PROCEEDINGS

Southern African Computer Symposium

6th de

1991

Rekendarsimposium van Suider Afrika

DE OVERBERGER HOTEL, CALEDON

2 - 3 JULY 1991

SPONSORED BY

ISM
FRD
GENMIN

EDITED BY
M H Linck

DEPARTMENT OF COMPUTER SCIENCE • UNIVERSITY OF CAPE TOWN
PROCEEDINGS / KONGRESOPSOMMINGS

6th
SOUTHERN AFRICAN COMPUTER SYMPOSIUM

6de
SUIDELIKE-AFRIKAANSE REKENAARSIPOSEIUM

De Overberger Hotel, Caledon
2 - 3 JULY 1991

SPONSORED by
ISM
FRD
GENMIN

EDITED by
M H LINCK
Department of Computer Science
University of Cape Town
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreword</td>
<td>1</td>
</tr>
<tr>
<td>Organising Committee</td>
<td>2</td>
</tr>
<tr>
<td>Referees</td>
<td>3</td>
</tr>
<tr>
<td>Program</td>
<td>5</td>
</tr>
<tr>
<td>Papers (In order of presentation)</td>
<td>9</td>
</tr>
<tr>
<td>&quot;A value can belong to many types&quot; B H Venter, University of Fort Hare</td>
<td>10</td>
</tr>
<tr>
<td>&quot;A Transputer Based Embedded Controller Development System&quot; M R Webster, R G Harley, D C Levy &amp; D R Woodward, University of Natal</td>
<td>16</td>
</tr>
<tr>
<td>&quot;Improving a Control and Sequencing Language&quot; G Smit &amp; C Fair, University of Cape Town</td>
<td>25</td>
</tr>
<tr>
<td>&quot;Design of an Object Orientated Framework for Optimistic Parallel Simulation on Shared-Memory Computers&quot; P Machanick, University of Witwatersrand</td>
<td>40</td>
</tr>
<tr>
<td>&quot;Using Statecharts to Design and Specify the GMA Direct-Manipulation User Interface&quot; L van Zijl &amp; D Mitton, University of Stellenbosch</td>
<td>51</td>
</tr>
<tr>
<td>&quot;Product Form Solutions for Multiserver Centres with Hierarchical Classes of Customers&quot; A Krzesinski, University of Stellenbosch and R Schassberger, Technische Universität Braunschweig</td>
<td>69</td>
</tr>
<tr>
<td>&quot;A Reusable Kernel for the Development of Control Software&quot; W Fouché and P de Villiers, University of Stellenbosch</td>
<td>83</td>
</tr>
<tr>
<td>&quot;An Implementation of Linda Tuple Space under the Helios Operating System&quot; P G Clayton, E P Wentworth, G C Wells and F de Heer-Menlah, Rhodes University</td>
<td>95</td>
</tr>
<tr>
<td>&quot;The Design and Analysis of Distributed Virtual Memory Consistency Protocols in an Object Orientated Operating System&quot; K Macgregor, University of Cape Town &amp; R Campbell University of Illinois at Urbana-Champaign</td>
<td>107</td>
</tr>
</tbody>
</table>
"Concurrency Control Mechanisms for Multidatabase Systems"
A Deacon, University of Stellenbosch 118

"Extending Local Recovery Techniques for Distributed Databases"
H L Victor & M H Rennhackkamp, University of Stellenbosch 135

"Analysing Routing Strategies in Sporadic Networks"
S Melville, University of Natal 148

The Design of a Speech Synthesis System for Afrikaans"
M J Wagener, University of Port Elizabeth 167

"Expert Systems for Management Control: A Multiexpert Architecture"
V Ram, University of Natal 177

"Integrating Similarity-Based and Explanation-Based Learning"
G D Oosthuizen and C Avenant, University of Pretoria 187

"Efficient Evaluation of Regular Path Programs"
P Wood, University of Cape Town 201

"Object Orientation in Relational Databases"
M Rennhackkamp, University of Stellenbosch 211

"Building a secure database using self-protecting objects"
M Olivier and S H von Solms, Rand Afrikaans University 228

"Modelling the Algebra of Weakest Preconditions"
C Brink and I Rewitsky, University of Cape Town 242

"A Model Checker for Transition Systems"
P de Villiers, University of Stellenbosch. 262

"A New Algorithm for Finding an Upper Bound of the Genus of a Graph"
D I Carson and O R Oellermann, University of Natal 276
FOREWORD

The 6th Computer Symposium, organised under the auspices of SAICS, carries on the tradition of providing an opportunity for the South African scientific computing community to present research material to their peers.

It was heartening that 31 papers were offered for consideration. As before all these papers were refereed. Thereafter a selection committee chose 21 for presentation at the Symposium.

Several new dimensions are present in the 1991 symposium:

* The Symposium has been arranged for the day immediately after the SACLA conference.

* It is being run over only 1 day in contrast to the 2-3 days of previous symposia.

* I believe that it is first time that a Symposium has been held outside of the Transvaal.

* Over 85 people will be attending. Nearly all will have attended both events.

* A Sponsorship package for both SACLA and the Research Symposium was obtained. (This led to reduced hotel costs compared to previous symposia)

A major expense is the production of the Proceedings of the Symposium. To ensure financial soundness authors have had to pay the page charge of R20 per page.

A thought for the future would be consideration of a poster session at the Symposium. This could provide an alternative approach to presenting ideas or work.

I would sincerely hope that the twinning of SACLA and the Research Symposium is considered successful enough for this combination survive. As to whether a Research Symposium should be run each year after SACLA, or only every second year, is a matter of need and taste.

A challenge for the future is to encourage an even greater number of MSc & PhD students to attend the Symposium. Unlike this year, I would recommend that they be accommodated at the same cost as everyone else. Only if it is financially necessary should the sponsored number of students be limited.

I would like to thank the other members of the organising committee and my colleagues at UCT for all the help that they have given me. A special word of thanks goes to Prof. Pieter Kritzinger who has provided me with invaluable help and ideas throughout the organisation of this 6th Research Symposium.

M H Linck
Symposium Chairman
SYMPOSIUM CHAIRMAN

M H Linck, University of Cape Town

ORGANISING COMMITTEE

D Kourie, Pretoria University.
P S Kritzinger, University of Cape Town.
M H Linck, University of Cape Town.

SPONSORS

ISM
GENMIN
FRD
<table>
<thead>
<tr>
<th>NAME</th>
<th>INSTITUTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barnard, E</td>
<td>Pretoria</td>
</tr>
<tr>
<td>Becker, Ronnie</td>
<td>UCT</td>
</tr>
<tr>
<td>Berman S</td>
<td>UCT</td>
</tr>
<tr>
<td>Bishop, Judy</td>
<td>Wits</td>
</tr>
<tr>
<td>Berman, Sonia</td>
<td>UCT</td>
</tr>
<tr>
<td>Brink, Chris</td>
<td>UCT</td>
</tr>
<tr>
<td>Bodde, Ryn</td>
<td>Networks Systems</td>
</tr>
<tr>
<td>Bornman, Chris</td>
<td>UNISA</td>
</tr>
<tr>
<td>Bruwer, Piet</td>
<td>UOFS</td>
</tr>
<tr>
<td>Cherenack, Paul</td>
<td>UCT</td>
</tr>
<tr>
<td>Cook Donald</td>
<td>UCT</td>
</tr>
<tr>
<td>de Jaeger, Gerhard</td>
<td>UCT</td>
</tr>
<tr>
<td>de Villiers, Pieter</td>
<td>Stellenbosch</td>
</tr>
<tr>
<td>Ehlers, Elize</td>
<td>RAU</td>
</tr>
<tr>
<td>Eloff, Jan</td>
<td>RAU</td>
</tr>
<tr>
<td>Finnie, Gavin</td>
<td>Natal</td>
</tr>
<tr>
<td>Gaynor, N</td>
<td>AECI</td>
</tr>
<tr>
<td>Hutchinson, Andrew</td>
<td>UCT</td>
</tr>
<tr>
<td>Jourdan, D</td>
<td>Pretoria</td>
</tr>
<tr>
<td>Kourie Derrick</td>
<td>Pretoria</td>
</tr>
<tr>
<td>Kritzinger, Pieter</td>
<td>UCT</td>
</tr>
<tr>
<td>Krzesinski, Tony</td>
<td>Stellenbosch</td>
</tr>
<tr>
<td>Laing, Doug</td>
<td>ISM</td>
</tr>
<tr>
<td>Labuschagne, Willem</td>
<td>UNISA</td>
</tr>
<tr>
<td>Levy, Dave</td>
<td>Natal</td>
</tr>
</tbody>
</table>
MacGregor, Ken  UCT
Machanick, Philip  Wits
Mattison Keith  UCT
Messerschmidt, Hans  UOFS
Mutch, Laurie  Shell
Neishlos, N  Wits
Oosthuizen, Deon  Pretoria
Peters Joseph  Simon Fraser
Ram, V  Natal, Pmb.
Postma, Stef  Natal, Pmb
Rennhackkamp, Martin  Stellenbòsch
Shochot, John  Wits
Silverberg, Roger  Council for Mineral Technology
Smit, Riel  UCT
Smith, Dereck  UCT
Terry, Pat  Rhodes
van den Heever, Roelf  UP
van Zijl, Lynette  Stellenbosch
Venter, Herman  Fort Hare
Victor, Hema  Stellenbosch
von Solms, Basie  RAU
Wagenaar, M  UPE
Wentworth, Peter  Rhodes
Wheeler, Graham  UCT
Wood, Peter  UCT
6TH RESEARCH SYMPOSIUM - 1991

FINAL PROGRAM

TUESDAY 2nd July 1991

10h00 - 13h00  Registration

13h00 - 13h50  PUB LUNCH

14h00 - 15h30  SESSION 1A
Venue: Hassner
Chairman: Prof Basie von Solms

14h00 - 14h30  "A value can belong to many types."
B H Venter, University of Fort Hare

14h30 - 15h00  "A Transputer Based Embedded Controller Development System"
M R Webster, R G Harley, D C Levy & D R Woodward, University of Natal

15h00 - 15h30  "Improving a Control and Sequencing Language"
G Smit and C Fair, University of Cape Town

14h00 - 15h30  SESSION 1B
Venue: Hassner C
Chairman: Prof Roelf v d Heever

14h00 - 14h30  "Design of an Object Orientated Framework for Optimistic Parallel Simulation on Shared-Memory Computers"
P Machanick, University of Witwatersrand

14h30 - 15h00  "Using Statecharts to Design and Specify the GMA Direct-Manipulation User Interface"
L van Zijl & D Mitton, University of Stellenbosch

15h00 - 15h30  "Product Form Solutions for Multiserver Centres with Hierarchical Classes of Customers"
A Krzesinski, University of Stellenbosch and R Schassberger, Technische Universität Braunschweig

15h30 - 16h00  TEA
16h00 - 17h30 SESSION 2A

Venue: Hassner

Chairman: Prof Derrick Kourie

16h00 - 16h30
"A Reusable Kernel for the Development of Control Software" W Fouche and P de Villiers, University of Stellenbosch

16h30 - 17h00
"An Implementation of Linda Tuple Space under the Helios Operating System" P G Clayton, E P Wentworth, G C Wells and F de-Heer-Menlah, Rhodes University

17h00 - 17h30
"The Design and Analysis of Distributed Virtual Memory Consistency Protocols in an Object Orientated Operating System" K MacGregor, University of Cape Town & R Campbell, University of Illinois at Urbana-Champaign

19h30 PRE-DINNER DRINKS

20h00 GALA CAPE DINNER
(Men: Jackets & ties)
WEDNESDAY 3rd July 1991

7h00 - 8h15 BREAKFAST

8h15 - 9h45 SESSION 3A

Venue: Hassner

Chairman: Assoc Prof P Wood

8h15 - 8h45
"Concurrency Control Mechanisms for Multidatabase Systems" A Deacon, University of Stellenbosch

8h45 - 9h15
"Extending Local Recovery Techniques for Distributed Databases" H L Victor & M H Rennhackkamp, University of Stellenbosch

9h15 - 9h45
"Analysing Routing Strategies in Sporadic Networks" S Melville, University of Natal

9h45 - 10h15 TEA

10h15 - 11h00 SESSION 4

Venue: Hassner

Chairman: Prof P S Kritzinger
Invited paper: E Coffman

11h00 - 11h10 BREAK

SESSION 3B

Venue: Hassner C

Chairman: Prof G Finnie

8h15 - 8h45
"The Design of a Speech Synthesis System for Afrikaans" M J Wagener, University of Port Elizabeth

8h45 - 9h15
"Expert Systems for Management Control: A Multiexpert Architecture" V Ram, University of Natal

9h15 - 9h45
"Integrating Similarity-Based and Explanation-Based Learning" G D Oosthuizen and C Avenant, University of Pretoria
11h10 - 12h40 SESSION 5A

Venue: Hassner

Chairman: Prof C Bornman

11h10 - 11h40
"Efficient Evaluation of Regular Path Programs"
P Wood, University of Cape Town

11h40 - 12h10
"Object Orientation in Relational Databases"
M Rennhackkamp, University of Stellenbosch

12h10 - 12h40
"Building a secure database using self-protecting objects" M Olivier and S H von Solms, Rand Afrikaans University

SESSION 5B

Venue: Hassner C

Chairman: Prof A Krzesinski

11h10 - 11h40
"Modelling the Algebra of Weakest Preconditions"
C Brink & I Rewitsky, University of Cape Town

11h40 - 12h10
"A Model Checker for Transition Systems"
P de Villiers, University of Stellenbosch

12h10 - 12h40
"A New Algorithm for Finding an Upper Bound of the Genus of a Graph"
D I Carson and O R Oellermann, University of Natal

12h45-12h55 GENERAL MEETING of RESEARCH SYMPOSIUM ATTENDEES

Venue: Hassner

Chairman: Dr M H Linck

13h00 - 14h00 LUNCH

FINIS 6th COMPUTER SYMPOSIUM
PAPERS
of the
6TH RESEARCH SYMPOSIUM
Extending Local Recovery Techniques for Distributed Databases

H L Viktor and M H Rennhackkamp

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

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 NRDNNIX 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 NRDNNIX 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 BEGIN-TRANSACTION and
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 ACIDSPrinciple:

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. The 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].

4.1 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].

4.1.1 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, 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 \(o\).

### 4.1.2 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. While it is possible to remove the effects of aborted transactions by executing an undo algorithm, 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.

### 4.2 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.

#### 4.2.1 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.
4.2.2 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.

4.3 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].

4.3.1 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.

4.3.2 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 $R = (B \cup A) - D$, 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 $A_t$ and $D_t$ files that are inaccessible to other transactions. When the transaction commits, the local files are appended to the main differential files $A_2$ and $D_2$. Finally, the timestamp of the committing transaction is written to a commitlist. If a transaction aborts,
its private $A_i$ and $D_i$ 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].

5.1 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.

5.2 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.
5.3 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.

5.4 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.

5.5 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 ($D_1$) and an add file ($A_1$). When a subtransaction is locally committed, this local differential file is appended to a main differential file $A_2$ and $D_2$. 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].

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sites</td>
<td>5 sites</td>
</tr>
<tr>
<td>Maximum Records per Site</td>
<td>400 records</td>
</tr>
<tr>
<td>Total Distributed Transactions (DTs)</td>
<td>19 sites</td>
</tr>
<tr>
<td>Completed and Committed DTs</td>
<td>5 sites</td>
</tr>
<tr>
<td>Completed and Aborted DTs</td>
<td>5 sites</td>
</tr>
<tr>
<td>Restarted and Committed DTs</td>
<td>5 sites</td>
</tr>
<tr>
<td>Restarted and Aborted DTs</td>
<td>4 sites</td>
</tr>
</tbody>
</table>

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 NRDNI X 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.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Disk bytes</th>
<th>I/O cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page State Log</td>
<td>69 836</td>
<td>102</td>
</tr>
<tr>
<td>Page Transition Log</td>
<td>28 016</td>
<td>68</td>
</tr>
<tr>
<td>Deferred Data Item Log</td>
<td>986</td>
<td>136</td>
</tr>
<tr>
<td>Immediate Data Item Log</td>
<td>1054</td>
<td>102</td>
</tr>
<tr>
<td>Deferred Data Item Log &amp; Shadows</td>
<td>3366</td>
<td>238</td>
</tr>
<tr>
<td>Immediate Data Item Log &amp; Shadows</td>
<td>3434</td>
<td>204</td>
</tr>
<tr>
<td>Logical Log &amp; Shadows</td>
<td>3672</td>
<td>153</td>
</tr>
<tr>
<td>Transaction Orientated Shadows</td>
<td>238</td>
<td>255</td>
</tr>
<tr>
<td>Transaction Orientated Diff. Files</td>
<td>34 918</td>
<td>221</td>
</tr>
<tr>
<td>Immediate Data Item Log &amp; Diff. Files</td>
<td>36 040</td>
<td>204</td>
</tr>
<tr>
<td>Deferred Data Item &amp; Diff. Files</td>
<td>35 972</td>
<td>238</td>
</tr>
<tr>
<td>Logical Log &amp; Diff. Files</td>
<td>36 244</td>
<td>153</td>
</tr>
</tbody>
</table>

Figure 2: Normal Processing Overhead per site.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Disk bytes</th>
<th>I/O cycles</th>
<th>Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page State Log</td>
<td>32 832</td>
<td>136</td>
<td>32</td>
</tr>
<tr>
<td>Page Transition Log</td>
<td>13 129</td>
<td>136</td>
<td>13</td>
</tr>
<tr>
<td>Deferred Data Item Log</td>
<td>448</td>
<td>68</td>
<td>1</td>
</tr>
<tr>
<td>Immediate Data Item Log</td>
<td>480</td>
<td>136</td>
<td>1</td>
</tr>
<tr>
<td>Deferred Data Item Log &amp; Shadows</td>
<td>1506</td>
<td>236</td>
<td>2</td>
</tr>
<tr>
<td>Immediate Data Item Log &amp; Shadows</td>
<td>1440</td>
<td>142</td>
<td>2</td>
</tr>
<tr>
<td>Logical Log &amp; Shadows</td>
<td>1552</td>
<td>36</td>
<td>2</td>
</tr>
<tr>
<td>Transaction Orientated Shadows</td>
<td>204</td>
<td>236</td>
<td>1</td>
</tr>
<tr>
<td>Transaction Orientated Diff. Files</td>
<td>16 432</td>
<td>40</td>
<td>16</td>
</tr>
<tr>
<td>Immediate Data Item Log &amp; Diff. Files</td>
<td>16 928</td>
<td>136</td>
<td>17</td>
</tr>
<tr>
<td>Deferred Data Item &amp; Diff. Files</td>
<td>16 896</td>
<td>68</td>
<td>17</td>
</tr>
<tr>
<td>Logical Log &amp; Diff. Files</td>
<td>17 040</td>
<td>36</td>
<td>17</td>
</tr>
</tbody>
</table>

Figure 3: Recovery 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.

The transaction orientated shadowing technique 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

The transaction orientated shadowing technique 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.

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


146


