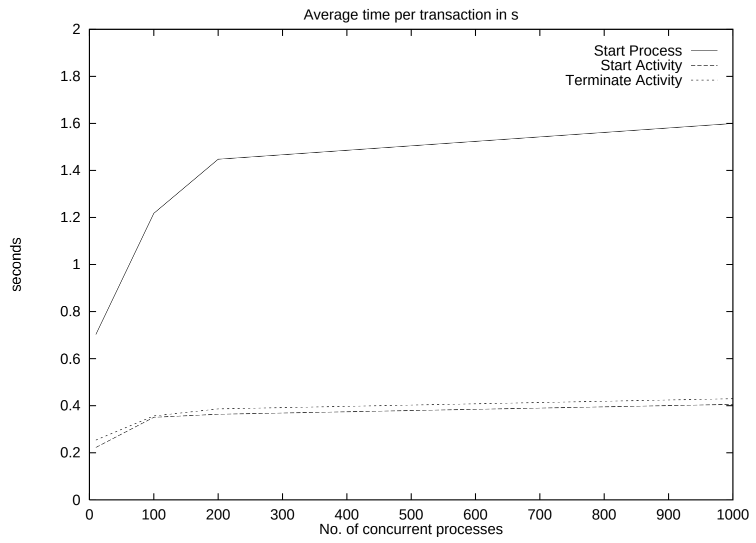
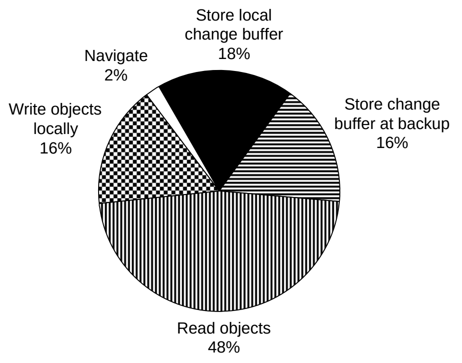


Figure 7: The performance of cold standby using semantic replication

around 65 % for small databases and falls to 40 % if the database size grows. Although this overhead may seem comparatively large, in practice it is almost negligible since it represents an overhead of between 0.5 and 0.3 seconds for starting a process and less than 0.3 seconds for the other two operations. In systems where there is human interaction involved, this delay will be easily masked by artifacts such as the terminal response time. For scientific applications this overhead is more significant but still within acceptable bounds, especially when considering the advantages of having a backup strategy for both fault-tolerance and process migration.

The differences between small and large databases arise from the fact that locating the process images at the primary gets increasingly expensive as the database grows while the cost for storing the change buffers stays constant. To analyze in more detail which operations are performed and their contribution to the overall overhead, Figure 8.a shows the relative cost of the operations involved in the transaction for a large database with 1000 processes. In the case of cold standby, reading the object takes as much time as in the non-replicated case. The overhead is created by the operations related to the change buffer. This implies that with a more efficient mechanism for change buffer storage the cost per transaction can be further reduced. It is questionable, however, whether the performance gain warrants the added complexity. We are currently exploring this topic in more detail but, given the results shown, cold standby, as currently implemented, seems to be a viable backup solution.



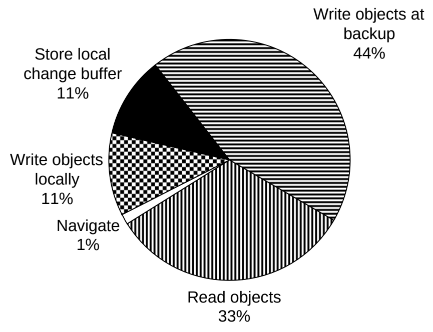


Figure 8: Cost analysis for a Terminate Activity step and 1000 concurrent processes: (a) cold standby, (b) hot standby

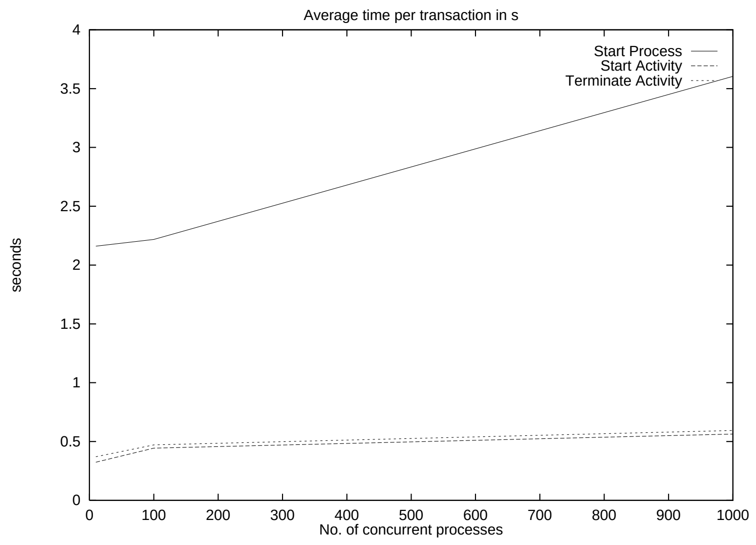


Figure 9: The performance of hot standby using semantic replication

5.5 Semantic replication, hot standby

The performance measures for the hot standby mechanism are shown in Figure 9. Due to the fact that both sites need to be synchronized and the change buffer needs to be applied at the backup, the overhead is larger than in the cold standby approach. It is not, however, significantly larger: for starting and terminating activities it is around 90% for the large database. Again, a comparatively large figure, but still acceptable since the actual time required is about 0.6 seconds. As before, this lapse of time is easily masked in many applications because there are additional operations like the communication with the clients, actualization of graphical user interfaces, transport of data between machines, and start of applications, that are quite expensive and are likely to be the real source of delays in practice. The contribution of individual operations to the overall overhead is shown in Figure 8.b. Note that even with the optimization of applying the changes at the backup in parallel, there is still the overhead of storing change buffers twice. In the case of starting a process, the overhead is significant and it does not seem to be possible to reduce it further. Since in a real application it is the user who determines which processes are critical and which important, the overhead can be assumed as part of the cost of having a hot standby configuration, but a large number of critical processes is likely to result in performance problems. Also note that we are still within the few seconds bound, which

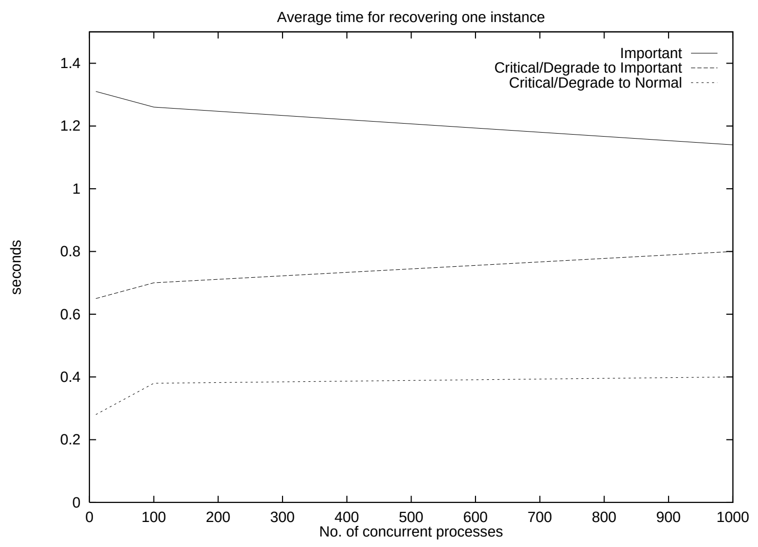


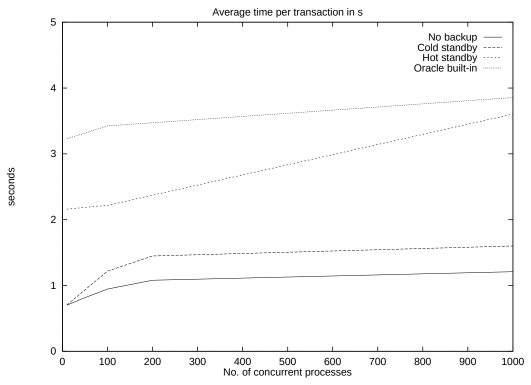
Figure 10: Time to recover from server failures depending on process type

is not a dramatic increase in delay.

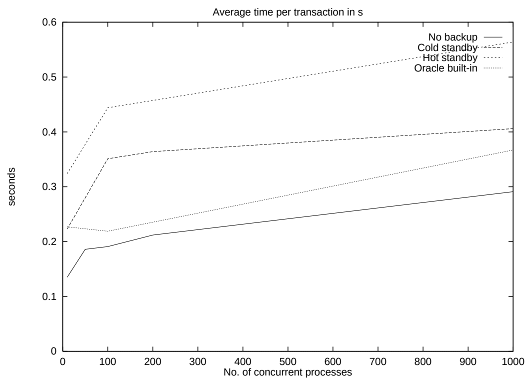
5.6 Time to recover from failures

The time to recover from server failure is shown in Figure 10. The average time for recovering one process was between 1.2 and 1.3 seconds. This means that for a large database with 1000 processes the overall recovery time is about 19 minutes, which leads to an average unavailability of 9.5 minutes since processes are available as soon as they are recovered. The main cost factor is the installation of the process image in the database, which requires re-executing all original transactions by interpreting the change buffers. While the time for recovery may seem very long, it is within the bounds that are tolerable in business process environments where activities are very long. Moreover, it is similar to the recovery cost in database systems. The performance can be further improved by recovering multiple processes in parallel. Since no data conflicts between process images exist, the degree of parallelism for process recovery is bound only by processor performance and disk bandwidth.

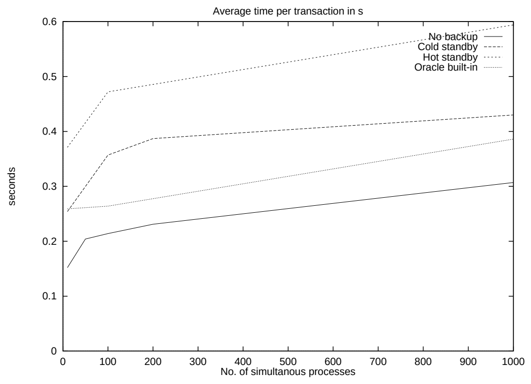
The recovery cost for critical processes is dependent on the degree of fault tolerance that a process has to provide *after* a fail-over. If the process has to stay *critical*, the recovery time is as long as for important processes, since a new backup has to be elected and the process has to be installed in its database. Because of this, normally *process degradation* is applied, i.e., the process' availability is reduced every time a failure occurs. If it is reduced to *important*, during failover only change buffers have to be stored the backup, which leads to a considerable improvement of the recovery time while preserving the process's availability. It is possible to promote a process to *critical* level again dynamically at a later point in time. If the probability of server crashes is moderate, critical processes can be downgraded to the *normal* availability level, which means that they will block if their new primary should fail. By using this scheme, the recovery cost is further reduced. We give the recovery cost for both variants of degradation in Figure 10. Note that, again, we did not apply parallel recovery of multiple processes. We plan to integrate mechanisms for parallel application of change buffers as a future optimization. The performance of the recovery using degradation to normal availability is especially promising for process migration facilities. Migrating a process involves creating a backup copy at the new location and then making this copy the working copy. With the performance obtained in our experiments, this seems like a feasible approach.



(A) Start Process



(B) Start Activity



(C) Term. Activity

Figure 11: Comparison of backup overhead for different operations

5.7 Comparing the different approaches

Figures 11.A, 11.B, and 11.C show the results for the three different operations in the four approaches discussed: no backup, database replication, cold standby, and hot standby. The first conclusion to draw from the comparison is that high level backup has about the same overhead as backup strategies implemented over database replication. Given the added flexibility, this allows us to conclude that semantic backup is not only feasible but also a good choice in a PSS. As discussed above, the database approach requires manual intervention in case of failures in order to switch from the primary to the backup. This, in many cases, renders the approach unfeasible in practice. In addition, database strategies only work if the database platform is homogeneous, that is, the same database is used as primary and backup. In some cases, it is even required that both primary and backup run on the same operating system. Experience shows that this is a very limiting factor. This fact has important implications. Since current commercial PSS do not, in general, incorporate backup mechanisms, it is possible to use the techniques described in this paper to extend their functionality with a backup service without degrading their performance.

5.8 Impact of server location

In the experiments described so far, the database servers were located on remote machines and each database access had the additional cost of communication between the PSS server and the database. Since the database accesses are by far the dominant factor in the cost of a transaction, it could be assumed that by reducing communication cost the overall overhead of replication would be reduced too. To investigate the impact of communication on the overall performance, experiments were made with a modified configuration that placed the process engines and their databases on the same machine (Figure 3B). With this configuration, PSS and database communicate over the much faster shared memory. The results of these experiments

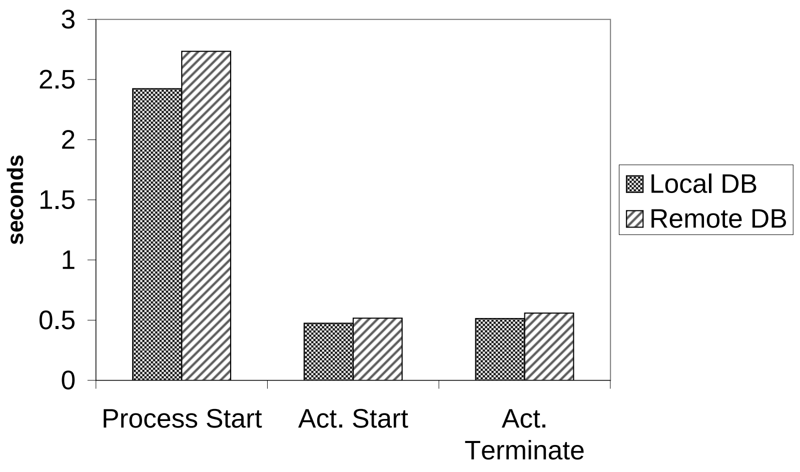
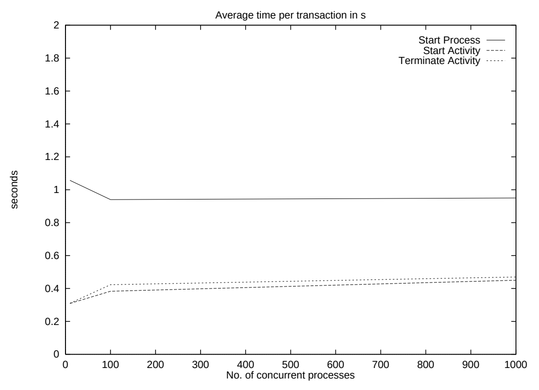


Figure 12: Impact of database location on performance

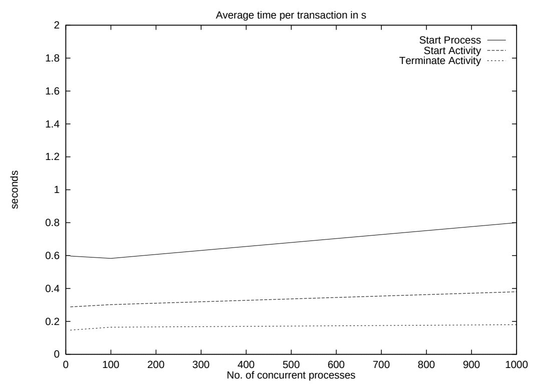
(Figure 12) show, however, that the gain in performance is very small, mainly because DBMS and PSS have to share the same processor. (Note that the absolute numbers shown cannot be compared to the results of the other experiments because the SUN10 servers used here are slower than the PCs employed in the previous measurements.) This small impact of communication overhead, even when using comparatively slow Ethernet connections, implies that the general deployment of a three-tier architecture with database and server residing on different nodes is feasible from a performance point of view. This allows to extend the proposed architecture, without performance penalties, to more general schemes where a server uses multiple, distributed databases in order to achieve a higher processing capacity and to further increase the resilience to failures. In such an environment, a process engine is operational even if some of its databases fail. While in the case of a database failure, some processes will have to be taken over by the backup, the other processes can continue navigation and the server stays accessible for the start of new process instances which are stored in the remaining databases. Another promising option is the shared use of databases by multiple servers. This allows, if a PSS server fails, for quick fail-over by starting a new server process on a running machine and connecting it to the still accessible database of the failed server.

5.9 Heterogeneous environments

An important challenge when introducing a PSS is to integrate all pre-existing systems. Most current commercial PSS require a specific DBMS that has to be bought and installed in addition to the process environment itself. One of our goals in the overall design was to avoid such restrictions. In order to study the behavior of the system when heterogeneous databases are used, we configured a system (Figure 3.C) with servers using both an object-oriented database engine (ObjectStore) and a relational engine (Oracle8). The servers worked according to the usual scheme, with the object-oriented databases acting as backup for the relational ones and vice-versa. Due to space limitations we present here only the results for the cold-standby backup algorithms (Figure 13). The performance of the ObjectStore-based server (a) was slightly better than that of the relational one (b), due to the more efficient caching mechanisms it provides. ObjectStore has a page-server architecture where a cache manager process at each client machine maintains a pool of cached database pages. This improves performance especially for applications where no concurrent access from multiple sites takes place. In contrast to this, in the Oracle-based implementation, each process had



(a)



(b)

Figure 13: Heterogeneous environment: (a) Oracle-based server and (b) ObjectStore-based server

to be fetched from the server prior to navigation. We expect that with the implementation of a client-side cache the performance of the systems will become similar. Note that the performance of the Oracle-based server was better here than in the homogeneous environment because of the faster application of the change buffers at the (ObjectStore-based) backup.

Our experiments with the heterogeneous environment are encouraging and show that database independence can be achieved with acceptable overheads. Another important point is that semantic replication of the type used in our backup strategies provides an efficient way of introducing replication mechanisms into systems that have no built-in replication support (like ObjectStore) or only offer asynchronous replication (like most commercial relational DBMS).

6 Conclusions

We have presented an architecture for implementing highly available process support systems based on clusters of cooperating servers and on semantic backup mechanisms. We have considered several possible solutions in the context of process support systems and shown how solutions based on commercial products, either on backup tools or on replication engines, suffer from a number of drawbacks. For instance, using database replication mechanisms restricts the configuration to use the same type of database on all servers. Furthermore, since the backup strategy needs to be 2-safe, the choice of database engines is restricted to those providing synchronous replication, which, to our knowledge, is currently only Oracle. In addition, changing the structure of a replicated environment in the case of fail-over is a costly operation that cannot be automated completely and involves human intervention, thereby incurring a long delay before processes can resume execution.

For these reasons, we have proposed an alternative solution based on using semantic knowledge about the application level, which allows to develop a backup system that is database-independent and flexible. Our experiments show that the approach is feasible and incurs in very low overheads. In addition, the solution proposed is comparatively fast when fail-over takes place. The backup mechanism was also extended to allow process migration during runtime, an issue we expect will play a significant role in the scalability of future systems.

The results are encouraging and show a feasible technique to achieve availability in the specific domain of process support systems. The principles presented can also be seen in a wider context. For instance, as a way to provide backup or replication services in environments where the database engine does not provide such services. Also, the access pattern observed in a PSS is quite different from that considered in traditional database applications. Neither OLTP (short transactions, small set of changes) nor OLAP (long transactions, mainly read operations) characterize the access patterns of a process support system, which can be characterized as online object manipulation. The work in this paper can thus be seen as a first step towards gaining a better understanding of such applications and, ultimately, establishing a new class of benchmarks.

References

[AAAM97] G. Alonso, D. Agrawal, A. El Abbadi, and C. Mohan. Functionality and limitations of current workflow management systems. *IEEE Expert, Special Issue on Cooperative Information Systems*, 12(5), Sept.-Oct. 1997.

[AH97] G. Alonso and C. Hagen. Geo-Opera: Workflow concepts for spatial processes. In *Proc. 5th Intl. Symposium on Spatial Databases (SSD '97)*, Berlin, Germany, 1997.

[AHST97] G. Alonso, C. Hagen, H.-J. Schek, and M. Tresch. Distributed processing over stand-alone systems and applications. In *23rd International Conference on Very Large Databases (VLDB '97)*, Athens, Greece, 1997.

[AM97] G. Alonso and C. Mohan. Workflow management: the next generation of distributed processing tools. In Sushil Jajodia and Larry Kerschberg, editors, *Advanced Transaction Models and Architectures*. Kluwer Academic Publishers, 1997.

[Bar81] J. Bartlett. A nonstop kernel. In *Proc. 8th ACM SIGOPS SOSP*, 1981.

[BHG87] P.A Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency control and recovery in database systems*. Addison Wesley, 1987.

[BK94] I.Z. Ben-Shaul and G.E. Kaiser. A paradigm for decentralized process modeling and its realization in the oz environment. In *Proceedings of the 16th International Conference on Software Engineering*, Sorrento, Italy, 1994.

[BN97] P.A. Bernstein and E. Newcomer. *Principles of Transaction Processing*. Morgan Kaufmann Publishers, 1997.

[BSR96] A. Bonner, A. Shrufi, and S. Rozen. LabFlow-1: A database benchmark for high throughput workflow management. In *Proc. of the Fifth International Conference on Extending Database Technology (EDBT96)*, Avignon, France, March 1996.

[GHOS96] J. Gray, P. Helland, P.E. O'Neil, and D. Shasha. The dangers of replication and a solution. In *Proc. ACM SIGMOD*, pages 173–182, 1996.

[GHS95] D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.

[GR93] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[HA98] C. Hagen and G. Alonso. Flexible exception handling in the OPERA process support system. In *Proc. 18th Intl. Conference on Distributed Computing Systems*, Amsterdam, The Netherlands, May 1998.

[Hsu95] M. Hsu, editor. *Bulletin of the IEEE Technical Comittee on Data Engineering. Special Issue on Workflow Systems*. IEEE Computer Society, March 1995.

[IBM97] IBM. *DB2 Replication Guide and Reference*. IBM, 1997.

[KGAM96] M. Kamath, R. Günthör, G. Alonso, and C. Mohan. Providing high availability in very large workflow management systems. In *Proc. of the Fifth International Conference on Extending Database Technology (EDBT)*, Avignon, France, March 1996.

[Kno97] T. Knoop. *SQL Server 11.0.x Technical Overview*. 1997.

[KR96] M. Kamath and K. Ramamritham. Bridging the gap between transaction management and workflow management. Athens, Georgia, USA, May 1996.

[LA94] F. Leymann and W. Altenhuber. Managing business processes as an information resource. *IBM Systems Journal*, 33(2):326–348, 1994.

[MVW96] J. Meidanis, G. Vossen, and M. Weske. Using workflow management in dna sequencing. In *Proceedings of the First International Conference on Cooperative Information Systems (CoopIS)*, Brussels, 1996.

[Ora95] Oracle. *Oracle7 Distributed Database Technology and Symmetric Replication*. Oracle Corporation, 1995. White Paper.

[Ora97] Oracle. *Oracle8 Server Replication*. Oracle Corporation, 1997. Reference Manual.

[SKM$^+$96] A. Sheth, K.J. Kochut, J. Miller, D. Worah, S. Das, C. Lin, D. Palaniswami, J. Lynch, and I. Shevchenko. Supporting state-wide immunization tracking using multi-paradigm workflow technology. In *Proc. of the 22nd Intl. Conf. on Very Large Databases*, Bombay, India, September 1996.

[TKP94] A.Z. Tong, G.E. Kaiser, and S.S. Popovich. A flexible rule-chaining engine for process-based software engineering. In *Proceedings of the Ninth Knowledge-Based Software Engineering Conference*, Monterey, CA, USA, 1994.