MaStA - An I/O Cost Model for Database Crash Recovery Mechanisms

S. Scheuerl†, R.C.H. Connor†, R. Morrison†, J.E.B. Moss¥ & D.S. Munro†

†School of Mathematical and Computational Sciences,
University of St Andrews,
North Haugh, St Andrews, Fife, KY16 9SS, Scotland
Email: {stephan, richard, ron, dave}@dcs.st-and.ac.uk

¥Department of Computer Science, University of Massachusetts,
Amherst, Massachusetts, MA 01003, U.S.A.
Email: moss@cs.umass.edu

Abstract

Crash recovery in database systems aims to provide an acceptable level of protection from failure at a given engineering cost. A large number of recovery mechanisms are known, and have been compared both analytically and empirically. However, recent trends in computer hardware present different engineering tradeoffs in the design of recovery mechanisms. In particular, the comparative improvement in the speed of processors over disks suggests that disk I/O activity is the dominant expense. Furthermore, the improvement of disk transfer time relative to seek time has made patterns of disk access more significant. The contribution of the MaStA (**Ma**ssachusetts **St A**ndrews) cost model is that it is structured independently of machine architectures and application workloads. It determines costs in terms of I/O categories, access patterns and application workload parameters. The main features of the model are:

- Cost is based upon a probabilistic estimation of disk activity, broken down into sequential, asynchronous, clustered synchronous, and unclustered synchronous disk accesses for each recovery scheme.
- The model may be calibrated by different disk performance characteristics, either measured by experiment or predicted by analysis.
- The model may be used over a wide variety of workloads, including those typical of object-oriented and database programming systems.

The paper contains a full description of the model and illustrates its utility by analysing four recovery mechanisms, delivering performance predictions for these mechanisms when used for some specific workloads and execution platforms. The refinement of I/O cost into the various access patterns is shown to give qualitative predictions differing from those of uniform access time models. Further the results are shown to vary qualitatively between two commercially available configurations.

1 Introduction

Recovery management in database systems provides engineering solutions to failure, offering a required degree of reliability by automatically restoring a system to a coherent and acceptable state. Conceptually the user manipulates data through reads

and writes to a database implemented on non-volatile storage, with volatile storage being used as a cache. Recovery is required because by definition volatile cache contents are lost when the system crashes. We do not specifically address failures of non-volatile storage. When failure occurs the user should not be left with an inconsistent state in the non-volatile store. Recovery mechanisms prevent this by controlling the writes to non-volatile storage so that some high level abstraction of atomicity is maintained. Examples of such an abstraction include: an atomic "save" operation in a text editor; a database atomic transaction; or the commitment of a mutually agreed design in a co-operative working session.

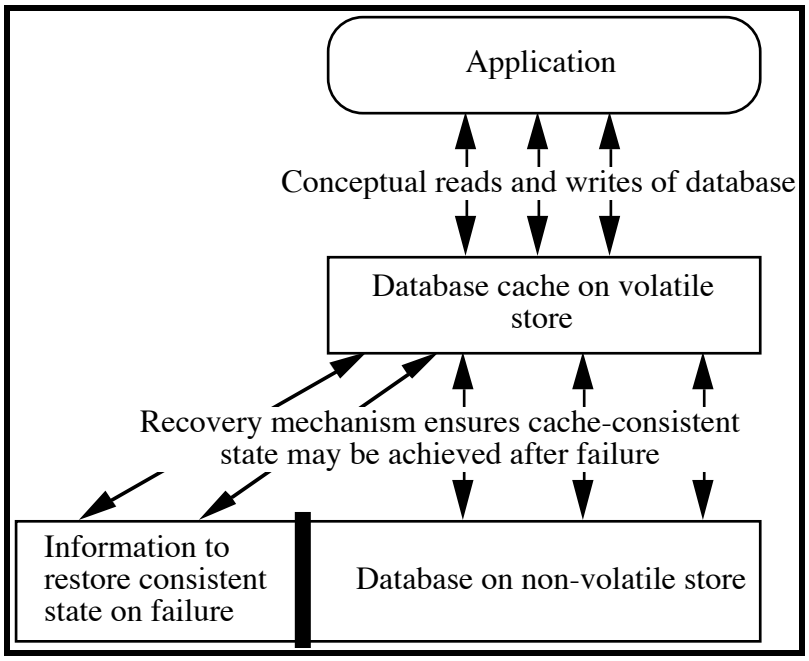

Figure 1: The General Structure of a Recovery Mechanism

Figure 1 shows the general principle behind all recovery mechanisms. The application, although conceptually communicating directly with the database, communicates instead with a database cache implemented on volatile storage. Reads are fetched from, and writes eventually propagated to, non-volatile storage. Non-volatile storage is partitioned into two logical areas: the database itself, and a recovery partition used to record whatever information is necessary for recovery in a given recovery mechanism. Examples of such information are: a log [Gra78]; a shadow page table [Lor77]; and a differential file [Sev82]. In every case the recovery data is maintained so that restart can restore a state consistent with the system's atomicity abstraction.

The cost of recovery mechanisms can be critical to the overall performance of data-intensive applications with I/O bandwidth being a limiting factor. Hence many recovery mechanisms have been invented, each with different performance tradeoffs [Gra78, Lor77, Sev82]. Each technique's costs involves not only the overhead of restoring data after failures but also the time and space overhead required to maintain sufficient recovery information during normal operation to ensure recovery. Under different workloads and configurations these crash recovery mechanisms exhibit different costs.

Modern trends in hardware design have given a disproportionate improvement in processor speed compared to disk access time, and within disks themselves a disproportionate improvement in transfer rates compared to seek times. These factors

change the engineering tradeoffs upon which recovery mechanisms are based, and recently designed mechanisms tend to favour extra processor activity to reduce disk I/O, and more asynchronous I/O as opposed to random access [EH84, OS94]. The purpose of the model described here is to provide an analytical framework for comparing recovery mechanisms under a variety of different workloads and configurations.

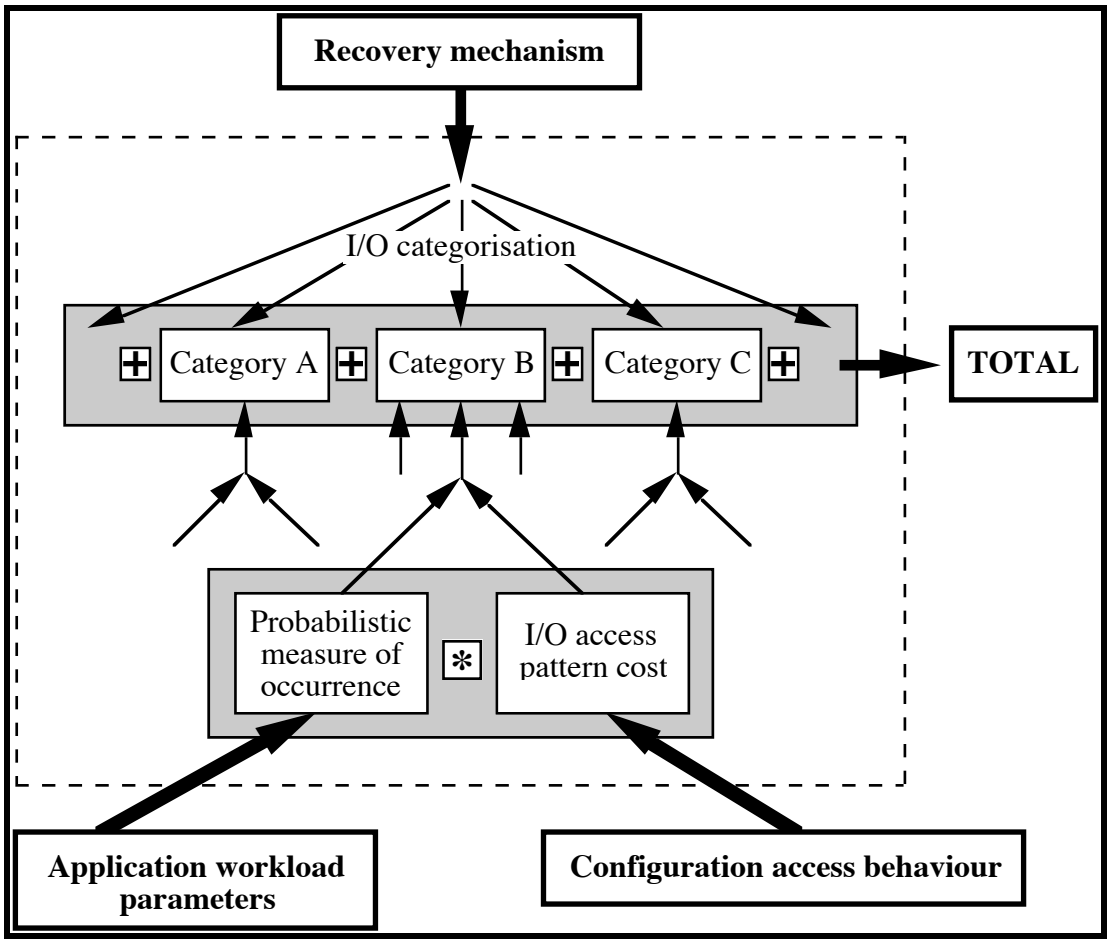

Figure 2: An overview of MaStA

Figure 2 shows a simplified view of the MaStA model. Each recovery mechanism's cost is broken down into constituent independent I/O cost categories, such as data reads or commit writes. The overall cost of a mechanism is the sum of the costs of each category:

$$\text{Total Cost} = \sum_{i} \text{CatCost}(i),\ (i \in \text{Categories})$$

Each category's cost is derived from the number of accesses it incurs of each access pattern:

$$\text{CatCost}(i) = \sum_{j,k} n_{ij} * A_k,\ (j \in \text{Occurrences},\ k \in \text{Access Patterns})$$

For any given medium a set of access patterns is developed such that each pattern is believed to have significantly different costs, such as sequential versus unclustered

synchronous writes. The number of accesses of each pattern is derived from workload parameters, such as page and object size, and the density of objects within pages.

The derivation of a cost estimate for a particular combination of mechanism, configuration and workload is therefore derived by analysing:

- The mechanism: identifying the cost categories, and for each category the access pattern and number of accesses.
- The configuration: determining the average cost of each access pattern experimentally or analytically.
- The workload: measuring and choosing values for the workload parameter.

The identification of these three categories allows the MaStA model to encompass the patterns of usage in both traditional and modern database systems.

2 Recovery Mechanisms

To illustrate the MaStA model, we chose four specific recovery mechanisms: *object logging* and *page logging*, both with deferred updates [Gra78]; *after-image shadow paging* [Lor77, Cha78]; and *before-image shadow paging* [Bro89, BR91]. These mechanisms were chosen because of our familiarity with them [Mun93, Bro89, RHB+90, MS88]. Because we have good intuitions about how these mechanisms perform under varying workloads, we can satisfy ourselves that the model predicts appropriate qualitative behaviours. Furthermore, having implemented some of these mechanisms we have a basis for future empirical validation of the model against actual implementations.

When using a logging mechanism with deferred updates, changes are recorded in a log but updates to non-volatile store may be deferred until commit or even later. The log may be written sequentially (for speed), and may be buffered until just before each transaction commits; group commit can offer further improvement by writing multiple transactions' changes together. Since the log is written to a separate area, database updates do not move database pages, so the original database clustering, be it good or poor, is maintained. Updates must eventually be *installed* in the non-volatile database. Installing an update always requires an *installation write*, though multiple changes to the same page may be merged to produce a single page write. In the case of object logging, installation may also require an *installation read*, to obtain the original version of a disk page into which to merge one or more partial-page updates. Otherwise, object logging and page logging differ only in the granularity of the items logged.

In a shadow paging system a page replacement algorithm controls the movement of pages between volatile and non-volatile store such that recovery will always produce a consistent state. To implement this a disk page table is used to maintain the correspondence between the virtual pages of the database and blocks on non-volatile store; the table may actually exist (after-image shadow paging) or may only be conceptual (before-image shadow paging).

After-image shadow paging writes each updated page to a free block and updates the disk page table to reflect the new mapping. The mechanism maintains a mirrored root block from which the last consistent map can be found. When a transaction commits, the new mappings, in addition to updated data pages, are written to non-volatile store and then the oldest root block is updated atomically. Since after-image shadow paging always writes pages to new disk blocks, the original clustering of the blocks may be lost. Note that reclustering may or may not improve performance; while most

database researchers assume that reclustering degrades performance, log structured file systems have been seen to offer *improved* performance.

In before-image shadow paging the first modification to a page causes the original to be written to a free disk block. Updates are then performed in place. A disk page table is used to record the location of the shadow pages (but not the database pages, since they do not move), and must be present on disk before updates overwrite the original disk blocks. The disk page table can be used to recover the last consistent state of the database. On commit, updated pages are written back to disk and the disk page table is no longer required. Since before-image shadow paging uses an update-in-place policy it maintains the original clustering.

3 Developing the MaStA Cost Model

The structure of the MaStA cost model as depicted in Figure 2. Here that structure is fleshed out, with specific I/O cost categories, access patterns and parameter variables, and applied to the four described recovery mechanisms.

3.1 I/O Categorisation of Recovery Mechanisms

The I/O cost categories used in the MaStA model are:

- **Data reads and writes**: The cost of data reads and writes are included in the model since the presence of a recovery mechanism may change the I/O access patterns of a running system. For this reason MaStA models total I/O costs as opposed to recovery overheads alone. For example, an after-image shadow paging mechanism may be forced to perform clustered and unclustered synchronous reads because of long term declustering.
- **Recovery reads and writes**: The information to provide recoverability typically imposes additional costs such as writing log records in a log based system.
- **Installation reads and writes**: Database recovery mechanisms that defer updates to the database may incur installation I/O costs. For example, an object logging mechanism must copy updated objects from the buffer to the database page containing the object.
- **Commit overhead**: This is the I/O overhead of recording the committed state of a transaction on disk. For example in a logging system this may include writing a transaction commit record to the log.

Within the MaStA model the four described recovery mechanisms discussed incur costs with I/O categories as shown in Table 1.

		Object logging	Page logging	After image SP	Before image SP
Data	**read**	✔	✔	✔	✔
	write	✔	✔	✔	✔
Recovery	**read**			✔	
	write	✔	✔	✔	✔
Installation	**read**	✔			
	write	✔	✔		
Commit	**write**	✔	✔	✔	✔

Table 1: Assigning I/O Cost Categories to Recovery Mechanisms

It can be seen that neither of the shadow paging schemes require installation reads or writes. This is because they do not defer updates past commit time. After-image shadow paging requires recovery reads and writes to maintain the disk page tables; the other mechanisms can use an update-in-place policy with a fixed disk page map. Page logging does not require installation reads because it installs whole pages as opposed to merging objects into pages.

3.2 Disk Access Patterns

The crucial refinement of the MaStA model is to distinguish various I/O access patterns, on the basis of their significantly different costs. The model includes four patterns, called *sequential*, *asynchronous*, *clustered synchronous*, and *unclustered synchronous*, and further breaks each down into reads and writes. The patterns are intended to reflect the characteristics of magnetic disk systems, but the general idea applies to any device whose access time varies according to the sequence of positions accessed. The patterns are defined as follows:

- Sequential reads/writes (r_{seq}, w_{seq}): The data are read/written in sequentially increasing positions. This is the most efficient access pattern because hardware, firmware, and software tend to be tuned specifically to support it well. A typical example is writing to a sequentially structured log.

- Asynchronous reads/writes (r_{asc}, w_{asc}): The system maintains a pool of read/write requests which can be processed in any order. The requests are scheduled in a favourable order, so if the pool is large enough the average cost can approach that of sequential I/O. A typical example is keeping a pool of modified pages requiring installation in the database.

- Clustered synchronous reads/writes (r_{clu}, w_{clu}): This comprises localised accesses that are synchronous and hence cannot be freely scheduled. A typical example is localised reads of database pages for active transactions.

- Unclustered synchronous reads/writes (r_{ucl}, w_{ucl}): These are synchronous accesses that involve moving the access position arbitrarily far. A typical example is forcing the log (if it is stored on the same device as the database), since the database area can be far from the log area.

The costs of the I/O access patterns vary with machine platform as well as database size and storage layout. Given a suitably accurate model of the disk device and associated software, one might derive an analytical or simulation model to determine

the cost of each I/O access pattern. One can also run experiments to measure these values, which is the approach taken here. Note that the cost of an I/O access pattern may also depend on the application workload. For example, the parts of the database accessed determine the locality of clustered I/O, and the size and structure of the pools of scheduled I/O requests. While the costs may remain approximately the same across a range of related workloads they may vary between substantially different workloads.

The refinement of I/O costs to include different access patterns turns out to be significant as will be seen later. The ratio of the cost of the most expensive unclustered synchronous write access pattern to the least, sequential read, was observed to be a factor of six in an actual system.

3.2.1 Assigning I/O Access Patterns

The I/O access patterns for the four recovery mechanisms are given in Table 2.

		Object logging	Page logging	After image SP	Before image SP
Data	**read**	clustered synchronous	clustered synchronous	clustered & unclustered synchronous	clustered synchronous
	write	sequential	sequential	sequential	clustered synchronous
Recovery	**read**			unclustered synchronous	
	write	sequential	sequential	sequential	sequential
Installation	**read**	asynchronous			
	write	asynchronous	asynchronous		
Commit	**write**	sequential (1) & unclustered synchronous(2)	sequential (1) & unclustered synchronous(2)	sequential (1) & unclustered synchronous(2)	sequential (1) & unclustered synchronous(2)

Table 2: I/O Access Pattern Assignments

In object logging data reads are clustered synchronous because the mechanism maintains the initial clustering of blocks. Data writes consist of writing sequential records to the log. Recovery writes are also to the log, so incur sequential write costs. Installation reads may be required when updated objects are copied from the buffer to the database. The pages containing the installed objects are written back to disk using installation writes. Installation I/O can be delayed and therefore are asynchronous. Commit I/O consists of writing a commit record to the log. This is normally written with other log records and so is given a sequential write cost. Writing the commit record may also incur two unclustered synchronous seeks: one to position the device at the log and one to move it back to the database area. The second actually occurs at the beginning of the next data read but is most conveniently modelled here. In accordance with our assumption that main memory is relatively plentiful, we assume that logged changes are retained in volatile store until installed, so that the log installation does not need to read the changes back from the log.

Page logging differs from object logging only in the granularity of the log records. Since the log contains complete updated pages, installation reads are not required.

In after-image shadow paging, updated pages are written to free disk blocks. These shadow blocks can be allocated contiguously, so data writes can be sequential. Through the loss of the original clustering of the pages, data reads may require clustered and unclustered synchronous reads. Recovery reads are required to read the page mapping tables; such reads require unclustered synchronous disk seeks. Recovery writes to write the page tables can be sequential once the disk head is correctly positioned. The cost of this seek is charged to commit I/O. Commit I/O consists of writing the root block and is given the cost of an unclustered synchronous write.

In before-image shadow paging clustering is maintained, so data reads and writes are both clustered synchronous. There are two costs involved in recovery writes. The first is writing before-images to shadow blocks. Shadow blocks can be allocated contiguously and written sequentially. The second cost is writing page table mappings indicating the locations of the shadow copies. These mappings must be written before an original block is overwritten, and therefore consists of unclustered synchronous writes. Commit I/O is as for after-image shadow paging.

3.3 Transaction Workload

The goal of the application workload model is to capture all the parameters that affect I/O. For example, the number of updated pages affects the number of log records or shadow pages written. These parameters are expressed in terms of derived variables which are normalised to produce the number of page I/Os incurred.

The application workload derived variables could be obtained by simulation, measurement of a real system or, as in this case, from a combination of basic variables that decompose workload into more fundamental units. The basic variables are described in Appendix I; the derivation functions are defined in Appendix II; and the method of calculation of the derived variables using the derivation functions is given in Appendix III. Here we concentrate on the derived variables to illustrate the workings of the MaStA model; Table 3 describes the application workload derived variables used to cost the four recovery mechanisms.

Derived Variable	Description
PMiss	the number of data page read misses
PDirty	the number of pages updated
PTMiss	the number of page table page read misses
PTDirty	the number of page table pages updated
PIRead	the number of installation reads
PIWrite	the number of installation writes
PLog	the number of pages written for log records
PolHouse	the number of pages written for log housekeeping information
PpHouse	the number of pages written to record the position of pages in the log
PcommR	the number of log pages written to secure a commit record
PRoot	the number of root pages written to the log to record commit state

Table 3: Workload Derived Variables

The model includes additional variables to take account of implementation details of particular recovery mechanisms. These variables are described below.

3.4 Parameter Determination of Object Logging Mechanisms

The following variables affect the amount of data written to the log when using object logging:

- The average ratio of object size to log record size. Some logging mechanisms may record only updated byte ranges thereby potentially reducing the amount of the data written to the log.
- The average number of log records per updated object. For example, a mechanism which writes a record for every update may write a different amount from a mechanism writing only one record per updated object.
- The size of per-log-record housekeeping data.

3.5 Parameter Determination of Shadow Paging Mechanisms

In shadow paging mechanisms the choice of I/O access patterns used in the I/O categories is influenced by the block allocation strategy used. Allocating new blocks sequentially, for example, may allow the mechanism to take advantage of sequential writes. Other possible allocation strategies include:

- Paired blocks: all blocks are allocated in pairs, so that shadow blocks are allocated adjacent to original blocks.

- Same cylinder: the mechanism tries to allocate shadow blocks in the same cylinder as originals.

- Dynamically clustered: new blocks are chosen from a set of free blocks, allowing some control over clustering.

3.6 The MaStA Cost Model for the Four Recovery Mechanisms

Table 4 shows the cost functions for the four recovery mechanisms. Within each category the I/O cost function is the product of a derived variable and an I/O access pattern cost, or in the case of before-images shadow paging, a sum of two such products.

I/O Categories	Object Logging		Page Logging		After Image Shadow Paging		Before Image Shadow Paging	
	Derived Variable	**Access Pattern**	**Derived Variable**	**Access Pattern**	**Derived Variable**	**Access Pattern**	**Derived Variable**	**Access Pattern**
Data Read	PMiss	r_{clu}	PMiss	r_{clu}	PMiss	r_{ucl}/r_{clu}	PMiss	r_{clu}
Data Write	PLog	w_{seq}	PDirty	w_{seq}	PDirty	w_{seq}	PDirty	w_{clu}
Recovery Read					PTMiss	r_{ucl}		
Recovery Write	PolHouse	w_{seq}	PpHouse	w_{seq}	PTDirty	w_{seq}	PTDirty PDirty	w_{seq} w_{seq}
Installation Read	PIRead	r_{asc}						
Installation Write	PIWrite	w_{asc}	PIWrite	w_{asc}				
commit/ other	PcommR 2	w_{seq} s_{ucl}	PcommR 2	w_{seq} s_{ucl}	PRoot 2	w_{ucl} s_{ucl}	PRoot 1	w_{ucl} s_{ucl}

Table 4: I/O Cost Functions for Four Recovery Mechanisms

An access pattern S_{ucl} is attributed to the commit/other category to indicate that unclustered seek costs are incurred by the mechanisms. Two unclustered seeks are incurred for example by the logging mechanisms to move to the log area and back to the data area when writing to the log.

As an example, when written out, the cost function for object logging is:

$$PMiss * r_{clu} + PLog * w_{seq} + PolHouse * w_{seq} + PIRead * r_{asc} + PIWrite * w_{asc} + PcommR * w_{seq} + 2 * s_{ucl}$$

4 Experimentation and Results

To show the flexibility and utility of the model, several experiments will be described. By experiment we mean supplying values for the variables and predicting the I/O costs of the four schemes.

In the experiments described here the following assumptions are made :-

- main memory is large enough to hold all required log records, page mapping tables and data pages accessed and updated by all running transactions;

- a paged virtual memory system is assumed and hence all mechanisms perform the same number of data reads.

The following recovery mechanism costs are omitted:

- the cost of recovering from a crash;
- the cost of aborting and re-running transactions;
- other costs of concurrency control schemes;
- the influence of multi-programming;
- checkpointing.

4.1 Calibration

Measurements were performed on two platforms to illustrate the platform independence of the MaStA model. From these measurements, values were obtained for the I/O access patterns for each platform. The two platforms were a Sun SPARCStation ELC running SunOS 4.1.3 with 48MB main memory, 500MB CDC Wren V SCSI drive, and a DEC Alpha AXP 3000/600 running OSF/1 V2.0 with 128MB main memory and a 2.1GB Seagate ST12550N (Barracuda II) SCSI drive. The experiments involved block read and write operations on large disk files which spanned the majority of the disk, intending to avoid operating system disk cache effects. The locality of I/O operations was controlled to simulate sequential, asynchronous, and unclustered synchronous I/O. All experiments were performed on a cold single-user system and timings were obtained using the operating systems' *time* commands. A "cold" cache was obtained by reading a large file from another device, forwards and backwards.

Sequential I/O was simulated by performing ordered I/O operations on contiguous blocks of the file. Unclustered synchronous I/O was simulated by choosing at random 10% of the blocks in the file and accessing the blocks in the random order. Asynchronous I/O was simulated by sorting the block numbers used in the unclustered synchronous experiment and then accessing these in order. There was less than 5% variation between runs. Table 5 shows the measured I/O access pattern costs as a ratio to that of sequential reads. It is important to point out that these do not compare the I/O access costs of the Alpha and the SPARCStation but give each machine's I/O access costs as multiples of the cost of a sequential read on that machine. ASR stands for Alpha sequential read and SSR for SPARCStation sequential read.

I/O Access Pattern	Alpha	SPARCStation
Sequential reads	1.0 ASR	1.0 SSR
Sequential writes	1.0 ASR	1.0 SSR
Asynchronous reads	4.0 ASR	1.8 SSR
Asynchronous writes	2.1 ASR	1.5 SSR
Clustered synchronous reads	3.0 ASR	1.5 SSR
Clustered synchronous writes	3.5 ASR	3.5 SSR
Unclustered synchronous reads	5.6 ASR	2.6 SSR
Unclustered synchronous writes	5.7 ASR	5.5 SSR

Table 5: Costs Assigned to I/O Access Patterns

4.2 Applications of the Model

Experiment 1

Experiment 1 considers the relative costs of the recovery mechanisms under a given workload. The I/O access pattern costs used are those of Table 5 with the basic workload variable values of Table 6.

Workload Variables	Description	Values
Oacc	number of objects accessed	10000
Psize	page size	2048 words
Osize	object size	16 words
ObjLoc	object locality within the address range*	0.3
DObjLoc	dirty object locality within pages	varied in the range [0, 1]
Odirt	percentage of objects updated	20%
iread	% of updated pages requiring installation reads	30%
iwrite	% of updated pages requiring installation writes	30%
PTemp	degree of page temporal locality	20%
Ploc	locality of updated pages	20%
LRover	average log record overhead	3 words
MapEntry	mapping entry size	2 words
LRratio	log record/object ratio	100%

Table 6: Basic Workload Variables Values used in Experiment 1

---

* This value can be varied in the range [0, 1]. A value of 1 indicates that the objects are held on the fewest pages possible. A value of 0 indicates that the objects are scattered in the address range and are held on as many pages as possible.

In this experiment all but the DObjLoc basic variable remain constant. In this and subsequent experiments the degree of locality of the objects updated is varied between 0 and 1 thus varying the number of pages dirtied obtained from the derived variable PDirty. The x-axis of the graphs plotted indicates the pages updated as a percentage of the number of pages accessed.

The graphs in Figure 3 plot values for this percentage from 0.19%, the minimum possible to 30% using three sets of I/O access pattern costs.

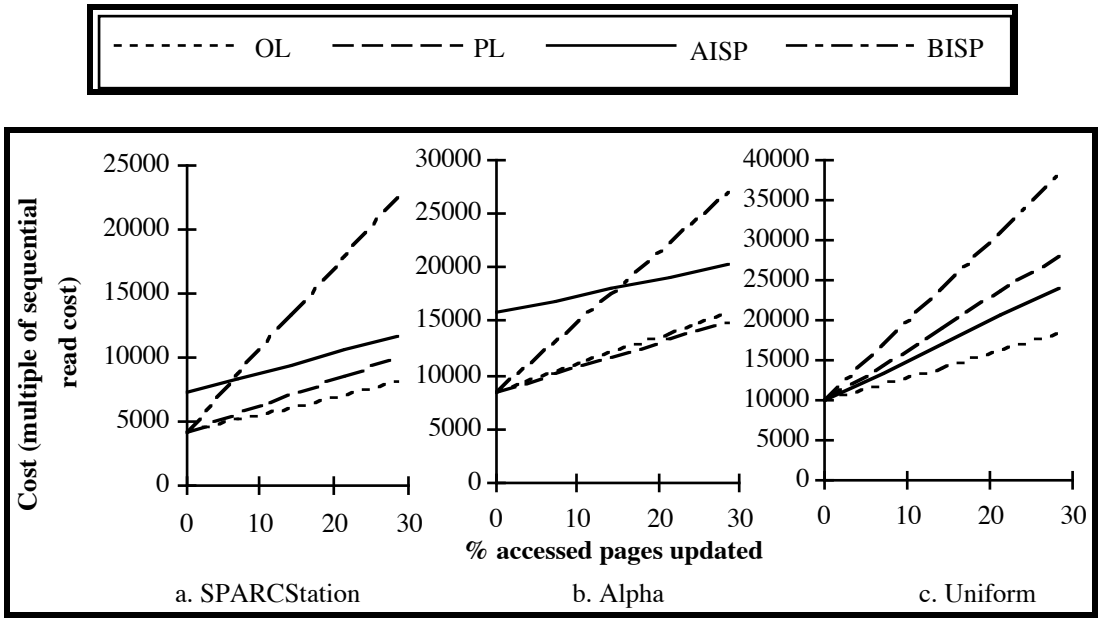

Figure 3: Experiment 1

Graphs 3.a and 3.b show that when the percentage of accessed pages updated is at the minimum, the mechanisms, except for after-image shadow paging, have similar costs. This is because they have the same data read costs, and because at this level of mutation write costs are small relative to read costs. After-image shadow paging has higher I/O costs even at low percentages of updated pages because its data reads are in part unclustered synchronous whilst the other schemes are clustered synchronous.

As would be expected the I/O costs of all the mechanisms increase as the percentage of updated pages increases. The graphs illustrate that the I/O cost of before-image shadow paging increases more rapidly compared to the other mechanisms. This is due to the extra page writes required in this mechanism. Figure 3.c illustrates the relative costs of the recovery mechanisms calculated using a uniform I/O access pattern cost of 3.5 in every I/O cost category. As can be seen the relative positions of the costs of the recovery mechanisms is different from that of graphs 3.a and 3.b, stressing the importance of distinguishing different I/O access patterns.

Experiment 2

This experiment increases the object locality from 30% to 100%, that is the objects accessed are densely packed, and increases the percentage of objects updated from 20% to 60%. The remaining workload variables are unchanged from Experiment 1.

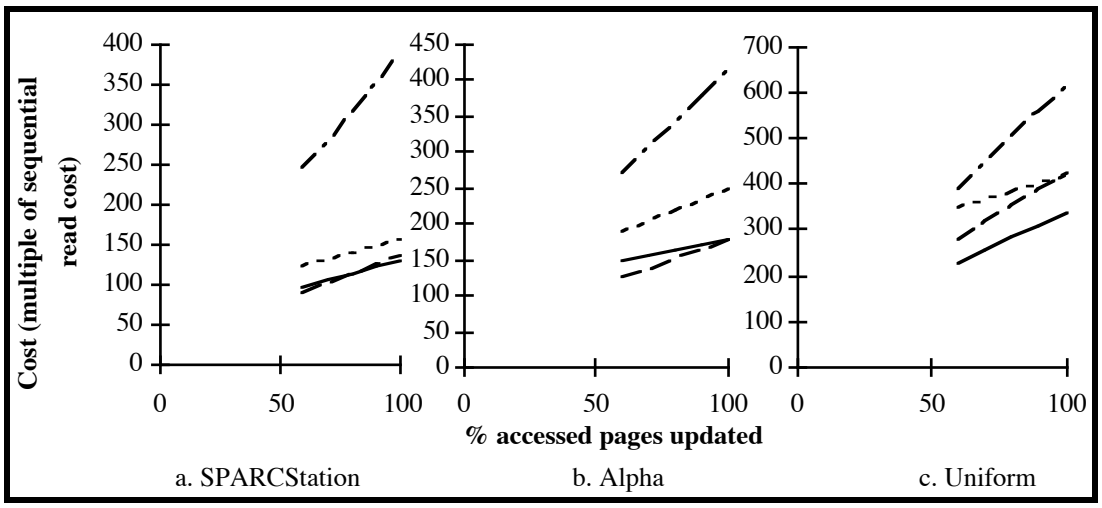

Figure 4: Experiment 2

Varying the application workload causes up to 100% of the accessed pages to be updated. Comparing graphs the 4.a and 4.b shows that after-image shadow paging outperforms the other schemes on the SPARCStation when the percentage of updated pages is higher than 75%. This happens for two reasons. Firstly, with high object locality and a higher percentage of objects updated, the amount of unchanged data written back is reduced in the page based mechanisms. This reduces the advantage of object logging over page based schemes. Secondly, as the number of pages updated increases, page logging costs increase because there are more installation writes. Figure 4.c shows predictions of the uniform I/O cost model, which again are different from the refined models.

Experiment 3

This experiment increases the average object size (Osize) from 16 to 1024 words to show the effect on the mechanisms of accessing large objects. The remaining workload variables, including object locality and the percentage of objects updated, are given the values in Table 9.

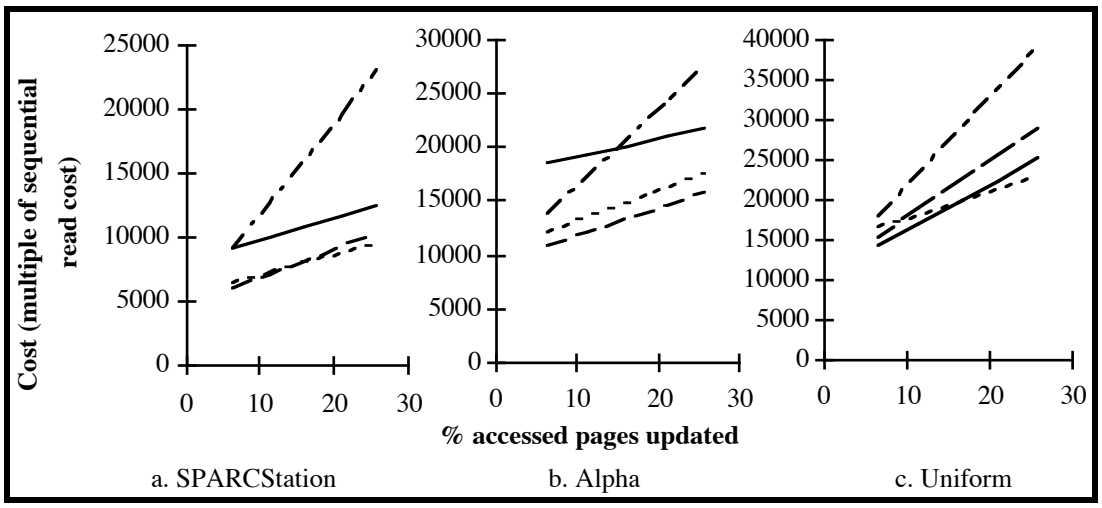

Figure 5: Experiment 3

Graphs 5.a and 5.b show that the minimum percentage of updated pages has increased due to the large object size. Under this workload, page logging has the lowest costs on the Alpha and when low percentages of pages are updated on the SPARCStation. There are two reasons for this: page logging requires no installation reads, and page logging reads are better clustered than those of after-image shadow paging. The difference between the cost of after-image shadow paging and the other mechanisms is more pronounced in graph 5.b than in graphs 5.a again due to the difference in the ratios of the I/O access pattern costs between the two configurations.

These experiments show that relative performance of the recovery mechanisms depends not only on the application workload but also on the platform. They also demonstrate a primary hypothesis of the model: that different I/O access patterns affect the costs strongly enough that they must be modelled.

5 Related Work

A classification of recovery mechanisms is given by Haerder and Reuter [HR83]. They stratify recovery into a hierarchy of propagation strategy, page replacement strategy, end-of-transaction processing and checkpointing strategies adopted by the mechanisms in systems which provide atomic transactions. The propagation strategy is said to be *atomic* if multiple updates to the non-volatile object store can be performed as a unit at end-of-transaction processing or ¬*atomic* if the propagation of updates is interruptable by a system crash. The recovery mechanism's page replacement strategy is described as *steal* if updated pages can be written out to disk during transaction processing or ¬*steal* if all updated pages must remain in main memory at least until end-of-transaction processing. The end-of-transaction processing strategy is designated as being *force if* updated data is propagated to the non-volatile store during end-of-transaction processing and ¬*force* if the propagation can be deferred until after commit time. In this classification the checkpointing strategy is split into four categories depending on the times at which checkpoints can be generated. These are:

- transaction-oriented checkpoints (TOC) which occur every time a transaction commits and is associated with a *force* propagation strategy.

- transaction-consistent checkpoints (TCC) which cause the checkpointing of all update transactions at the same time. In progress update transactions are allowed to terminate and new update transactions are blocked. All updates are then propagated to the object store after which normal execution is resumed.

- action-consistent checkpoints (ACC) which are generated in a similar manner to transaction-consistent checkpoints except they are at an operational level instead of at the level of transaction commit.

- fuzzy checkpoints where only pages which have not been propagated since the last checkpoint are propagated to the store.

Using this classification the object logging and page logging mechanisms described above exhibit (¬atomic, ¬steal, ¬force, …) properties, where the ellipses indicate that checkpointing can be transaction-oriented, transaction consistent or fuzzy. The before-look shadow paging exhibits (¬atomic, steal, force, transaction oriented checkpoint) properties whereas after-look shadow paging is (atomic, steal, force, transaction oriented checkpoint). Indeed, one observation that can be made from this work is that shadow paging is a variant of page based logging differing only on the I/O access patterns required to read and write the data and the recovery data, and to write commit points.

Reuter [Reu84] presents a cost model which uses this classification to analyse and compare the performance of a number of recovery schemes. The model ignores CPU costs using the number of I/O operations as a cost unit. The model computes a number of costs associated with a recovery mechanism taking into account the mean time between failures, the frequency of the checkpointing interval, the probability of abort and the availability of shared pages. Altogether ten recovery schemes are analysed and compared. These schemes are split into three groups with the following properties:

page-level logging				
¬atomic	steal	¬force	TCC	(only at system shutdown)
¬atomic	steal	¬force	ACC	(at regular intervals)
¬atomic	steal	force	TOC	
object-level logging				
¬atomic	steal	¬force	TCC	(only at system shutdown)
¬atomic	steal	¬force	ACC	(at regular intervals)
¬atomic	steal	force	TOC	
miscellaneous				
¬atomic	steal	¬force	fuzzy	
atomic	steal	¬force	ACC	
atomic	steal	force	TOC	
¬atomic	¬steal	¬force	fuzzy	

From simulations using different transaction workloads, Reuter concludes that page-logging is generally more costly than object-level logging, that an increase in shared pages makes all *force* algorithms drastically worse than others and that schemes that use indirect mapping, such as after-image shadow paging, impose extra overheads unless the page-table costs can be amortised.

Agrawal and DeWitt [AD85] produced an analytical model which they use to investigate the relative costs of object logging, shadow paging, and differential files and their interaction with locking, timestamp ordering and optimistic concurrency control schemes. Rather than produce costings based on transaction throughput their model uses a performance metric that describes the burden imposed on a transaction by a recovery mechanism and a particular concurrency control scheme. The model recognises that real systems have finite resources and incorporates CPU costs and the impact that the concurrency control schemes may have on the probability that a transaction will run to completion. Burden ratios for the different integrated concurrency control and recovery mechanisms are calculated and compared using sample evaluations from varying transactions workloads and database characteristics. The conclusions from these test runs suggest that there is no overall best integrated mechanism but that a load which comprises of a mix of transaction sizes favours logging with a locking approach. Shadow paging performs rather poorly in their tests. However their model takes no account of synchronous costs, such as the writing back of data pages in shadow paging, or checkpointing in logging. A weakness of the model with respect to more modern systems is that shadow page tables reads were assumed to be from disk, whereas with modern memory sizes the entire shadow page table may reasonably be assumed to be resident in main memory.

These previous studies represent work that is probably closest to that presented in this paper. However re-investigation of this study is considered to be worthwhile in the light of more recent knowledge and more modern machine architectures. In particular, both models have only a single disk I/O cost, making no allowance for the different costs of sequential, asynchronous or synchronous I/O, whereas most modern schemes are designed to take advantage of the differences between these costs.

In contrast to the analytical models described above, the Predator project [KGC85] takes an empirical approach to comparing recovery methods. Prototype databases supporting different recovery mechanisms are constructed on stock hardware together with a transaction simulator used for experimentation. A suite of transaction experiments which vary locality of update, abort frequency and I/O access methods is carried out over databases supporting concurrent shadow paging and page based logging. The performance metrics are based on transaction throughput and mean response time. The experiments are constructed from short transactions on a small system and conclude that shadow paging works best when there is locality of reference and where the page table cache is large, otherwise logging is the better mechanism. The main observation from this work suggests that there is no one best mechanism and that the choice of recovery method is application dependent. However one interesting observation made is that the transaction abort rate has a more radical effect on the performance of logging recovery schemes than on shadow paging.

6 Conclusions

Comparisons between different recovery mechanisms is often a difficult and inconclusive task. A number of not necessarily independent criteria have to be considered when making comparisons. These are:

- The tradeoffs in the time taken to recover after failure, the time and resources used to collect recovery information and the time and resources used in constructing a recoverable system.

- The store architecture and its anticipated use. The issues here include the frequency of updates, locality of reference, object identity and addressing. Scalability of the recovery mechanism with respect to store size may also be of concern.

- The expected frequency of hard and soft crashes. In conflict concurrency systems the frequency of aborted actions is also a factor. This may also depend on the concurrency control implementation used, for example optimistic concurrency control may result in more transaction aborts than say two-phase locking.

- The frequency, cost and style of checkpoints.

- The hardware and operating system support.

The major motivation for this work is that ongoing changes in machine architectures, hardware technology, and database usage patterns are perceived to change the cost comparisons among different recovery mechanisms. To this end an attempt has been made to produce a cost model which is independent from both machine and application workload parameters; these parameters may be injected into the model based either on measurements from real systems or as the result of further analyses and estimates.

Although the work is at an early stage, the line of investigation is believed to have been justified for two reasons: firstly, a cost model which is independent from machine and workload parameters has been implemented and has been shown to be usable; secondly, early investigations of the model fit with both intuition and some rudimentary experiments with real systems. The results do show already that some of the orthodoxy of crash recovery engineering may be challenged in the context of modern application loads and hardware systems.

Much work remains to be done in terms of the refinement and verification of the model. To avoid complexity, the model is somewhat simplistic; in particular, the costs of checkpointing and transaction failure recovery are not yet included. The results shown by the current model therefore favour object logging somewhat artificially over the other mechanisms.

In terms of verification, it is intended to parameterise the model with a number of full sets of configuration and workload parameters which are obtained by measurement and analysis from real database programming applications. These will be used to calibrate and inspire confidence in the model, with the eventual intention of being able to predict the running time of an arbitrary mixture of application, recovery mechanism and platform.

7 Acknowledgements

This work was undertaken during the visit of Eliot Moss to the University of St Andrews as an EPSRC Senior Visiting Fellow on grant GR/K 34924.

8 Appendix I

Transaction workload basic variables

Application workload derived variables may use a combination of the following application workload basic variables:

- **Oacc**: the transaction size. This defines the number of objects which the transaction accesses;
- **Odirt**: the percentage of accessed objects which are updated;
- **Osize**: the average size of an objects;
- **Psize**: the size of a page;
- **ObjLoc / DObjLoc**: the locality of accessed and updated objects within pages. The percentage assigned to this variable determines the degree of locality of the objects within pages. A value of 100% indicates that all objects accessed are densely packed within the fewest pages possible and a small percentage indicates that the objects are scattered over a large number of pages.
- **PTemp**: the temporal locality of pages. This variable describes the degree to which pages remain in main memory long enough to allow more than one transaction to access the page. This may affect the number of reads required by a transaction to fault data pages into memory;
- **Ploc**: the locality of pages updated. This may affect the recovery I/O costs;
- **iread / iwrite**: the percentage of a transaction's updated pages requiring installation reads and writes. If pages are updated by more than one transaction, an installation read and write may incorporate changes by a number of transactions. Therefore these variables can be used to determine the number of installation reads and writes performed by a single transaction in relation to the number of pages updated by the transaction.
- **LRover**: the size of the log record overhead used to store the transaction identifier etc.

- **MapEntry**: the size of a page map entry used in after-image shadow paging.
- **LRratio**: the log record size to object size ratio. Note that this variable is not an application workload variable but is used in the derivation of PLog.

Appendix II

The following derivation functions are used in evaluating the derived variables in Appendix III. Descriptions of the workload variables used in the following functions can be found in Appendix I.

```
procedure numOD( -> int )       ! number of objects dirtied
        Oacc * Odirt

procedure Pmax( -> int )        ! maximum number of pages accessed
        if Osize <= Psize then 2 * Oacc
        else Oacc * roundUp( ( Osize - 1) / Psize )

procedure Pmin( -> int )        ! minimum number of pages access
        roundUp( Oacc * ( Osize / Psize ) )

procedure Pacc( -> int )        ! number of pages accessed
        if ObjLoc = 0 then Pmax else Pmax - (Pmax - Pmin) * ObjLoc

procedure PDmin( -> int )       ! minimum number of accessed pages dirtied
        roundUp( numOD * ( Osize / Psize ) )

procedure PDmax( -> int )       ! maximum number of accessed pages dirtied
        if Osize < Psize then min( 2 * numOD, Pacc)
        else numOD * ( roundUp( ( Osize - 1 ) / Psize ) + 1 )
```

Appendix III

Table 7 illustrates how the application workload functions are composed.

Derived Variable	Description
PMiss	PTemp * Pacc
PDirty	PDmin + ((PDmax - PDmin) * DObjLoc)
PIRead	iread * *PDirty*
PIWrite	iwrite * *PDirty*
PLog	(Osize * numOD * LRratio) / Psize
PpHouse	*PDirty*/(Psize/2)
PolHouse	(LRover / Psize) * numOD
PTMiss	0
PTDirty	*PDirty* / (Ploc * (Psize / MapEntry))
PcommR	LRover/Psize

Table 7: Transaction Workload Derived Variables

9 References

[AD85] Agrawal, R. & DeWitt, D. "Integrating Concurrency Control and Recovery Mechanisms: Design and Performance Evaluation". ACM Transactions on Database Systems, Vol. 10, No. 4, December 1985, pp 529-564.

[BR91] Brown, A.L. & Rosenberg, J. “Persistent Object Stores: An Implementation Technique”. In Dearle, Shaw, Zdonik (eds.), Implementing Persistent Object Bases, Principles and Practice, Morgan Kaufmann, 1991 pp 199-212.

[Bro89] Brown, A.L. “Persistent Object Stores”. Ph.D. Thesis, University of St Andrews (1989).

[Cha78] Challis, M.P. “Data Consistency and Integrity in a Multi-User Environment”. Databases: Improving Usability and Responsiveness, Academic Press, 1978.

[EB84] Elhardt, K. & Bayer, R. "A Database Cache for High Performance and Fast Restart in Database Systems". ACM Transactions on Database Systems, Vol. 9, No. 4, December 1984, Pages 503-525.[Gra78]
Gray, J.N. “Notes on Database Operating Systems”. LNCS 60, Springer-Verlag (1978) pp 393-481.

[Hag87] Hagmann, R.B. “Reimplementing the Cedar file system using logging and group commit”. In Proc. 11th Symposium on Operating Systems Principles, 1987 pp 155-162.

[HR83] Haerder, T. & Reuter, A. “Principles of Transaction-Oriented Database Systems”. ACM Computing Surveys, 15,4 (1983) pp 287-318.

[KGC85] Kent, J., Garcia-Molina, H. & Chung, J. “An experimental evaluation of crash recovery mechanisms”. In Proc 4th ACM Symposium on Principles of Database Systems (1985) pp 113-122.

[KSD+91] Koch, B., Schunke, T., Dearle, A., Vaughan, F., Marlin, C., Fazakerley, R. & Barter C. “Cache Coherency and Storage Management in a Persistent Object System”. in Dearle, Shaw, Zdonik (eds.), Implementing Persistent Object Bases, Principles and Practice, Morgan Kaufmann, 1991 pp 103-113.

[Lor77] Lorie, A.L. “Physical Integrity in a Large Segmented Database”. ACM Transactions on Database Systems, 2,1 (1977) pp 91-104.

[MBC+89] Morrison, R., Brown, A.L., Connor, R.C.H. & Dearle, A. “The Napier88 Reference Manual”. University of St Andrews Technical Report PPRR-77-89 (1989).

[MCM+94] Munro, D.S., Connor, R.C.H., Morrison, R., Scheuerl, S. & Stemple, D.W. “Concurrent Shadow Paging in the Flask Architecture”. To appear in Proc. 6th International Workshop on Persistent Object Systems, Tarascon, France (1994).

[MS88] Moss, J.E.B. & Sinofsky, S. “Managing persistent data with Mneme: Designing a reliable shared object interface”. In Dittrich, K.R. (ed.)

Advances in Object-Oriented Database Systems: Second International Workshop on Object-Oriented Database Systems, LNCS 334, Springer-Verlag, 1988 pp 298-316.

[Mun93] Munro, D.S. “On the Integration of Concurrency, Distribution and Persistence”. Ph.D. Thesis, University of St Andrews (1993).

[OS94] O'Toole, J. & Shrira, L. "Opportunistic Log: Efficient Installation Reads in a Reliable Object Server". Technical Report MIT/LCS-TM-506, March 1994. To appear in 1st International Symposium on Operating Systems Design and Implementation, Monterey, CA (1994).

[RHB+90] Rosenberg J., Henskens F., Brown A.L., Morrison R. & Munro D.S. "Stability in a Persistent Store Based on a Large Virtual Memory.". International Workshop on Computer Architectures to Support Security and Persistence of Information, Universität Bremen, West Germany, (May 1990). In Security and Persistence. (Eds. J.Rosenberg & L.Keedy). Springer-Verlag pp 229-245.

[RO91] Rosenblum, M. & Ousterhout, J.K. “The design and implementation of a log-structured file system”. In Proc 13th Symposium on Operating Systems Principles, 1991 pp 1-15.

[Sev82] Severance, D. “A Practical Guide to the Design of Differential Files for Recovery of On-line Databases”. ACM Transactions on Database Systems, 7,4 (1982) pp 540-565.