South African Computer Journal
Number 6
March 1992

Suid-Afrikaanse Rekenaar-tydskrif
Nommer 6
Maart 1992

Computer Science and Information Systems

Rekenaarwetenskap en Inligtingstelsels

The South African Computer Journal

An official publication of the South African Computer Society and the South African Institute of Computer Scientists

Die Suid-Afrikaanse Rekenaartydskrif

'n Amptelike publikasie van die Suid-Afrikaanse Rekenaarvereniging en die Suid-Afrikaanse Instituut vir Rekenaarwetenskaplikes

Editor
Professor Derrick G Kourie
Department of Computer Science
University of Pretoria
Hatfield 0083
Email: dkourie@dos-lan.cs.up.ac.za

Subeditor: Information Systems
Prof John Schochot
University of the Witwatersrand
Private Bag 3
WITS 2050
Email: 035ebrse@witsvma.wits.ac.za

Production Editor
Prof G de V Smit
Department of Computer Science
University of Cape Town
Rondebosch 7700
Email: gds@cs.uct.ac.za

Subscriptions

	Annual	Single copy
Southern Africa:	R45_00	R15_00
Elsewhere	$45_00	$15_00

to be sent to:
Computer Society of South Africa
Box 1714 Halfway House 1685

Computer Science Department, University of Pretoria, Pretoria 0002
Phone: (int)+(012)+420-2504 Fax: (int)+(012)+43-6454 email: dkourie@rkw-risc.cs.up.ac.za

Sigma Press

Guest Contribution

This guest contribution is a slightly edited report to the Foundation for Research and Development (FRD) drawn up by Ed Coffman. Ed was an FRD-sponsored guest at the 6th South African Computer Research Symposium. The report was not originally intended for general distribution. Rather, it was specifically compiled for the FRD and its staff. I am therefore grateful to both the FRD and the author for agreeing to its publication in SACJ. I believe that it contains several incisive observations that merit further thought and discussion amongst South African computer scientists. (Editor)

Impressions of Computer Science Research in South Africa

E. G. Coffman, Jr.
AT&T Bell Laboratories, USA

In commenting on the cross section of computer science research in South Africa, I will use the classification in the table of contents of the "Summary of Awards: Fiscal Year 1989," a document published recently by the US National Science Foundation. Of the 5 categories, I will treat Numeric and Symbolic Computation as inappropriate for the discussion below. In this category I noted no research in the computer science setting in South Africa. It is also common in the US and elsewhere to place this effort in other departments, e.g., departments of mathematics or applied mathematics.

Of the remaining categories I found South Africa to be strongest in software systems and engineering, to have a substantial investment in computer systems and architecture, and to be weakest in computer and computation theory.

The coverage in software systems and engineering (SSE) was broad, topical, and similar in scope to that in US universities. Technology transfer and the corresponding relations with industry seemed to be in place or developing along promising lines. I comment in passing that this was rather surprising to me. In the US the development of SSE within university departments has lagged behind almost all other disciplines of computer science. A primary problem has been the insatiable appetite of industry for all Ph.D. graduates in the SSE field.

The investment in parallel processing, computer networks, and distributed computing appears sound, although I expected to see a greater emphasis on mathematical foundations (see my remarks below), particularly in the parallel algorithms area. Given current resources, South African institutions are doing remarkably well in computer science research. But computer science is a fundamentally important course of study, beginning at an early age and extending through graduate Ph.D. research; I take this as sufficiently obvious that I need not dwell on justifications. With this in mind, and with the necessary resources in hand, South Africa should, in my opinion, expand and consolidate its computer science research effort, increase its visibility in the international arena, and correct the rather thin distribution of graduate research among universities.

I can see much of this proceeding along present lines, but I would strongly recommend a concerted development in computer and computation theory (CCT), education and research; this is mainstream computer science and forms the basis for virtually all other fields of study within computer science. It is by no means absent in South Africa curricula, but it appears to be under--represented in advanced studies and Ph.D. level research.

At the graduate level CCT is heavily mathematical. I understand that mathematical foundations are supplied by mathematics departments in certain cases. This is not ideal, but workable and it is justified by limited resources. However, it is important that mathematics departments not regard this as a mere service; faculty will have to make a major commitment to theoretical computer science, publishing in its leading journals (e.g. SIAM Journal of Computing, Journal of the ACM, Journal of Algorithms, Algorithmica, Journal of Computer and Systems Sciences, Theoretical Computer Science, etc.), and providing the supervision of theses sponsored by computer science departments and leading to degrees in computer science. I would also encourage active participation in the international computer science "theory" societies and their meetings; two highly prestigious examples of the latter are the annual Symposium on the Theory of Computing and the Foundations of Computer Science conference.

Returning to the thin distribution of computer science research, I would make the following point. If the current situation is only a stage of development - i.e., if further resources (both human and financial) can be counted on to bring at least a few of the departments to a critical mass - then little needs to be said beyond the earlier remarks. Critical mass is hard to define, but calls for adequate, expert coverage of mainstream computer science re- search. In view of the breadth of this research, 8-10 Ph.D. full- time-equivalent faculty would seem to be barely adequate; with the usual clumping of faculty in specific research areas, more would be expected. South Africa has a talent base such that there is little doubt that such departments would achieve a much wider international recognition.

On the other hand, if resources remain fixed at current or even slightly retrenched levels, then I would recommend consolidation to achieve the same goals on a smaller scale. Within a university this can often be done by establishing interdisciplinary, degree-granting laboratories or institutes of computer science, which bring together the computer science efforts located in various departments other than computer science, such as electrical engineering, industrial engineering, business/-management science, mathematics, and operations research. The idea is to enjoy the advantages (opportunity, synergy, awareness, etc.) to both students and faculty of reasonably large computer science programs. There are many examples of such intramural laboratories in North America and Europe.

This approach could also be considered among universities within a confined geographical area, admittedly with greater difficulty perhaps. The Institute of Discrete Mathematics and Computer Science connecting Princeton University, Rutgers University, AT&T Bell Laboratories, and Bell Communications Research is a possible model. Examples in South Africa might consist of universities and research institutions on the Reef or those in the Western Cape (just to mention those with which I'm a little familiar).

As a final comment, I should note that my impressions have been based on limited information which may not give a representative picture. I am sure that my reactions will be appropriately discounted where I have been off target.

Editor's Notes

Prof John Schochot has graciously accepted to be SACJ's subeditor for papers relating to Information Systems. Authors wishing to submit papers in this general area should please contact him directly. I look forward working with John, and to a significant increase in IS contributions in future.

The hand of the new production editor, Riël Smit, will be clearly evident in this issue. Those papers not prepared in camera-ready format by the authors themselves were prepared by him in TEX. He will be announcing revised guidelines for camera-ready format in a future issue. If you use TEX or one of its variations, Riël would be happy to provide you with a styles document to SACJ format.

At last some Department of National Education committee has decided that SACJ should now be on the list of approved journals. This places it amongst the ranks of some 6800 other journals. These include not merely a number of ACM and IEEE Transactions but also such journals as *Ostrich*, *Trivium*, *Crane Bag*, *Koers*, *Mosquito News*, *Police Chief*, *Connoisseur*, *Lion and the Unicorn*, *About the House* and *Ohio Agricultural Research and Development Center Department Series ESS*. You will recall that in 1990 this same committee decided that, if judged on its own merits, SACJ did not deserve to be on the illustrious list. In the absence of other evidence, we must assume that the sole reason for its revised decision is that SACJ's predecessor, Quæstiones Informaticæ, was there. (I have a secret suspicion that the committee liked that name.)

It is my understanding that for official purposes, all journals on this list are regarded as *equally* meritorious, and all of them are more meritorious than *any* conference proceedings. What does all of this mean?

The momentous implication of the committee's deliberations is that the State will not give your institution a single cent for anything that you publish in SACJ. Instead, the State and your institution will scrupulously keep a score of the annual number of publications that count - but actually don't - because someday they might! And to encourage your enthusiastic participation in this Alice in Wonderland exercise, your institution might actually give you some of the standard subsidy funding that the State should have provided according to its own formulae, but didn't.

You will not be allowed to use this money to buy yourself a car - not even a casual meal. You may only use it to finance activities that are provably directed towards producing more papers in approved journals. The great consolation, of course, is that you will not be required to pay income tax on this money. The only tax involved will be the VAT component when you spend it in an approved manner. As a good computer scientist who enjoys recursion, my vote would be that all such revenue collected by the State should be earmarked to be placed in the pay packets of committee members who decided that SACJ should be approved.

If you publish in these approved journals with sufficient regularity and enthusiasm you will almost deserve to be regarded as a researcher. What you additionally need to do, is to ensure that you befriend and impress at least three overseas referees. You then apply to the FRD for official recognition as a researcher, and if they are sufficiently impressed, they will give you more of the non-taxable kind of money that you need to spend on research to publish in approved journals.

Derrick Kourie
Editor

Extending Local Recovery Techniques for Distributed Databases

H L Viktor and M H Rennhackkamp
Department of Computer Science, University of Stellenbosch, Stellenbosch, 7600

Abstract

The most frequently used local database system failure recovery techniques are logging, shadowing and differential files. In a distributed database these local system failure recovery techniques may be utilized for recovery from a single site failure. However, these techniques need to be extended to facilitate continued distributed executions. Various extended local system failure recovery techniques are presented. The results of a comparison of these techniques are shown. It is concluded the deferred data item logging technique proves to be the best for the system under consideration.
Keywords: *Database management, distributed database system, failures, recovery techniques, recovery.*
Computing Reviews Categories: *H.2.2, H.2.4, H.2.7*

Received September 1991, Accepted September 1991

1 Introduction

During a local system failure the contents of main memory is lost and processing is terminated at the site at which the failure occurred. This could result in an inconsistent local database, since logically completed updates may not be reflected in the physical database files. At restart, the results of such locally committed, unpropagated executions should be propagated. Operational sites can continue execution during the failure of another site. This increases the complexity of distributed database recovery. At restart, the outcome of partially completed distributed executions should be determined. In addition, copies of replicated data items may be outdated. These copies should be updated to reflect the new values.

The recovery management function provides the facilities to preserve the consistency of the database in the event of a failure. In a distributed environment these facilities include techniques to consistently recover the local database, bearing in mind distributed recovery aspects. Various local system failure recovery techniques for centralized databases exist. The design and implementation of these techniques are well understood. In a distributed database these techniques may be extended to facilitate for the recovery from a site failure caused by a local system failure. Few authors have addressed distributed database recovery utilizing local recovery techniques. The results of such an extension are presented.

In the next section, a distributed transaction execution model is presented. This study forms part of the NRDNIX DDBMS, which is currently being developed at the University of Stellenbosch. An overview of this system is given. This is followed by a description of widely used local system recovery techniques. The extensions to these techniques, as required by the NRDNIX DDBMS, are discussed. Lastly, results of a comparative modelling of these extended techniques are presented.

2 Distributed Transactions

A transaction performs operations on the data items in a database. The transaction execution includes the reading, possible modification and the subsequent writing of data items. A sequence of operations constituting a transaction are delimited by the BEGIN-TRANSACTION, COMMIT or ABORT keywords. It is a sequence of actions considered as an atomic logical unit of work. If the transaction completes successfully, it is committed. Otherwise, it is aborted and no effects of the transaction remains in the database.

In a distributed environment transactions perform operations on data scattered across sites of a network. A distributed transaction is executed by processes at these sites. These processes, called subtransactions, are able to communicate and cooperate with each other.

The actions performed by a transaction should be indivisible. Either all of a transaction's actions should be properly reflected in the database, or none. This indivisibility requirement is met by a transaction satisfying the following characteristics, also referred to as the ACIDS-principle:

1. Atomicity. Acceptance of the transaction should be a guarantee that it will be run exactly once, any update executed only once and its results produced exactly once. The atomicity of distributed transactions are guaranteed by utilizing an atomic commit protocol.
2. Consistency. Whenever a transaction executes on a database state that is initially consistent, it must leave the database in a consistent state after it terminates. A successful distributed transaction results from the execution and completion of successful subtransactions.
3. Isolation. Events within the transaction should be hidden from others. From the distributed transaction definition it follows that this property is satisfied.
4. Durability. Once a transaction has successfully completed execution, the system should guarantee that the results of its operations will never be lost, except in the

event of a catastrophe.

Once a distributed transaction has committed its results should be guaranteed to be permanent, even in the event of a failure. The results of subtransaction execution cannot be ensured since a successful subtransaction is aborted if the coordinator issues a global abort.

5. Serializability. In most systems transactions are allowed to execute concurrently. The concurrent execution of transactions should be guaranteed to be serializable. An interleaved execution of transactions is serializable if it produces the same output and has the same effect on the database as some serial execution of the same transaction. The activity of guaranteeing transactions' serializability is called concurrency control. In a distributed database, the global as well as the local concurrency must be maintained.

A transaction which satisfies the ACIDS-principle is said to be successful. A database is consistent if and only if it contains the results of successful transactions [11].

The indivisibility of a distributed transaction is enforced by employing an atomic commit protocol. The basic idea of an atomic commit protocol is to determine a unique decision for all participants with respect to committing or aborting a transaction. The most widely used commitment protocol is the two phase commit (2PC) protocol. The protocol consists of two phases. During the first phase, the coordinator decides to either commit or abort the distributed transaction. The second phase is used to implement this decision. If a participant is unable to locally commit the subtransaction, then all participants must locally abort [7].

The 2PC protocol has been extended to eliminate unnecessary blocking during site failures [22]. Here, an additional phase 2a is introduced. The participants do not directly commit the transaction during the second phase of commitment. Instead, they reach a new prepared-to-commit or prepared-to-abort state. This modified 2PC protocol is utilized in the distributed database management system (DDBMS) under consideration.

3 NRDNIX DDBMS

In this section an overview of the NRDNIX system is given. The design areas which are of importance when discussing recovery related aspects are highlighted.

The NRDNIX DDBMS is implemented as a number of processes, called Managers, on XENIX, a version of UNIX System V [17].

Presentation Manager

The aim of the presentation manager is to form an efficient user interface to the DDBMS. It accepts user requests and transforms them to optimized internal commands according to information obtained from the data dictionary. The communication kernel is activated to execute the internal commands. The presentation manager receives the results of these executions and presents them to the users.

A distributed transaction consists of one or more sessions. The internal commands which forms part of a distributed transaction are decomposed into sessions. Each group of related single relation operations, each binary join operation and the final project operation constitute a session. A session is considered a unit and a session sequence number is assigned to it. Concurrent split processing is utilized. The local executions of a distributed databases' sessions are grouped into subtransactions which are concurrently executed at all operational sites in the network.

Data Dictionary

The function of the data dictionary is to manage the central inventory of the metadata, i.e. descriptive data about the data stored in the database. It provides a number of functions for managing, inserting, updating and retrieving metadata from the system relations. The data dictionary maintains relevant information to support the management of replicated data.

Communication Kernel

The communication kernel controls the concurrent execution of distributed transactions. The site at which a transaction is initiated is the *coordinator* of the transaction. The other sites are referred to as *participants*.

The communication kernel consists of 5 components.

- The *transaction manager* identifies and manages the sessions of locally issued transactions. The global execution of the session is initiated by requesting the network manager to broadcast the session to *all* remote sites. The concurrency controller is requested to schedule the session locally. The results of executed transactions are returned to the presentation manager.
- The *slave session manager* controls the local execution of remotely issued sessions. It acts as a participant in the transaction of which the session forms part.
- The *master session manager* controls the global execution of each locally initiated session and thus coordinates the transaction's sessions.
- The *concurrency controller* schedules the execution of transactions in conservative timestamp order. Sessions received from both local and remote sites are buffered in timestamp ordered lists.
- The *recovery manager* is responsible for the coordination of all recovery related aspects introduced by the distributed nature of the database. Its task includes the recovery from site failures as well as network partitions. The recovery manager also controls the global commitment of transactions by utilizing the modified 2PC protocol.

Database Manager

The database manager is responsible for the physical retrieval of data and the physical execution of operations. It consists of the *cache* and *access managers*. The cache manager controls the execution of operations on the database. The access manager forms the interface between the cache manager and the XENIX file system. It is primarily

responsible for managing records of data files. Accesses to the local database files are attained through the XENIX kernel. The access manager of the local DBMS interacts with the XENIX kernel via standard system calls. All low level operations on the database files are therefore controlled by the XENIX file system, thus abstracting the DBMS from the physical disk accesses.

Network Manager

The network manager provides a reliable error-free communication service to the communication kernel. The ArcNet local area network communication facilities are used. ArcNet supports a modified token passing protocol on a bus architecture. It forms a virtual ring, where each site receiving the token acknowledges it.

A DDBMS component wishing to communicate with another site sends a message via the network protocol manager. A message is divided into fixed length packets, which are sent to the network device driver. The device driver is responsible for the actual communication.

The network protocol manager employs point-to-point communication with packet acknowledgement. In addition, a reliable broadcasting facility is offered when not more than a specified threshold of sites fail. The broadcasting facility is used when a coordinator initiates remote sessions [16].

4 Local System Failure Recovery Techniques

Three basic techniques are widely used for recovery from local system failures, namely logging, shadowing and differential files [1, 12].

Log Files

An incremental log, also called journal or audit trail, is a representation of the history of transaction execution at a particular site. It records all actions performed on the local database. Data are collected for the sole purpose of recovering invalid data from the local database and supplementing the local database with updates of completed transactions that were not yet reflected in it at the time of failure [11, 18, 2].

Recovery data are written to the log prior to the actual transaction execution. Write ahead logging satisfies the following two rules [8, 11]:

1. Undo information is written to the log file before the corresponding updates are propagated to the materialized local database.
2. Redo information is written to the log before a transaction is committed. The system must be able to ensure the transaction's durability once the redo information has been written.

Logging is usually combined with some form of checkpointing. A checkpoint records the local database state at a particular instance. Checkpoints are used to eliminate unnecessary redo of transactions which have already written their updates to the local database, as well as to obtain the list of transactions to be undone or redone. At restart, the checkpoint is used to generate undo and redo lists. At a checkpoint all log information is written to nonvolatile storage. All modified local database file blocks are propagated. At the completion of the checkpoint, a checkpoint record is written to the log. The checkpoint record may contain lists of the currently active and recently aborted or committed transactions.

There are two basic approaches to log transaction executions, namely physical and logical logging [11, 18].

Physical Logging

Some part of the physical presentation of modified data is written to the log. Either the *state*, before or after a change, or the *transition* causing the change is logged. A before-value log file record the value of modified data before a transaction execution and is used by the undo algorithm. After-value log files, which record the value of data items after updates have occurred, are utilized by the redo algorithm.

Physical Page Logging

The most basic physical logging method uses a page as unit of log information. The before-image and/or after-image of a page, or the difference between these images, is written to the log.

- With *state logging*, all the pages containing the changes executed by a transaction operation are written to the log. The before-images are written to the log file before the corresponding updates are applied to the local database. The after-image of a page is written to the log file before the transaction is committed.

  At system restart the undo algorithm uses the before-images of all the modified pages to restore incomplete transactions. The redo algorithm uses the after-image of each page to propagate the results of committed, but as yet unpropagated, transactions.
- With *transition logging*, the *exclusive-or'ed* difference between the old and the new page is written to the log before it is propagated to disk. Usually only a small part of data on a page are affected by a change. The *xor'ed* difference will thus contain long strings of 0's, which are removed by compression techniques [11]. With compression techniques, savings of 20 to 50 percent are typical for text files.

  At system restart the *xor* is applied to the decompressed pages. By applying the *xor* difference to the before-image of a page, the after-image are obtained. On the other hand, applying the *xor* to a after-image will subsequently yield the before-image.

Physical Data Item Logging

The changes to data items are logged, rather than the whole page.

- In the *deferred write* approach all updates on data-items are recorded on the log, but the actual writes are deferred until the transaction partially commits. The log consists of entries of the form *[T,ts,x,v]*, identifying the value v that transaction T wrote into data item x at

time *ts*, thus reflecting the state and the transition of a data item after modification. When the transaction partially commits, a commit record is written to the log. The actual updating of the records is initiated by applying the log to the physical local database.

At restart, a no-undo, redo algorithm is executed. If the failure occurred after the commit of a transaction was recorded in the log, a redo algorithm updates the database files to include the results of committed unpropagated values. The log entries regarding incomplete or aborted transactions are simply ignored.

- With the *immediate write* approach all updates are immediately applied to the local database. The write ahead log contains all changes to data items. The log consists of entries of the form *[T,ts,x,o,v]*, identifying the old value *o* of data item *x* as well as the new value *v* that transaction *T* wrote at time *ts*.

  At restart, the log information is used to restore the state of the system to a previous consistent state. All committed, but unpropagated transactions are redone by propagating the new value *v*. Similarly, all relevant active or aborted transactions are undone by propagating the old value *o*.

Logical Logging

A logical log consists of a sequence of transaction operations executed on the local database. The operations, the arguments they operate on, as well as some control information are logged prior to the execution thereof. Thus the log is abstracted from the physical level. No information regarding the actual accessed data items are maintained.

At restart, committed unpropagated transactions are redone by re-executing the data manipulation language (DML) statements as recorded in the log. It is possible to remove the effects of aborted transactions by executing an undo algorithm. However, such an undo algorithm may introduce inconsistencies and should therefore be avoided [21]. The logical logging recovery technique need therefore be combined with another recovery technique which utilizes a no-undo algorithm.

Shadows

Two copies of the data being updated during transaction execution are kept, the original copy referred to as the *shadow* and a modified *current* copy. When a transaction commits, the shadow copy is replaced by the newly updated copy [12, 14, 15]. These two copies exist only during updating; otherwise only a single copy exists which contains the current value.

Shadowing is difficult to implement in systems which allow several transactions to execute concurrently [14]. If several transactions concurrently alter a file, file save or restore is inappropriate because it aborts or commits the updates of all transactions to the file. It is desirable to commit or abort on a per transaction basis.

Shadowing and Logging

Shadow paging is combined with logging [10] to facilitate for concurrent transaction execution. A current and shadow version of a file is maintained. When a shadow page is updated for the first time, a new disk page is assigned to it. Thereafter, when the page is read from or written to disk, the new page is used. The shadow is never updated. Saving the results of a transaction consists of writing to disk all altered relevant pages currently in main memory and freeing superseded shadow pages. At the commitment of a transaction the results are propagated to disk. The results of all transactions which have previously committed will be reflected in the database.

Aborting a transaction is achieved by discarding pages of that file in the buffer pool, freeing all the new disk pages of that transaction and reverting to the shadow page table. At restart all shadowed files are reset to their shadow versions. The log is used to remove the effects of aborted transactions and to restore the effects of committed transactions.

Transaction Orientated Shadowing

Another approach, based on ideas by Lorie [15], is presented by Agrawal [1]. For each relation a shadow page table is maintained. For each transaction an incremental current page table is formed in main memory. When a transaction wishes to commit, the transaction's current page table is written to a commit list. This acts as a transaction's precommit record. It ensures that all updated buffer pages corresponding to the transaction are output to disk. The current page table of a transaction is used to update the system shadow page table. The disk address of the new system current page table overwrites the address of the old system shadow table. The system shadow table is discarded and the current system table becomes the new shadow as the transaction commits. Finally, a commit statement is written to the commitlist.

At restart, the current page tables of transactions are discarded to remove the effects of incomplete executions. The commitlist is examined to determine those transactions for which a precommit record appears in the list, but not the commit record. For all such transactions, the shadow system page table is updated using the precommit record.

Differential Files

All logical files comprise of two physical files: a read-only database file and a read-write differential file [19]. Accessed data items are maintained in the differential file. The differential file is always searched first when data are retrieved, thus obtaining the most recent entry for a given item. Data not found in the differential file are retrieved from the main local database. A hashing scheme is usually implemented to minimize the overhead associated with the determination of the location of an item. The base file remains unchanged until a merging algorithm is executed. During merging the results of committed transactions are moved from the differential file to the local database file [21].

Differential Files and Logging

All updates by concurrent transactions are written to one differential file. The differential file is combined with logging to determine the transaction execution sequence. The transaction identifiers, a unique timestamp as well as information on the transaction's operations are written to the log.

At restart, the logical database consists of the main database and differential file. In addition, committed transactions which have not yet been written to the differential file are redone by utilizing the logging information.

At a differential file checkpoint, all buffer pages are written to the differential file by utilizing one of the local checkpointing schemes. In addition a new differential file may be opened. The results of active transactions may be moved to a new differential file. The current differential file, which contains only the results of committed transactions, is closed and the new differential file becomes current.

Transaction Orientated Differential File

Each transaction maintains its own transaction differential file which is inaccessible to other transactions. The differential file is decomposed into two files: an append file A and a deletion file D. Each logical file R is considered a view $R = (B \bigcup A) - D$, where B is the read-only base portion of R. Each transaction is assigned a unique timestamp. The differential file tuples are widened to include an extra timestamp. While a transaction is active, its updates are executed on its local A_l and D_l files that are inaccessible to other transactions. When the transaction commits, the local files are appended to the main differential files A_g and D_g. Finally, the timestamp of the committing transaction is written to a commitlist. If a transaction aborts, its private A_l and D_l files are simply discarded and its timestamp is not appended to the commitlist.

To recover from a system crash, instead of R, a view of R is used that consists solely of the tuples whose timestamp fields contain values that appear in the commitlist. The logical database consists of the main database as well as the results of transactions included in the commitlist. These committed transactions are contained in the main differential file.

5 Distributed Recovery

The autonomy of sites are of critical importance when designing a DDBMS [7]. A NRDNIX recovery manager must be able to recover a site to a consistent state autonomously. The previously introduced local recovery techniques need to be extended to be utilized in the NRDNIX distributed environment.

The degree with which distributed transaction executions satisfy the ACIDS-principle is used as metric to evaluate the correctness thereof. The presented extended local system recovery techniques satisfy the ACIDS-principle. A globally consistent database state is reached at the successful completion thereof [22].

Logging

The results of locally initiated distributed transactions and subtransaction executions are logged utilizing local logging. Commit processing information is also logged. This occurs at all sites of the network.

At restart, a previous consistent local database state is obtained by consulting a previous checkpoint. In distributed databases, a global checkpoint refers to a set of saved local states of sites. Such a set of local checkpoints is consistent if it forms a globally consistent state [5, 9, 13, 22].

The results of subtransactions which have committed since this last checkpoint are redone. The results of distributed executions which occurred during the failure are obtained from the logs of operational sites via the communication network. The local redo algorithm is applied to the remote log information. First, subtransactions of partially completed distributed transactions are completed. Second, the results of newly submitted transactions are made part of the database.

The remote information may be obtained in one of two ways. One or more sites may be elected as the source. Alternatively the recovery information may be received from all operational sites. In NRDNIX, a transaction is concurrently executed at all sites of the database. It is therefore sufficient to request the logged information from only one site. Usually the log of the site which acted as token during the failure is utilized.

Shadowing and Logging

Locally initiated distributed transactions and subtransactions execute on current copies of the local database files. These executions are logged prior to their execution. Commit processing information is also logged as part of the local log. Local checkpointing is implicitly executed each time a subtransaction commits during the last phase of the commit protocol.

At restart, a previous consistent database state is obtained by starting from the local shadow file. The local log is consulted and the results of all locally committed, but as yet unpropagated, transaction executions are executed on this copy. The recovering site obtains the results of partially completed transactions from the commit log of the token site. These transactions are locally completed. The log at the token site is consulted to obtain the results of newly submitted transactions. These results are made part of the local log by execution a redo algorithm.

Transaction Orientated Shadowing

A separate current file is created for each subtransaction of a distributed transaction. The current page table is written to a commitlist during the last phase of the commit protocol. Once this has been done, it is ensured that the results of the subtransaction will survive a site failure. Finally, a commit record is written to the commitlist.

At restart, the local commitlist is examined to determine those subtransactions for which a precommit record appears in the list but not a commit record. The shadow page table is updated using the precommit record. The

commitlist is obtained from an operational site. The remote shadow file information of all newly submitted transactions are made part of the local database.

Differential Files and Logging

One main differential file is utilized. The results of subtransactions of locally as well as remotely issued distributed transactions are maintained therein.

At restart, the main database together with the committed transactions recorded in the differential file reflect a consistent state. The results of locally committed, unpropagated transaction executions may be found at two places: in the differential file or in the log. The results of committed logged transactions which are not contained in the differential file are made part of the database by executing a redo algorithm.

The results of recently completed distributed transactions are located in the differential files of operational sites and are located in their logs. One of two approaches may be utilized. In the first, the remote log information is obtained and the transactions are explicitly re-executed. In the second, both the remote differential file and log information are utilized. The differential file records corresponding to the committed transactions are selected and are copied to the local database files.

Transaction Orientated Differential Files

A logical transaction differential file is utilized by the subtransactions at each participant in a distributed transaction. Physically this file consists of two files: a deletion file (D_l) and an add file (A_l). When a subtransaction is locally committed, this local differential file is appended to a main differential file A_g and D_g. The timestamp thereof is written to a commitlist. The commit information is maintained in the commitlist.

At restart, a previous consistent database state is constructed. This state consists of the main database file as well as the records in the main differential file for which an entry is the commitlist has been recorded. In addition, the information of transactions committed during the failure are made part of the local database. This information is obtained from operational sites. The relevant parts of remote main differential files are received via the network. It is appended to the main differential file at the recovering site.

6 Comparison of Techniques

A model to uniformly model the overhead associated with the various recovery techniques was developed [22]. The various techniques are uniformly modelled against the same set of transactions, number of sites and distributed database specifications. These specifications are directly derived from the way processing is performed in NRDNIX.

Figure 1 shows the parameters which are used in the modelling. The distributed database consists of 5 sites. For this example the maximum number of records at a site is taken as 400. A total of 19 distributed transactions are concurrently executed at the 5 sites in the distributed database. These distributed transactions are in different stages of execution when the failure occurs [22].

Parameter	Amount
Sites	5 sites
Maximum Records per Site	400 records
Distributed Transactions (DTs)	19 sites
Completed and Committed DTs	5 sites
Completed and Aborted DTs	5 sites
Restarted and Committed DTs	5 sites
Restarted and Aborted DTs	4 sites

Figure 1: Modelling Parameters

The disk storage, message and I/O overheads on the distributed transactions, imposed by the extended recovery techniques, are determined.

Disk Storage Overhead

Disk storage overhead is incurred by the redundant recovery data which are accumulated and maintained during normal processing [10, 12, 20]. The amount of additional disk storage overhead is determined by the number of pages written to disk. The page length is considered to be 1024 bytes, corresponding to the size of a XENIX page.

Message Overhead

The additional message traffic [6] imposed by the recovery techniques are taken as metric when discussing message overhead, since it will directly influence the transmission delay and thus the overall transaction execution. In the NRDNIX system, the fixed cost associated with each message dominates the overall transmission cost. The message transmission cost is therefore not much influenced by the variable cost caused by the message length [16]. The number of additional messages is easily determined and is sufficient to model the message overhead [6].

I/O Overhead

The additional I/O operations are incurred when accessing and propagating recovery related data to disk and when executing I/O operations during recovery. The movement of data is performed by executing a XENIX read or write system call. The number of additional I/O cycles are used as the unit of modelling when discussing I/O overhead [1, 10, 12].

The CPU overhead imposed by the recovery techniques are implicitly included in the I/O costs. The reason therefore is threefold. Firstly, the ability of the system to exploit the capacity of faster CPUs is directly influenced by the amount of I/O operations [18]. Secondly, a large amount of the CPU overhead is incurred by the setting up of the I/O operations [12]. Thirdly, the I/O and communication costs are shown to dominate the processing costs [3].

7 Results and Evaluation

The results of the comparison are shown in figures 2 and 3. These values reflect the overhead during normal and recovery processing [4, 22]. The normal processing overhead is taken as the cost incurred during a fixed time interval T. This overhead is calculated by considering all sites in the distributed database. The resulting values reflect the overhead at the most expensive site. The recovery processing overhead is calculated at the recovering site. It consists of the reprocessing overhead required to recover the set of distributed transactions to a globally consistent state.

Technique	Disk bytes	I/O cycles
Page State Log	69 836	102
Page Transition Log	28 016	68
Deferred Data Item Log	986	136
Immediate Data Item Log	1054	102
Deferred Data Item Log & Shadows	3366	238
Immediate Data Item Log & Shadows	3434	204
Logical Log & Shadows	3672	153
Transaction Orientated Shadows	238	255
Transaction Orientated Dif. Files	34 918	221
Immediate Data Item Log & Dif. Files	36 040	204
Deferred Data Item & Dif. Files	35 972	238
Logical Log & Dif. Files	36 244	153

Figure 2: Normal Processing Overhead

Normal Processing Overhead Evaluation

During normal processing, message overhead is incurred by the execution of the commit protocol. This overhead is equal for all techniques and is therefore not shown.

Transaction orientated shadowing yields the best disk storage overhead, but the I/O cost is high. The physical page transition logging technique yields the best I/O overhead results, with a high disk storage overhead. Both the physical data item logging approaches yields both low disk storage and I/O costs, with the deferred write approach slightly better than the immediate write approach. Data item logging therefore seems the most suitable method for the system under consideration.

Recovery Processing Overhead Evaluation

Transaction orientated shadowing yields the best disk storage overhead and message overhead, but the I/O cost is high. Logical logging combined with the shadowing techniques yields the best I/O overhead with a moderately low disk storage and message overhead. Both the physical data item logging approaches yields low disk storage, I/O and message costs, with the deferred write approach slightly better than the immediate write approach. Physical data item logging therefore seems the most suitable method for the system under consideration.

Technique	Disk bytes	I/O cycles	Messages
Page State Log	32 832	136	32
Page Transition Log	13 129	136	13
Deferred Data Item Log	448	68	1
Immediate Data Item Log	480	136	1
Deferred Data Item Log & Shadows	1506	236	2
Immediate Data Item Log & Shadows	1440	142	2
Logical Log & Shadows	1552	36	2
Transaction Orientated Shadows	204	236	1
Transaction Orientated Dif. Files	16 432	40	16
Immediate Data Item Log & Dif. Files	16 928	136	17
Deferred Data Item & Dif. Files	16 896	68	17
Logical Log & Dif. Files	17 040	36	17

Figure 3: Recovery processing overhead

8 Conclusion and Extensions

Local system failure recovery techniques are widely utilized in centralized databases. These techniques need to be extended for a distributed environment. The extension of local system recovery techniques is useful since the distributed database management system utilizes existing and well understood techniques. The autonomy of sites are preserved since the recovery management function is localized.

An overview of the extended local system failure recovery techniques was given. The results of a comparative modelling of these techniques were presented. The modelling is aimed at a specific distributed database environment. The deferred data item logging technique proves the best technique for the specific system, both during normal and recovery processing.

The paper concerns the failure of one site due to local system failures. Multiple site failures and network partitions were not discussed. An extension to this work presents techniques and modelling to incorporate these types of failures [22]. In addition, the model should also be applied to other distributed database environments.

References

[1] R Agrawal and D J DeWitt, Integrated Concurrency

Control and Recovery Mechanisms: Design and Performance Evaluation, *ACM Transactions on Database Systems*, **10**(4), December 1985.

[2] P A Bernstein, V Hadzilacos and N Goodman, *Concurrency Control and Recovery in Database systems*, Addision-Wesley, Reading, 1987.

[3] B Bhargava and J Riedl, The Raid Distributed Database System, *IEEE Transactions on Software Engineering*, **15**(6), June 1989, p726-736.

[4] A F Cardenas et al, Performance of Recovery Architectures in Parallel Associative Database Processors, *ACM Transactions on Database Systems*, **8**(3), September 1985.

[5] J Chang, Simplifying Distributed Database Systems by using a Broadcast Network, *Proceedings of the ACM Sigmod*, 1984.

[6] C-P Chou and M-T Liu, A Concurrency Control Mechanism and Crash Recovery for a Distributed Database System, *Distributed Data Base*, North-Holland Publishing Company, 1980.

[7] S Ceri and G Pelagatti, *Distributed Databases: Principles and Systems*, McGraw-Hill, New York, 1984.

[8] C J Date, *An Introduction to Database Systems*, Volume 1, Addison-Wesley, 1986.

[9] E Gelenbe and D Derochette, Performance of Rollback Recovery Systems under Intermittent Failures, *Communications of the ACM*, **21**(6), June 1978.

[10] J Gray et al, The Recovery Manager of the System R Database Manager, *ACM Computing Surveys*, **13**(2), June 1981.

[11] T Haerder and A Reuter, Principles of Transaction-Orientated Database Recovery, *ACM Computing Surveys*, **15**(4), December 1983.

[12] J Kent and H Garcia-Molina, Optimizing Shadow Recovery Algorithms, *IEEE Transactions on Software Engineering*, **14**(2), February 1988, p155-168.

[13] R Koo and S Toueg, Checkpointing and Rollback Recovery for Distributed Systems, *IEEE Transactions on Software Engineering*, Vol SE-13(1), January 1987.

[14] H F Korth and A Silberschatz, *Database System Concepts*, McGraw-Hill, 1986.

[15] R A Lorie, Physical Integrity in a Large Segmented Database, *ACM Transactions on Database Systems*, **2**(1), March 1977.

[16] M D Meumann, The Design and Implementation of a Communication Kernel for a Distributed Database System, *M.Sc. Thesis*, Department of Computer Science, University of Stellenbosch, South Africa, 1990.

[17] M H Rennhackkamp, The NRDNIX Distributed Database System, The South African Computer Journal, **1**, 1990, p5-10.

[18] A Reuter, Performance Analysis of Recovery Techniques, *ACM Transactions in Database Systems*, **9**(4), December 1984.

[19] D G Severance and G M Lohman, Differential Files: Their Application to the Maintenance of Large Databases, *ACM Transactions on Database Systems*, **1**(3), 1976, p256-267.

[20] S H Son and A K Agrawala, Distributed Checkpointing for Globally Consistent States of Databases, *IEEE Transactions on Software Engineering*, **15**(10), October 1989, p1157-1167.

[21] H L Viktor and M H Rennhackkamp, Database Consistency under UNIX, *The South African Computer Journal*, **4**, March 1991.

[22] H L Viktor, A Comparative Study of Recovery Techniques in Distributed Databases, *M.Sc. Thesis*, Department of Computer Science, University of Stellenbosch, South Africa, 1991.

Notes for Contributors

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems, as well as shorter technical research papers. However, non-refereed review and exploratory articles of interest to the journal's readers will be considered for publication under sections marked as a Communications or Viewpoints. While English is the preferred language of the journal papers in Afrikaans will also be accepted. Typed manuscripts for review should be submitted **in triplicate** to the editor.

Form of Manuscript

Manuscripts for review should be prepared according to the following guidelines.

- Use double-space typing on one side only of A4 paper, and provide wide margins.
- The first page should include:
  - title (as brief as possible);
  - author's initials and surname;
  - author's affiliation and address;
  - an abstract of less than 200 words;
  - an appropriate keyword list;
  - a list of relevant Computing Review Categories.
- Tables and figures should be on separate sheets of A4 paper, and should be numbered and titled. Figures should be submitted as original line drawings, and not photocopies.
- Mathematical and other symbols may be either handwritten or typed. Greek letters and unusual symbols should be identified in the margin, if they are not clear in the text.
- References should be listed at the end of the text **in alphabetic order** of the (first) author's surname, and should be cited in the text in square brackets. References should thus take the following form:

  [1] E Ashcroft and Z Manna, [1972], The translation of 'GOTO' programs to 'WHILE' programs, *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, 250-255.

  [2] C Bohm and G Jacopini, [1966], Flow diagrams, Turing machines and languages with only two formation rules, *Comm. ACM*, **9**, 366-371.

  [3] S Ginsburg, [1966], *Mathematical theory of context free languages*, McGraw Hill, New York.

Manuscripts *accepted* for publication should comply with the above guidelines, and may be provided in one of the following formats:

- in a **typed form** (i.e. suitable for scanning);
- as an **ASCII file** on diskette; or
- as a **WordPerfect**, **TEX** or **LATEX** or file; or
- in **camera-ready** format.

A page specification is available on request from the editor, for authors wishing to provide camera-ready copies. A styles file is available from the editor for Wordperfect, **TEX** or **LATEX** documents.

Charges

Charges per final page will be levied on papers accepted for publication. They will be scaled to reflect scanning, typesetting, reproduction and other costs. Currently, the minimum rate is R20-00 per final page for camera-ready contributions and the maximum is R100-00 per page for contributions in typed format.

These charges may be waived upon request of the author and at the discretion of the editor.

Proofs

Proofs of accepted papers will be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

Note that, in the case of camera-ready submissions, it is the author's responsibility to ensure that such submissions are error-free. However, the editor may recommend minor typesetting changes to be made before publication.

Letters and Communications

Letters to the editor are welcomed. They should be signed, and should be limited to about 500 words.

Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

Book reviews

Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

Advertisement

Placement of advertisements at R1000-00 per full page per issue and R500-00 per half page per issue will be considered. These charges exclude specialized production costs which will be borne by the advertiser. Enquiries should be directed to the editor.

South African Computer Journal

Number 6, March 1992
ISSN 1015-7999

Suid-Afrikaanse Rekenaar-tydskrif

Nommer 6, Maart 1992
ISSN 1015-7999

Contents

GUEST CONTRIBUTION

Impressions of Computer Science Research In South Africa
E.G. Coffman, Jr. 1

RESEARCH ARTICLES

An Implementation of the Linda Tuple Space under the Helios Operating System
PC Clayton, EP Wentworth, GC Wells & FK de-Heer-Menlah 3

Modelling the Algebra of Weakest Preconditions
C Brink and I Rewitzky 11

The Design and Analysis of Distributed Virtual Memory Consistency Protocols in an Object Oriented OS
KJ McGregor and RH Cambell 21

An Object Oriented Framework for Optimistic Parallel Simulation on Shared-Memory Computers
P Machanik 27

Analysing Routing Strategies in Sporadic Networks
SW Melville 37

Using Statecharts to Design and Specify a Direct-Manipulation User Interface
L Van Zijl & D Mitton 44

Extending Local Recovery Techniques for Distributed Databases
HL Viktor & MH Rennhackkamp 59

Efficient Evaluation of Regular Path Programs
PT Wood 67

Integrating Similarity-Based and Explanation-Based Learning
GD Oosthuizen & C Avenant 72

Evaluating the Motivating Environment For IS Personnel in SA Compared to the USA. (Part I)
JD Cougar & DC Smith 79

TECHNICAL NOTE

An Implementation of the Parallel Conditional
U Jayasekera and NCK Philips 85

COMMUNICATIONS AND REPORTS

Book Review 87

The CSP Notation and its Application to Parallel Processing
PG Clayton 90