Recovery Architectures for Multiprocessor Database Machines

Rakesh Agrawal
AT&T Bell Laboratories
Murray Hill, New Jersey

David J DeWitt
Computer Sciences Department
University of Wisconsin
Madison, Wisconsin

ABSTRACT

During the past decade, a number of database machine designs have been proposed Most of these designs have been optimized with respect to retrieval queries, while ignoring the issue of recovery and its impact on performance of the proposed architecture In this paper, we propose several parallel recovery architectures for multiprocessor database machines, examine the characteristics and performance of each, and evaluate their impact on the performance of the database machine Our results indicate that a recovery architecture based on parallel logging has the best overall performance

1. Introduction

During the past decade, database machines have been the subject of intense research activity, and a number of database machine designs have been proposed (see the surveys in [13,24]) However, most of these designs have been optimized only with respect to retrieval queries In reality, databases are continually updated With update operations, a major function of the database management software is to keep the database consistent in presence of failures This problem, referred to as the recovery problem, requires that a consistent state of the database objects be restored after a software or hardware failure

Database-machine designers (except for producers of commercial products such as Britton-Lee and Terradata) have virtually ignored the issue of recovery and its impact on the performance of the proposed machines The study of recovery architectures for RAP-like associative processors in [8] is the only work in this area In this paper, we propose several parallel recovery architectures for the multiprocessor-cache class of database machines, and evaluate their performance and impact on the performance of the database machine

The organization of the rest of the paper is as follows In Section 2, we give an overview of the multiprocessor database machine architecture that we have considered In Section 3, we present several parallel recovery architectures based on three basic recovery mechanisms logging, shadows, and differential files In section 4, we investigate the behavior and the performance of each of these recovery architectures Finally, in Section 5, we examine the comparative performance of the proposed recovery architectures and present our conclusions

2. Database Machine Architecture

The database machine architecture that we consider consists of a set of processors, a multi-level memory hierarchy, and an interconnection device

Some of the processors, designated query processors, process transactions and operate asynchronously with respect to each other One of the processors, designated the back-end controller, acts as an interface to the host processor (the processor with which a user interacts) and coordinates the activities of the other processors

We assume that the memory hierarchy consists of three levels The top level consists of the internal memories of the query processors Each processor's local memory is assumed to be large enough to hold both a compiled transaction and several data pages Mass storage devices (disks) make up the bottom level and the middle level is a disk cache that is addressable by pages Management of pages within this cache is performed by the back-end controller

We assume that the bottom two levels of the memory hierarchy are connected in such a way that simultaneous data transfers can occur between each of the disk drives and any page frame in the cache A processor, designated the I/O processor, is responsible for transferring data pages between the disks and the cache The top two levels of the hierarchy are so connected that each processor can read and write a different page of the cache simultaneously and all processors can simultaneously read the same page of the cache See [6] for a discussion of the interconnection schemes that may satisfy these requirements

This database machine design has been classified as a multiprocessor-cache design [10] Examples of database machines in this class include DIRECT [9], RAP 2 [22], INFOPLEX [19], RDBM [14], and DBMAC [20]

3. Recovery Architectures

In this section we present parallel recovery architectures based on three basic recovery mechanisms (we assume familiarity with the basic notions of these recovery mechanisms, interested reader is encouraged to see [2] for a summary of these mechanisms)

- Log
- Shadows
- Differential Files

In an earlier study [2] we identified the major cost components associated with each of these recovery mechanisms In our design of the recovery architectures, the emphasis has been on using parallelism to reduce the penalty of these costs Recovery has two aspects

- collection of recovery data during the normal execution of the transactions,
- use of recovery data to perform necessary recovery actions in the event of a failure

All transactions, whether they complete or abort, incur the cost of collecting recovery data However, only in case of a failure is the recovery data used A recovery mechanism may make collection of recovery data relatively less expensive at the price of making recovery from failures costly On the other hand, a recovery mechanism may make recovery from failures cheaper at the expense of incurring a large penalty during the collection of recovery data From the performance viewpoint, we claim that the focus of an implementation should be on making the normal case efficient Therefore, in the algorithms presented below, we have optimized the collection of recovery data even if it meant making recovery from a failure more expensive while insuring that recovery can still be performed correctly

In all our architectures, we assume that a scheduler, located in the back-end controller, which employs page-level locking is used for concurrency control

3.1 Parallel Logging

The log-based approach [11,17] relies upon a redundant representation of the database on an append-only log Before updating a data object, every update operation creates a log record which is used to restore a consistent state of the database in case of a hardware or software failure

The basic idea of the parallel logging is to make the collection of recovery data efficient by allowing logging to occur in parallel at more than one log disk We postulate N log processors, where $N \geq 1$ Each log processor has associated with it a log disk We assume some interconnection between the query processors and the log processors that allows any query processor to send a log fragment to any log processor One possibility is to use a dedicated connection between the query and log processors Alternatively, the query processors can write their log fragments to the disk cache and then the log processors can read the fragments from there

When a query processor updates a page, it creates a log fragment for this page, selects a log processor and sends it the log fragment The log processor may be selected randomly, cyclicly (that is, each query processor cycles amongst all log processors), or by taking the mod of the query processor number (QpNo mod TotLp) or the transaction number (TranNo mod Totlp) with the total number of log processors Next the query processor sends the page number of the updated page and the identifier

of the log processor to which the log fragment for this page has been sent to the back-end controller The back-end controller records the log processor identifier in a page table

The log processor assembles log fragments received from different query processors in a log page When a log page has been filled, the log processor first writes it to its log disk Next the log processor sends the page numbers of all the updated pages which caused this log page to be created to the back-end controller where this information is recorded in the page-tables of the corresponding relations

If the back-end controller is forced to flush an updated page (e g when the cache is filled and there are no other pages to eject), it first determines whether the log record for this page has been written to disk by examining the page table If the log record for the page has not yet been written to the log, it sends a message to the corresponding log processor to flush the log page Only after the log processor acknowledges that the log page has been written to stable storage, will the data page be written to disk When committing a transaction, the back-end controller first ensures that all the log records of the transaction have been written to stable storage

Observe that in this architecture, log fragments for a transaction will in general be distributed over more than one log processor In [4] we show how the recovery from transaction and system failures may be performed without merging these distributed logs into one physical log We also show in [4] how system checkpointing can be performed in parallel with the normal data processing and logging activities without complete system quiescing

3.2 Shadow

Our model of the shadow mechanism, given in detail in [2], is based on the ideas in [15,18] This model is slightly different from the one System R uses [12], which is a combination of logging and the single-user shadow scheme proposed in [18] For each file, there is a shadow page-table that contains physical addresses of the data pages in the file When a transaction updates a data page, a new disk block is obtained for the updated copy, and its physical address is recorded in an incremental current page-table for the transaction At commit time, shadow page-table is carefully updated using the current page-table

Our evaluation in [2] (which matches the experience of the System R designers [12]) indicated that the major cost of the shadow [18] mechanism is the cost of *indirection* through the page table to access data pages In most of the file systems (except in UNIX†), one can calculate the location of a page given base of the file There appear to be two approaches that can be used to improve the performance of this recovery mechanism

- reduce the penalty of indirection,
- avoid indirection altogether

3.2.1 Reducing the Penalty of Indirection

The performance penalty of indirection through the page table may be reduced by keeping page tables on one or more page-table disks (which are different from data disks), and by using separate page-table processors for them The page-table processors are assumed to be under the control of the back-end controller Thus, if the accesses are uniformly distributed across the page-table processors, the processors may work in parallel reducing the degradation due to indirection

This architecture requires the back-end controller to know the mapping between a relation's page table and the page-table processors This can be accomplished by a simple transformation such as hashing or by caching this information in the memory of the back-end controller

3.2.2 Avoiding Indirection

Two techniques avoid indirection altogether

- Version Selection
- Overwriting

3.2.2.1 Version Selection

One way to avoid indirection through a page-table is to alternately use two physically adjacent blocks on a disk for holding the original (shadow) and updated (current) copies of data pages and retrieve both the copies in response to a read request A version selection algorithm is then applied to decide which is the current copy The current copy may be differentiated from the shadow copy by storing a timestamp with every updated page This

† UNIX is a trademark of AT&T Bell Laboratories

architecture, based on the idea in [21] is predicated on the basic disk property that because of the relatively long time required for head positioning and rotational delays, accessing an additional disk block on the same track requires only a small amount of additional time It is expected that the cost of retrieving both the current and the shadow copies and doing version selection would be less than the cost of going through a page table to retrieve only the current copy In [1] we show how the recovery from transaction and system failures may be performed using this scheme

The significant price that this architecture pays is that disk space consumption is doubled One possible optimization would be to share a smaller number of blocks among the data pages on a disk track by adding page numbers to the disk blocks and retrieving the whole track in order to select the current version If parallel-access disks (in which all pages on the different tracks of the same cylinder may be read and written in parallel in one disk access) are used for storing data (such as proposed by the SURE [16] and DBC [5] database machine projects), the extra blocks for shadowing may be provided on a per cylinder basis A problem with these solutions is that overflow will occur if a transaction updates more adjacent pages than available spare blocks (requiring special overflow handling code)

3.2.2.2 Overwriting

A second approach for avoiding the cost of indirection through the page table is to maintain separate shadow and current copies of each data page only while the updating transaction is active On transaction completion, the shadow copy is overwritten with the current copy We will present two architectures one requires *no redo* and the other requires *no undo* at the time of recovery from a system crash Both architectures require scratch space on disk which is managed as a ring buffer

No-Redo Architecture Before updating a data page, the original of the page (the shadow) is written in the scratch space A transaction is committed only after all its updates have been written to disk A list of uncommitted transactions is also needed that should survive system crash Recovery essentially requires restoring of shadows from the scratch space

No-Undo Architecture Unlike the No-Redo architecture, the shadow pages are kept in their original location while a transaction is active All updated pages are first written to the scratch space and then the transaction is considered committed Locks are, however, released only after the updated pages have replaced the shadow pages A list of committed transactions whose updates have not yet overwritten the shadows is required to survive system crash

3.3 Differential Files

In the differential file mechanism proposed in [23], there is a read-only base file that remains unchanged until reorganization, and all updates are confined to a read-write differential file It was proposed in [25] that the differential file be decomposed into two files an A file and a D file Thus, each file R can be considered a view, $R = (B \cup A) - D$, where B is the read-only base portion of R Intuitively, additions to R are appended to the A file, and deletions to R are appended to the D file

The major cost overhead of this approach consists of two components (see [2])

1 I/O cost of reading extra pages from the differential files, A and D

2 Extra cpu processing cost as, for example, a simple retrieval gets converted into a set-union and a set-difference operation

While the number of A and D pages which must be read to process a query depends on the frequency of update operations and the frequency with which A and D files are merged with the base file B, the parallelism inherent in a database machine architecture may be exploited to alleviate the cpu overhead of this recovery scheme In [1] we presented parallel algorithms for operations on differential files and assume that the database machine uses these algorithms

4. Performance

We performed a number of simulation experiments to determine the characteristics of our recovery architectures, their impact on database machine performance, and their relative performance Two metrics were used to study the performance of recovery architectures i) average *execution time per page* and ii) average *transaction completion time* The execution time per page is defined to be the time taken by the database machine to execute a given transaction load divided by the total number of data pages processed by the machine, and is a measure of the throughput of the machine

The transaction completion time is defined to be the time from the time that the back-end controller allocates the first cache frame to a transaction to the writing of the last page updated by the transaction to disk

The database machine was assumed to have 25 query processors, 100 4K byte cache frames, and 2 data disks The query processors were modeled after VAX 11/750s, and the data disks were modeled after IBM 3350 disks We also modeled parallel-access data disks as proposed by the SURE [16] and DBC [5] projects On a parallel-access disk, all pages on the different tracks of the same cylinder may be read or written in parallel (with one disk access)

A transaction was modeled by the number of pages it accesses This value was assumed to be a uniform random variable in the range of 1 to 250 pages Both random and sequential reference strings for the transaction were modeled The write set of a transaction was assumed to be a random subset of its read set and was taken to be 20% of the pages read by the transaction We assumed that all the transactions were available at the start of the simulation and each simulation run was stopped after executing 500 transactions (see [1] for a discussion on the stability of the simulation results)

Experiments were performed using the following four configurations

Conventional-Random conventional disks and random transactions,

Parallel-Random parallel-access disks and random transactions,

Conventional-Sequential conventional disks and sequential transactions,

Parallel-Sequential parallel-access disks and sequential transactions

The details of the simulator can be found in [1,3] In the following sections, we will summarize the important results from these experiments

4.1 Logging

The questions that we attempted to answer from the simulation of the logging architecture include

- Effect of logging on the throughput of the database machine?
- When to have more than one log processor?
- Performance of various log processor selection algorithms?
- Effect of the communication medium between the query processors and log processors?
- Effect of routing the log fragments through the disk cache?

4 1.1 Effect on Database machine Performance

Table 1 summarizes the impact of logging (the unit of time throughout this paper is milliseconds) assuming only one log processor

With our logging architecture, assembly of log fragments into log pages and writing them to the log disk is overlapped with the processing of data pages and, therefore, does not affect the throughput of the database machine The effect of logging manifests itself in two ways First, some updated pages are blocked in the cache for the corresponding log records to be written This causes the transaction completion times to increase The throughput in terms of the time taken by the machine to execute a given transaction load, however, is not affected as the blocking of updated pages in the cache does not cause the disks or the processors to become idle While the blocked pages may hinder anticipatory reading of other data pages, this will happen only if cache frames are scarce and the blocked pages are large in number In our experiments, more cache frames were available for anticipatory paging than the disks could feed, and on average, there were less than 5 pages in the cache waiting for their log records to be written to the log disk

The second effect of logging is that extra query processor time is required to construct log fragments We found in an earlier simulation of the bare (without provision for recovery) database machine [3] that, except for the parallel-sequential configuration, the query processors were very poorly utilized Thus, the overhead of constructing the log fragments did not increase processor utilization significantly as the extra work required was absorbed by the slack capacity of the database machine

4.1 2 Number of Log Processors and Log Processor Selection

The most striking result from these experiments is the poor utilization of even one log disk as shown in Table 2 The rate at which the query processors update pages and hence create log fragments is just not fast enough even to keep a single log disk busy We showed in [3] that the I/O bandwidth between the data disks and the disk cache severely limits the

Configuration	Execution Time per Page (in ms)		Transaction Completion Time (in ms)	
	Without Log	With Log	Without Log	With Log
Conventional-Random	18 0	17 9	7398 4	7543 2
Parallel-Random	16 6	16 5	6476 0	6649 9
Conventional-Sequential	11 0	11 4	4016 5	4333 5
Parallel-Sequential	1 9	2 0	758 1	862 2

Table 1 Impact of Logging

rate at which the query processors update the data pages

From our simulation of the bare database machine [3] we had found that the utilization of the query processors was quite high when the parallel-access disks were used to process sequential transactions We, therefore, simulated the data base machine with 75 query processors, 2 parallel-access data disks, and 150 cache frames for the sequential transactions The log disk was still assumed to be a conventional disk However, instead of logical logging, the physical logging was modeled In physical logging, for each updated page, two log pages are written one contains the before image and the other contains the after image of the updated page The results of the experiment are summarized in Table 3

The performance of the database machine degraded considerably with physical logging, and using more than one log disk improved performance significantly When only one log disk is used, it becomes the bottleneck Consequently, the log pages wait for a long time in the log-disk queue before they are written to the log This results in an increase in the number of updated pages waiting in the cache for the corresponding log pages to be written On the average, 129 frames out of 150 cache frames were occupied by updated pages waiting for their log records to be written Thus, only 21 frames were available for reading new data pages Availability of fewer cache frames severely affects the performance of the parallel-access disks When database changes are not logged, 5,849 data disk accesses were made With only one log processor, a total of 25,993 data disk accesses were required Furthermore, with logical logging when a log page is written all the corresponding updated data pages are moved to the data disk queue at the same time If they belong to the same cylinder, they may be written to disk in one I/O With physical logging, only one updated data page at a time is transferred to the data disk queue This results in a significant decrease in the effectiveness of the parallel access drives

Amongst the log processor selection algorithms, the performance of the cyclic, random, and QpNo mod TotLp selection algorithms are comparable while the TranNo mod TotLp selection algorithm is a loser These results indicate that unless there are a large number of transactions running concurrently, the transaction number mod total log processors selection causes congestion at some log processor while the other log processors are idle

4.1.3 Connection Between the Query and the Log Processors

To explore the effect of the medium connecting the log processors to the query processors, we performed two sets of experiments First we assumed a separate interconnection network between the query processors and the log processors, distinct from the interconnection network between the query processors and the cache, devoted to the task of transmitting log fragments Experiments were performed for three different values for the effective bandwidth of the interconnection medium 1 0, 0 1, and 0 01 megabytes/second In the second set of experiments, no separate interconnection was assumed and the log fragments were routed through the disk cache

The performance of the database machine was found to be quite insensitive to the bandwidth between the query and log processors A log

Configuration	Log Disk Utilization
Conventional-Random	0 02
Parallel-Random	0 02
Conventional-Sequential	0 02
Parallel-Sequential	0 13

Table 2 Log Characteristics (one log processor)

No of Log Disks	Execution Time per Page (in ms)				Transaction Completion Time (in ms)			
	cyclic	random	QpNo mod TotLp	TranNo mod TotLp	cyclic	random	QpNo mod TotLp	TranNo mod TotLp
1	5 1	5 1	5 1	5 1	4518 1	4518 1	4518 1	4518 1
2	2 5	2 6	2 6	2 7	1999 5	2104 3	2232 0	2165 4
3	1 7	1 8	1 8	2 1	1078 9	1137 2	1135 7	1381 8
4	1 5	1 5	1 5	2 0	830 7	854 6	837 8	1137 5
5	1 3	1 4	1 3	2 0	716 3	741 7	714 1	1128 4
w/o logging	0 9	0 9	0 9	0 9	430 6	430 6	430 6	430 6

Table 3 Performance of Parallel Logging and Log Processor Selection Algorithms
(75 query processors, 2 parallel-access disks, and 150 cache frames)

processor assembles the log fragments received from different query processors into a log page, and when a log page is filled up, it is written to the log disk Thus, normally a log fragment must wait in the log processor buffer before it is written to the log disk The reduced bandwidth of the interconnection medium increases the transmission time thus resulting in an increase in the average fragment waiting time This in turn causes the average the number of updated pages waiting for their log records to be written to increase However, since there is a time gap between arrivals of fragments at the log processor, the delay in the arrival of a log fragment is absorbed in the interarrival gap and the fragment waiting time increases only marginally

The performance of the database machine was not affected even when the log fragments were routed through the disk cache Routing the fragments through the cache causes query processor utilization to increase and results in some cache frames being used to hold the in-transit fragments However, the query processors or the number of cache frames are not the constraining factors for the performance of the database machine

4.2 Shadow

We performed a number of simulation experiments to investigate the following issues

- Effect of the shadow mechanism on the throughput of the database machine?
- When is it worthwhile having more than one page-table processor?
- Effect of the size of the page-table buffer?
- What if the logically adjacent pages are not physically adjacent on data disk as the result of the shadow mechanism?
- When do the version selection and the overwriting architectures become desirable?

We assumed that the page-tables were maintained on conventional disks and a page-buffer of size 10 (separate from the disk cache for data pages) was available for buffering page-table pages This buffer was managed using an LRU policy

4.2.1 Effect on Database machine Performance

Table 4 summarizes the effect of the shadow mechanism when 1 and 2 page-table processors are used The results are very different for the random and the sequential transactions

Configuration	Execution Time per Page (in ms)			Transaction Completion Time (in ms)		
	Bare Machine	1 PageTable Processor	2 PageTable Processors	Bare Machine	1 PageTable Processor	2 PageTable Processors
Conventional-Random	18 00	20 51	17 99	7398 41	8367 19	7758 92
Parallel-Random	16 62	20 49	16 69	6476 04	8352 91	6962 23
Conventional-Sequential	11 01	10 98	10 99	4016 46	4066 86	4061 19
Parallel-Sequential	1 92	1 94	1 93	758 06	829 34	816 29

Table 4 Impact of the Shadow Mechanism

Random Transactions With 1 page-table processor, the database machine performance degrades as the page-table processor becomes the bottleneck and the average utilization of the data disks decreases from 100% to 85% (Table 5) The degradation, however, is ameliorated because the page table accesses and the processing of data pages is pipelined While a data page is being read from the disk and being processed by a query processor, the page-table processor fetches the disk-address of the next data page When 2 page-table processors are used, the page-table processors are no longer the bottleneck and the performance of the database machine again becomes limited by the I/O bandwidth between the disk and the cache

Sequential Transactions For 4096 byte page-table pages, more than 1000 page-table entries are contained in one page Since our transactions access at most 250 data pages, at most 2 page-table pages will have to be accessed to get all the disk addresses Therefore, the throughput is not affected at all due to the shadow mechanism as the small amount of time spent in reading and updating of page-table entries is completely overlapped with the processing of data pages

A consequence of using the shadow mechanism is that logically adjacent pages may not stay physically adjacent resulting in disk seeks even though the accesses are logically sequential A crucial assumption in these experiments was that the logically adjacent pages are somehow kept physically clustered within a cylinder Later on, we will present the results of experiments to examine the effects of this assumption

4.2.2 Size of the Page-Table Buffer

The degradation in the throughput of the database machine due to the shadow mechanism for the random transactions may be annulled by choosing a suitably large page-table buffer, even when only 1 page-table processor is used (Table 6) With an increase in the buffer size, the number of page-table pages that are accessed from disk decreases substantially due to page hits Also, the number of page-table pages that are reread for updating due to the buffer-size constraint at the time of committing a transaction also decreases

4.2.3 Logically Adjacent Pages not Physically Clustered

Table 7 shows that if the logically adjacent pages are not kept physically clustered but are scrambled all over the data disk due to the shadow mechanism, the performance of the database machine degrades very significantly for the sequential transactions The average time to access a data page increases by a factor of more than 2 as the logically adjacent pages are physically scattered In addition, for parallel-access disks, this scattering causes the number of disk accesses to increase considerably as the logically sequential pages can no longer be fetched in one disk access The adverse usage of the data disks results in a dramatic degradation of the performance

4.2.4 Overwriting

Table 7 also shows the performance of the no-undo version of the overwriting architecture Recall that in the overwriting architecture, a current copy separate from the original (shadow) is kept only while the transaction is active Once the transaction commits, the shadow is overwritten with the current copy Thus, the overwriting maintains

Configuration	Bare Machine	1 PageTable Disk		2 PageTable Disks	
	Data Disk	PageTable Disk	Data Disk	PageTable Disk	Data Disk
Conventional-Random	0 99	1 00	0 86	0 60	0 99
Parallel-Random	1 00	1 00	0 85	0 64	1 00
Conventional-Sequential	0 75	0 06	0 75	0 03	0 75
Parallel-Sequential	0 92	0 34	0 90	0 16	0 91

Table 5 Average Utilization of Data and Page-Table Disks

Data Disk Type	Bare Machine	Page-Table Buffer Size		
		10	25	50
Conventional	18 00	20 51	18 02	18 01
Parallel-access	16 62	20 49	17 18	16 70

Table 6 Execution Time per Page (Random Transactions, 1 Page-Table Processor)

Data Disk Type	Bare Machine	Clustered (thru PageTable)	Scrambled (thru PageTable)	Overwriting
Conventional	11 01	10 98	20 74	24 08
Parallel-access	1 92	1 94	18 54	2 31

Table 7 Execution Time per Page (Sequential Transactions, 1 Page-Table Processor)

the correspondence between the physical and logical sequentiality and also makes the page-table required with the canonical shadow mechanism redundant

The performance of the overwriting architecture is very different for the conventional disks and for the parallel-access disks With the conventional data disks, the overwriting performs much worse than the "thru page-table" shadow architecture even when the logically adjacent pages are scattered The reason for the poor performance of the overwriting algorithm is the large increase in the number of disk I/Os Also, the average access time increases due to the movement of the disk arm between the scratch area and the data area during the overwriting of the shadows

With the parallel-access disks, however, after a transaction completes, all the updated pages may be read from the scratch area potentially in one access Similarly, all the shadows may also be overwritten in one or very few accesses An additional advantage is that after the shadows have been overwritten, as many cache frames as the pages updated by a transaction will become free at the same time, and the number of data pages fetched in parallel in the next access will increase

We also experimented with the overwriting architecture for random transactions The results of the experiment are summarized in Table 8 Compared to the "thru page-table" architecture, the overwriting architecture has poorer performance due to a higher number of disk I/Os With the parallel-access disks, the updated pages can be read from the scratch area in one access While these updated pages can be overwritten in a minimum number of accesses in the case of sequential transactions, a separate access is required for each page updated by a random transaction

Data Disk Type	Bare Machine	thru PageTable	Overwriting
Conventional	18 00	20 51	26 94
Parallel-access	16 62	20 49	21 65

Table 8 Execution Time per Page with Overwriting for Random Transactions

4.2.5 Version Selection

The version selection architecture to avoid indirection through the page-table is not appropriate from a performance view-point We have shown that for random transactions, either by using a large page-table buffer with 1 page-table processor or by using 2 page-table processors, the page-table accesses can be completely overlapped with the processing of data pages The version selection approach requires that for reading a data page, all the versions of the page be fetched and then a version selection algorithm be applied Thus, unless the disk heads are augmented with enough intelligence to perform "on-the-fly" version selection, the average time to access a data page will increase Since the performance of our database machine architecture is limited by the I/O bandwidth, the version selection algorithm will have poor performance The "thru page-table" architecture performs poorly with the sequential transactions if the logically adjacent pages get scattered The logically adjacent pages may be kept physically clustered if one is prepared to pay the disk storage penalty, and in that case, the "thru page-table" architecture performs well The version selection architecture requires substantial redundant storage to hold versions, and hence for sequential transactions also, the "thru page-table" architecture is preferable

4.3 Differential File

We assume that the number of pages of the A and D differential files accessed by a transaction is a function of the size of differential files relative to the size of the base file, B, and take this size to be 10% Furthermore, we assume that if a page is updated, 10% of the tuples on the page are updated

Recall that in the differential file architecture, the only effect of the tuples in the D file is to eliminate some of the tuples that are otherwise in the result of a query We, therefore, also modeled an optimization in the query processing strategy The query processors first make a scan over the B (or A) page, and the set-difference with the tuples in D pages is taken only if this scan yields at least one result tuple

We performed a number of experiments with the differential-file architecture to explore the following issues

- Effect of the differential file mechanism on the execution time per page?
- Effect of the fraction of the output page that is created when a data page is updated?
- Effect of the size of the differential files, A and D?

4.3.1 Effect on Database machine Performance

Table 9 shows the effect of the differential file mechanism on database machine performance The results have been presented both for the *basic* query processing approach wherein a set-difference operation is performed on every page of the B and A files, and the *optimal* approach wherein the set-difference is taken only on those pages of B and A files that have at least one tuple that satisfies the qualification in the query

With the basic approach, the differential file recovery mechanism results in a significant degradation in performance The execution time per page is almost the same for all the four configurations The cause of this degradation is that, unlike the bare machine, the performance is not limited by the I/O bandwidth between the disk drives and the cache, but rather by the query processors Each of the 25 query processors were almost 100% utilized, whereas the utilization of the disk drives was very low

Compared to the basic approach, the optimal approach reduces the degradation in performance as the set-difference operation is performed on significantly fewer pages For random transactions, the query processors are no longer the bottleneck and the performance was again bound by the I/O bandwidth available from the disk drives, as evidenced by almost 100% utilization of the disk drives and about 60% utilization of each of the query processors Compared to the bare machine,

Configuration	Execution Time per Page (in ms)			Transaction Completion Time (in ms)		
	Bare Machine	Basic Approach	Optimal Approach	Bare Machine	Basic Approach	Optimal Approach
Conventional-Random	18 0	37 8	19 2	7398 4	11589 8	6634 3
Parallel-Random	16 6	37 7	18 0	6476 0	11565 1	6207 6
Conventional-Sequential	11 0	37 6	17 8	4016 5	11443 7	5795 5
Parallel-Sequential	1 9	37 6	13 9	758 1	11368 8	4573 5

Table 9 Impact of the Differential File Mechanism

the execution times per page are higher as the result of a slightly higher number of disk operations caused by the extra accesses to the differential file pages

For sequential transactions with the parallel-access disks all the query processors were almost 100% utilized whereas the disks were busy only 50% of the time With the conventional disks, the query processors were about 75% utilized and the disks were 58% utilized In the conventional-sequential configuration, there were instances when a disk drive was idle and query processors were busy as many set-difference operations were in progress and no cache frames were free, and at times, some query processors were forced to be idle as the data pages were not available from disk On the other hand, in the parallel-sequential configuration, the query processors were not able to keep up with the rate at which the data pages were available from disk Both for the conventional as well as the parallel-access disks, the execution times per page for the sequential transactions are much poorer than the bare machine

There was a substantial increase in the number of disk accesses due to extra differential file pages However, this increase was somewhat ameliorated by the reduced number of updated pages Recall that we have assumed that, on the average, only 10% of the output page is created when a page is updated Thus, if a transaction accesses N pages and updates u% of them, then with the differential file approach, potentially (N*u% - 0 1*N*u%) fewer updated pages will be written to disk In practice, however, the decrease in the number of updated pages will be less than the value given by the above expression as the result of the fragmentation of output pages

4.3.2 Fraction of the Output Page

Table 10 shows the effect of assuming that a larger fraction of an output page is created when a data page is updated (assuming the optimal query processing strategy) The number of output pages does not increase linearly with an increase in the output fraction As alluded to earlier, this small increase is explained by the page-fragmentation with the smaller output fractions

With the query-processor allocation strategy that we implemented, if a page P of transaction T is available and a query processor QP1 is free, then P is immediately assigned to QP1 even though QP1 may not have yet updated any page of T To reduce the amount of fragmentation, it might be advantageous to allow QP1 to remain idle if there is another processor QP2 which is about to become free such that QP2 already has a partially filled output page corresponding to T In [7] several query-processor allocation strategies for multiprocessor database machines have been considered Examining query-processor allocation strategies that minimize fragmentation appears to be an interesting research topic

4.3.3 Size of Differential Files

Table 11 shows that performance degrades nonlinearly in all four configurations as the size of the differential files increases With higher values for the size of the differential files, utilization of both the disk drives and the query processors starts decreasing as the query processors become idle during the time the D pages referenced by a transaction are being read from disk, and a disk

Configuration	Bare Machine	Fraction of Output Page		
		10%	20%	50%
Conventional-Random	18 0	19 2	19 2	20 3
Parallel-Random	16 6	18 0	18 0	18 9
Conventional-Sequential	11 0	17 8	17 9	17 8
Parallel-Sequential	1 9	13 9	13 9	13 6

Table 10 Effect of Output Fraction on Execution Time per Page

drive becomes idle after reading a B or A page into the cache while the query processors are performing the set-difference operations

In order to minimize the size of the differential relations, the differential relations will have to be frequently merged with the base relation In our simulation, we have not modeled the effect of merging of differential relations with the base relation Given the generally poor performance of this recovery mechanism, we did not feel that it was worthwhile exploring the cost of this operation

5. Comparison of the Recovery Architectures and Conclusions

Table 12 shows the comparative performance of the recovery architectures that we considered in terms of their impact on the average execution time per page of the database machine We have only shown the execution times per page in this Table and have omitted transaction completion times as they display the same relative trends

The bare database machine consisted of 25 query processors, 100 cache frames, and 2 data disks The logging results assume only one log disk and an effective bandwidth of 1 megabyte per second between the query processors and the log processor The results for the "thru page-table" shadow architecture have been presented for i) one page-table processor and a page-table buffer capable of holding 10 page-table pages, ii) one page-table processor and a page-table buffer of 50 page-table pages, and iii) two page-table processors and a page-table buffer of 10 pages It is assumed that in each of these three cases that the logically adjacent pages may be kept physically clustered The numbers under the scrambled column correspond to the case when this assumption is not satisfied, and the logically adjacent pages are scattered across the disk For this case, we have also assumed one page-table processor and a page-table buffer of 10 pages The results of using the overwriting architecture, wherein on transaction completion the shadows are overwritten with the current copies, are summarized under the overwriting column For the differential-file architecture, the results have been presented assuming that the size of the differential files is 10% of the size of the base file

The differential-file architecture degrades the throughput of the database machine even when the size of the differential files was assumed to be only 10% of the base file The degradation increases nonlinearly with an increase in the size of the differential files The degradation in the throughput is due to the extra disk I/Os required to access the differential file pages and the extra processing cycles required for the set-difference operation Since the I/O bandwidth available from the disk drives is the factor limiting the throughput of the bare database machine, the extra disk accesses have a negative impact The extra processing requirement would not be a problem as long as the query processors do not become the bottleneck However, when the size of differential files is larger than 10%, (and, in the case of sequential transactions, even for 10% size) the query processors become saturated and adversely affect the throughput of the database machine

In the case of the "thru page-table" shadow architecture, for random transactions, the throughput degrades somewhat when 1 page-table processor is used with a page-table buffer of 10 pages However by increasing the buffer size to 50 or by using 2 page-table processors, the reading and updating of page-table entries may be overlapped with the processing of data pages and there is virtually no degradation in the performance For sequential transactions, if it is assumed that the

Configuration	Bare Machine	Size of Differential Files		
		10%	15%	20%
Conventional-Random	18 0	19 2	24 8	37 0
Parallel-Random	16 6	18 0	24 4	37 0
Conventional-Sequential	11 0	17 8	25 8	39 6
Parallel-Sequential	1 9	13 9	23 5	36 4

Table 11 Effect of Size of Differential Files on Execution Time per Page

Configuration	Bare Machine	Logging	Shadow					Differential File
		1 log disk	1 PageTable Processor		2 PageTable Processors	Scrambled	Overwriting	
			buffer=10	buffer=50				
Conventional-Random	18 0	17 9	20 5	18 0	18 0	20 5	26 9	19 2
Parallel-Random	16 6	16 5	20 5	16 7	16 7	20 5	21 6	18 0
Conventional-Sequential	11 0	11 4	11 0	11 0	11 0	20 7	24 1	17 8
Parallel-Sequential	1 9	2 0	1 9	1 9	1 9	18 5	2 3	13 9

Table 12 Average Execution Time per Page (in ms)

logically adjacent pages may be kept physically clustered, then the performance of the "thru page-table" architecture is very good In practice, this assumption is difficult to justify, and if the physical clustering of the logically sequential pages cannot be maintained, then this architecture performs very poorly for sequential transactions The poor performance is due to the relatively large seek times The overwriting architecture maintains the correspondence between the physical and logical sequentiality and avoids the need of indirection through a page table However, the overwriting architecture performs poorer than the "thru page-table" shadow architecture for random transactions and also in the case of sequential transactions when the data disks used are conventional disk drives The reason for the poorer performance is that extra accesses to the data disks are required with the overwriting architecture Whereas accesses to the page-table disk in the "thru page-table" shadow architecture may be overlapped with the processing of data pages, the overwriting architecture is not amenable to such overlapping When parallel-access disks are used in the processing of sequential transactions, however, current copies may be read from the scratch area and the shadow copies may be overwritten in very few disk I/Os, and hence, the overwriting architecture has quite good performance

Overall, the parallel logging emerges as the best recovery architecture as the collection of the recovery data may be completely overlapped with the processing of data pages We have seen that the communication medium between the query processors and the log processor has no significant effect on the performance of logging In particular, the performance of the logging was not degraded when the log pages were routed through the disk cache, and hence, an interconnection dedicated to communicating the log pages between the query processors and the log processor is not necessary Therefore, the parallel logging can be implemented with no modification in the hardware architecture

by simply designating one or more query processors as log processors and supplementing them with the log disks It was shown that the rate of processing of data pages is not fast enough to warrant more than one log disk However, if the data processing rates improve in the future by solving the problem of I/O bandwidth available from the mass-storage devices, then logging can still be performed in parallel by using more than one log disk and our parallel logging algorithm

6. Acknowledgements

We wish to thank Rudd Canaday for his suggestions and support This research was partially funded by the Department of Energy under contract #DE-AC02-81ER10920 and the National Science Foundation under grant MCS82-01870

References

1 R Agrawal, Concurrency Control and Recovery in Multiprocessor Database Machines Design and Performance Evaluation, Computer Sciences Tech Rep #510, Univ Wisconsin, Madison, Sept 1983 Ph D Dissertation

2 R Agrawal and D J DeWitt, Integrated Concurrency Control and Recovery Mechanisms Design and Performance Evaluation, Computer Sciences Tech Rep #497, Univ Wisconsin, Madison, March 1983

3 R Agrawal and D J DeWitt, Whither Hundreds of Processors in a Database Machine, *Proc Int'l Workshop on High-Level Computer Architecture 84*, May 1984, 6 21-6 32

4 R Agrawal, A Parallel Logging Algorithm for Multiprocessor Database Machines, *Proc 4th Int'l Workshop on Database Machines*, March 1985

5 J Banerjee, R I Baum and D K Hsiao, Concepts and Capabilities of a Database Computer, *ACM Trans Database Syst 3*, 4 (Dec 1978), 347-384

6 D Bitton, H Boral, D J DeWitt and W K Wilkinson, Parallel Algorithms for the Execution of Relational Database Operations, *ACM Trans Database Syst 8*, 3 (Sept 1983), 324-353

7 H Boral and D J DeWitt, Processor Allocation Strategies for Multiprocessor Database Machines, *ACM Trans Database Syst 6*, 2 (June 1981), 227-254

8 A F Cardenas, F Alavian and A Avizienis, Performance of Recovery Architectures in Parallel Associative Database Processors, *ACM Trans Database Syst 8*, 3 (Sept 1983), 291-323

9 D J DeWitt, DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems, *IEEE Trans Computers C-28*, 6 (June 1979), 395-406

10 D J DeWitt and P Hawthorn, A Performance Evaluation of Database Machine Architectures, *Proc 7th Int'l Conf on Very Large Data Bases*, Sept 1981

11 J N Gray, Notes on Database Operating Systems, in *Lecture Notes in Computer Science 60, Advanced Course on Operating Systems*, R Bayer, R M Graham and G Seegmuller (ed), Springer Verlag, New York, 1978

12 J N Gray, P R McJones, B G Lindsay, M W Blasgen, R A Lorie, T G Price, F Putzolu and I L Traiger, The Recovery Manager of the System R Database Manager, *ACM Computing Surveys 13*, 2 (June 1981), 223-242

13 P Hawthorn and D J DeWitt, Performance Analysis of Alternative Database Machine Architectures, *IEEE Trans Software Eng SE-8*, 1 (Jan 1982), 61-75

14 W Hell, RDBM - A Relational Database Machine Architecture and Hardware Design, *Proc 6th Workshop on Computer Architecture for Non-Numeric Processing*, June 1981

15 B Lampson and H Sturgis, Crash Recovery in a Distributed Data Storage System, Computer Science Lab , Xerox PARC, 1979

16 H O Leilich, G Stiege and H C Zeidler, A Search Processor for Data Base Management Systems, *Proc 4th Int'l Conf on Very Large Data Bases*, Sept 1978, 280-287

17 B G Lindsay, P G Selinger, C Galtieri, J N Gray, R A Lorie, T G Price, F Putzolu

and B W Wade, Notes on Distributed Databases, Rep RJ2571, IBM Research Lab , San Jose, California, July 1979

18 R A Lorie, Physical Integrity in a Large Segmented Database, *ACM Trans Database Syst 2*, 1 (March 1977), 91-104

19 S E Madnick, The INFOPLEX Database Computer Concepts and Directions, *Proc IEEE Computer Conf*, Feb 1979

20 M Missikoff, An Overview of the Project DBMAC for a relational machine, *Proc 6th Workshop on Computer Architecture for Non-Numeric Processing*, June 1981

21 A Reuter, A Fast Transaction Oriented Logging Scheme for Undo Recovery, *IEEE Trans Software Eng SE-6*, 4 (July 1980), 348-356

22 S A Schuster, H B Nguyen, E A Ozkarahan and K C Smith, RAP 2 - An Associative Processor for Data Bases and its Applications, *IEEE Trans Computers C-28*, 6 (June 1979),

23 D G Severance and G M Lohman, Differential Files Their Application to the Maintenance of Large Databases, *ACM Trans Database Syst 1*, 3 (Sept 1976), 256-267

24 S W Song, A Survey and Taxonomy of Database Machines, *IEEE Database Engineering Bulletin 4*, 2 (Dec 1981), 3-13

25 M R Stonebraker, Hypothetical Data Bases as Views, *Proc ACM-SIGMOD 1981 Int'l Conf on Management of Data*, May 1981, 224-229