# Analysis of Recovery in a Database System Using a Write-Ahead Log Protocol

**Anant Jhingran**
*IBM TJ Watson Research Center*
*Yorktown Heights, NY 10598*

**Pratap Khedkar**
*CS Division, Department of EECS*
*University of California at Berkeley*
*Berkeley, CA 94720*

**Abstract**

In this paper we examine the recovery time in a database system using a Write-Ahead Log protocol, such as ARIES [9], under the assumption that the buffer replacement policy is strict LRU. In particular, analytical equations for log read time, data I/O, log application, and undo processing time are presented. Our initial model assumes a read/write ratio of one, and a uniform access pattern. This is later generalized to include different read/write ratios, as well as a "hot set" model (i.e. x% of the accesses go to y% of the data). We show that in the uniform access model, recovery is dominated by data I/O costs, but under extreme hot-set conditions, this may no longer be true. Furthermore, since we derive analytical equations, recovery can be analyzed for any set of parameter conditions not discussed here.

## 1 INTRODUCTION

Several recovery algorithms have been proposed in the literature, and the most common and popular ones are those that use the write-ahead log (WAL) protocol with no force and steal policies [6]. In these, a transaction updates the data pages in place, and commits by writing log records to the disk. Furthermore, a page can be written to the disk anytime during or after the transaction, provided the corresponding log record is forced before the write happens.

In [9], Mohan et al. propose an idempotent WAL recovery protocol called ARIES, which we will use in our analysis. However, this analysis can be modified to other WAL protocols, especially those that record the state of the memory in the checkpoints. Recovery in ARIES proceeds by looking at the last checkpoint record, and using it to go to a particular point in the log and scanning the log forward from that point onwards. Each log record may then result in a data I/O (and possibly a write of another page to free up buffer space), and the updates in that log record may then be applied to the data page.

In this paper, we analyze the various components of recovery time. In particular, we concentrate only on software/hardware failures (i.e. ignore media failures), since they are the most common and the most critical from the perspective of quick recovery. Our analysis should not only help the database designer "anticipate" the recovery time, but also help him design appropriate policies to restrict it in cases where it becomes inordinate.

While a lot of work has been done in the area of database buffer modeling [3, 5], most of it has been directed towards analyzing the "buffer hit probabilities" under various page replacement strategies. Analysis of recovery has often been directed towards worse-case scenarios [2] and is based on relatively simple page write policies like forcing dirty pages at checkpoint. However, recovery analysis when based on LRU or its variations[1] is much more difficult, and the true average case can only be obtained after the entire distribution of the recovery time is obtained.

This paper presents a first step towards analyzing recovery based on various page replacement strategies by studying LRU in considerable detail. We make the standard Independent Reference Model assumption, which states that all accesses to data pages are statistically independent [8]. The analysis will proceed by making some simplifying assumptions, and then successively refining them in order to tackle more complex scenarios:

- Each data page that is accessed is also updated. This is true of TPCB [1] type transactions, but might not be true otherwise. We later allow different read/write ratios.
- Each data page is equally likely to be accessed, i.e., initially, we assume that there are no hot-spots in the database. Since frequently, databases have hot-spots with typical x-y distribution of accesses (i.e. x % of accesses go to y % of the database), we later relax the uniform access assumption.

The rest of the paper is organized as follows. In Section 2 we give a brief overview of ARIES taken from [9] and in Section 3 we present our analytical model under the uniform access assumption, and study the behavior under various parameter settings. In Section 4, we discuss how read/write ratios different from 1 can be handled, and in effect we show that the recovery time decreases as the read/write ratio is increased. Section 5 refines the the model to include a hot-cold dichotomy, and we show that some of the components of the recovery time might increase many-fold. The paper ends with conclusions and suggestions for future work, including a discussion on how technology changes might affect the conclusions in this paper.

[1] The variations on LRU try and restrict the recovery time by forcing additional I/O's and are described in the conclusions.


1992 ACM SIGMOD - 6/92/CA, USA
© 1992 ACM 0-89791-522-4/92/0005/0175...$1.50

## 2 ARIES

In this section, we describe the relevant aspects of the ARIES algorithm as applicable to recovery. The description used in this paper does not adhere completely to that in [9], but its salient features are present.

In ARIES, every log record has an implicit log sequence number (or **LSN**), which is the address of the first byte of the log record. Each data page also stores the LSN of the log record that describes the last change made to that page, in the field **pageLSN**.

ARIES maintains a dirty page table (**DPT**) for all pages that have not yet been pushed to the disk. This DPT contains two columns for each page that is dirty and in-memory: the **page-id**, and **RecLSN** (or recovery log sequence number, which is the address of the next log record to be written when the page was first modified). In other words, whenever a non-dirty page is updated in memory, a row is added to the DPT with the corresponding page-id and the next LSN to be assigned. Thus, the RecLSN indicates a position in the log from where we have to start examining the log to discover the log record that describes the changes made to this page. Whenever a page is written to the disk, the corresponding entry in DPT is deleted. Furthermore, at a checkpoint (which happens periodically), the current DPT is also recorded. Finally, whenever a page is updated, the pageLSN entry on that page is updated to reflect the LSN of the log record performing that update.

When the database system is recovering from a crash, it goes through the log from the last check-point record (which contains the DPT at the time of the checkpoint) to the end, and reconstructs the DPT. In addition, it determines winner and loser transactions. This scan is called *analysis phase*.

The next phase is called the *redo phase*, and the pseudocode in Figure 1 describes the actions performed. **MinRecLSN** is the minimum RecLSN of the entries in DPT at the last checkpoint, and indicates the earliest change in the database log that was potentially not written to the disk. The page describing the given log record needs to be examined only if

1. The redo processing is before the last checkpoint, and the page appears in the DPT at the checkpoint, or
2. The redo processing has crossed the last checkpoint.

This criterion is a slight modification to the actual ARIES algorithm, and is based on the assumption that the pages are brought in as if the analysis phase has not taken place. The exact rationale is given in [7] and does not affect the results in any significant way.

If the page needs to be examined, and is not in the memory, then it is fetched (resulting in data I/O costs), and its pageLSN entry is examined. If the pageLSN entry is less than the LSN of the corresponding log record (indicating that the page was written prior to this log record, and hence does not contain its changes), the log record is reapplied to the data page (resulting in log application costs).

Finally, in the third pass, called *undo phase*, all updates to the loser transactions (i.e. those transactions that were in flight at the time of the crash) are undone.

In summary, the recovery components are:

1. Analysis Phase
2. Redo phase, consisting of a) Log Scan Time, b) Data I/O time, and c) Log Application Time.
3. Undo phase.

## 3 MODEL FOR UNIFORM ACCESS

Under the assumption that no writes were in progress while the DPT was being written in the checkpoint record, this DPT records the memory state **exactly**. As mentioned previously, we also assume that accesses occur with uniform probability, the read/write ratio is one, and the buffer replacement policy is strict LRU.

In order to model LRU replacement policy, it is convenient to define the **rank** of a page in the buffer. If we assume that the pages in the buffer are sorted in the increasing order of the elapsed time since their last access, then each page has a unique rank between 1 and $B$ where $B$ is the size of the buffer pool (in pages). We can extend this concept and state that if the rank of a page is $B+1$, then it is on the disk. Every time a page is accessed, it either enters the buffer pool with a rank 1, or its rank within the buffer is **reset** to 1. As it stays in the buffer without being accessed, its rank eventually increases to $B+1$, and it is thrown out of the buffer. This method of specifying the state of the buffer has long been used [8], and is useful because the resulting markov chain on the state space has a unique stationary distribution [4], which is uniform when the access pattern is uniform. Finally, we define the **age** of a page in the buffer as the number of accesses (to it, or to other pages) since it entered the buffer (including the one that brought it into the buffer).

Table 1 describes the relevant notation for our analytic model. The first eight parameters are applicable to our simplest model (namely uniform access), as well as to the other two models (hot-set and different read/write ratios). The other sets of parameters are applicable only to their respective models. Based on these parameters, we will derive the distributions of the random variables described in Table 2. In this section, we derive the various components of the recovery time when the accesses are uniform. We first analyze the redo phase components, followed by the analysis and the undo phases.

### 3.1 Components of Redo Phase

In the next three subsections, the three components of redo – data I/O, log application, and log scan – are analyzed in detail. We show closed form solutions for the first two, and algorithmic derivation for the last.

#### 3.1.1 Data I/O time

If we assume that the entire buffer is lost during the crash, and that $B$ pages are available during the recovery, then it can be seen that each page in the DPT at the last checkpoint is needed to be brought into memory at least once. However, in addition to the pages in the DPT at the last checkpoint, all other pages that appear in the log records after the checkpoint also need to be brought in during the redo phase. The number of such pages is just the number of different pages that appear in the log records after the last checkpoint *and which do not also appear in the DPT at the last checkpoint*. In order to derive the distribution for this number, let us define $r_j$ to be the number of pages required to be brought in during redo till the $j^{th}$ record after the last checkpoint. Then $r_j$ has the following distribution (for

```
for each log record from MinRecLSN
{
        Scan the log record;  /* i.e. pay log scan cost */
        if (page needs to be examined)
        {
                if (page not in memory)
                        fetch page from disk;              /* i.e. page data I/O cost */
                if (pageLSN < LSN of this log record)
                        apply the log record to the page;  /* i.e. log application cost */
        }
}
```

Figure 1: ARIES redo algorithm

| Model | Parameter | Description |
|---|---|---|
| Uniform | $B$ | Buffer Size (in pages) |
| | $N$ | Total DB size (in pages) |
| | $CP$ | Number of log records between checkpoints |
| | $lpt$ | Number of log records per transaction |
| | $mpl$ | Number of transactions running concurrently |
| | $M_{logscan}$ | I/O + CPU time to read a log record |
| | $M_{IO}$ | I/O time per data page |
| | $M_{logapply}$ | Time to apply a log record to a data page |
| Hot-Set | $p_h$ | Prob. of access to a hot page |
| | $H$ | Number of hot pages |
| Different R/W ratios | $f_w$ | Prob. that an access is a write |

Table 1: Parameters of the Analytical Model

| Model | Random Variable | Description |
|---|---|---|
| Uniform and Hot-Set | $r_j$ | Number of data reads during redo upto the $j^{th}$ record after the last checkpoint |
| | $io_j$ | Number of data I/O (read + writes) during redo upto the $j^{th}$ record after the last checkpoint |
| | $io$ | Number of data I/O's during redo upto to the point of crash |
| | $L_j$ | Number of accesses to a page in rank $j$ since it was last written to disk |
| | $L$ | Total number of accesses to pages in the buffer |
| | $A_j$ | Age of the page with rank $j$ in buffer |
| | $A$ | Age of the oldest dirty page in the buffer |
| Different R/W ratios | $W_j$ | Number of writes to a page in rank $j$ since it was last written to this disk |
| | $W$ | Total number of writes to pages in the buffer |

Table 2: Distributions derived in this paper

$j, k > 0$):

$$\Pr(r_j = k) = \Pr(r_{j-1} = k)\frac{k}{N} + \Pr(r_{j-1} = k-1)(1 - \frac{k-1}{N}) \quad (1)$$

with $\Pr(r_0 = B) = 1$ as the starting condition.

The above equation says that if after $j-1$ records, the number of different pages required is $k-1$, then with a probability of $1-(k-1)/N$ it will increase by one after $j$ records (i.e. a page not currently required is accessed), and otherwise it will remain the same.

If $c$ records appear in the log after the last checkpoint, then the total number of different pages that are required during redo is given by $r_c$. Furthermore, if this much of buffer space is available during recovery, then no page need be brought in more than once, and hence the amount of data I/O during redo is given by $io_c = r_c$.

When the buffer is only $B$ (i.e. not sufficient to accommodate $r_c$ number of pages), the amount of data I/O during recovery is likely to be more than $r_c$, since dirty pages might have to be written out. In order to simplify the analytic modeling, we assume that $CP < B$, which allows us to show that every page in rank $B$ is dirty after the redo processing has crossed the checkpoint record. It is clear that during recovery, when the redo processing is at the checkpoint record, every page in the buffer is dirty. Furthermore, under the assumption that $CP < B$, not more than $B$ new pages enter the buffer, guaranteeing that at every instance during recovery past the checkpoint record, the page in rank $B$ is dirty. Consequently, during redo processing, the number of writes is exactly equal to the number of data pages that have to be read in beyond the checkpoint record.

Based on the above discussion, it can be seen that the following relationships exist:

$$\Pr(r_j = k) = \Pr(r_{j-1} = k)\frac{B}{N} + \Pr(r_{j-1} = k-1)(1 - \frac{B}{N}) \quad (2)$$

since beyond the checkpoint, the buffer always contains exactly $B$ pages. Now it is easy to see that $r_j - B$ is binomially distributed with parameters $j$ and $1 - B/N$, and hence we have

$$\Pr(r_j = k) = \binom{j}{k-B}(1 - \frac{B}{N})^{k-B}(\frac{B}{N})^{j-k+B} \quad (3)$$

The total number of I/O's is given by $io_j = r_j + r_j - B$, since $r_j - B$ is the number of pages read in after the checkpoint, and hence the number of pages written out.

Figure 2a plots, for $B = 1000$ and $N = 10000$, the distribution of $io_c$ for the four combinations of a) $c = 100$, $c = 200$ and b) limited, unlimited buffer during recovery. We see that $io_c$ has a sharp spike slightly less than $c + B$ for the unlimited buffer cases, and at $2c + B$ for the limited buffer case. The expected number of I/O's on a random crash is given by:

$$E(io) = \frac{1}{CP}\sum_{c=0}^{CP-1} E(io_c) \quad (4)$$

The time required for the data I/O's is given by:

$$T_{IO} = M_{IO} * E(io) \quad (5)$$

### 3.1.2 Log Application Time

Recall the condition for applying a log after a page has been brought into memory - the PageLSN must be less than the LSN of the log record being considered. Viewed differently, this implies that a log record needs to be applied only if the page was last written **before** the log record. A bit of introspection will reveal that for a particular page $p$ in the buffer, the number of log records that need to be applied is exactly equal to the number of accesses to it while it was in the buffer before the crash (including the one that brought it into the buffer).

The number of log applications is then given by $L = \sum_{j=1}^{B} L_j$, where $L_j$ is the number of log records (equivalently, the number of accesses since last being written to the disk) for the page in the $j^{th}$ rank. It can be shown that the $L_j$ are identically distributed. This follows from the recurrence (for $i > 1$):

$$\begin{aligned}\Pr(L_i = k) &= \Pr(L_i = k)\Pr(\text{access to } 1..i-1) \\ &\quad + \Pr(L_{i-1} = k)\Pr(\text{access not to } 1..i-1)\end{aligned} \quad (6)$$

The distribution of $L_1$ is easy to derive. In rank 1, $k$ resets will be seen only if a page in the buffer with $k-1$ resets gets accessed. This holds for $k > 1$, and $k = 1$ can occur only if a fresh page is brought in (with prob. $pe = 1 - B/N$). We have

$$\Pr(L_1 = k) = \sum_{j=1}^{B}\frac{1}{N}\Pr(L_j = k-1), \quad k > 1 \quad (7)$$

which leads to the geometric distribution

$$\Pr(L_j = k) = \Pr(L_1 = k) = pe(1-pe)^{k-1} \quad (8)$$

with mean $1/pe$. Now the distribution of $L$ (which is a sum of independent and identically distributed geometrics) is Pascal with the following distribution:

$$\Pr(L = k) = \binom{k-1}{B-1} pe^B (1-pe)^{k-B}, \quad k \geq B \quad (9)$$

The independence follows from the assumption that accesses to different pages are independent. The mean of the above distribution is $B/(1 - B/N)$, and the standard deviation is $B/\sqrt{N}(1 - B/N)$. Figure 2b plots the distribution of $L$ for $B = 1000$, $N = 10000$.

The time for log application is given by:

$$E(T_{logapply}) = M_{logapply} * E(L_{logapply}) \quad (10)$$

### 3.1.3 Scanning the Log

During redo processing, the number of log records that need to be examined upto the last checkpoint is exactly the maximum age of the pages in the buffer pool (i.e. in the DPT in the last checkpoint). This follows from the definition of MinRecLSN in the ARIES recovery algorithm. Let us define $A$ to be $Max(A_j, 1 \leq j \leq B)$ (recall that $A_j$ is the age of the page in rank $j$).

In order to derive the distribution of $A$, we need the distributions of $A_j, 1 \leq j \leq B$. It is easy to derive a recursive

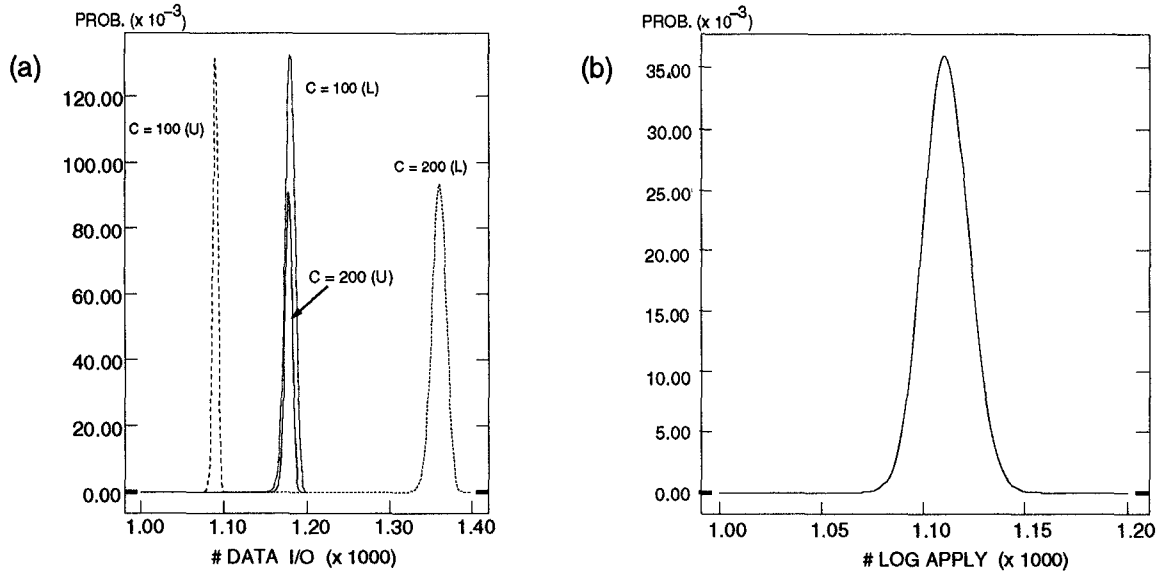


Figure 2: (a) Distribution of Data I/O's. L and U indicate limited and unlimited buffers respectively. (b) Distribution of Total # of Log Applications.

formulation for $\Pr(A_j = k)$, which is the probability that the page with a rank of $j$ has an age of $k$.

$$\Pr(A_j = k) = \begin{cases} 1 - \frac{B}{N} & \text{if } j = 1, k = 1 \\ 0 & \text{if } j > k \\ \frac{1}{N}\sum_{i=1}^{B} \Pr(A_i = k-1) & \text{if } j = 1, k > 1 \\ \Pr(A_{j-1} = k-1)(1 - \frac{j-1}{N}) & \\ +\Pr(A_j = k-1)\frac{j-1}{N} & \text{otherwise} \end{cases} \quad (11)$$

We now describe each of these components in turn.

- $j = 1, k = 1$: This happens when a page not in the buffer is brought in, and happens with a probability $1 - B/N$.
- $j > k$: A page cannot travel farther into the rank than its age, and hence this probability is zero.
- $j = 1, k > 1$: This happens when a page within a buffer is referenced, and it results in setting the rank of the page to one, and incrementing the age of the page by one. The expression above reflects the sum of the probabilities of the (mutually exclusive) events that lead to this.
- Otherwise: On a general reference, the age of the page increases. Its rank also increases if the referenced page has a greater rank (or is on the disk), but remains the same if the referenced page is ranked below it. The above is reflected in the last term in the above equation.

Based on the distributions of $A_j$, we can write $A$ as;

$$\Pr(A \leq k) = \prod_{j=1}^{B} \Pr(A_j \leq k) \quad (12)$$

provided we make the simplifying assumption of independence of the $A_j, 1 \leq j \leq B$. It is obvious that this assumption is not quite true, because no two ranks can have the same age. However, our simulations have shown that the product of cumulative distribution functions (cdf's) overestimates the true distribution slightly. In this paper, we will use this product for studying $A$ with the caveat that the resulting cdf is above the true cdf. Furthermore, the true effect of this component of the recovery is multiplied by $M_{logscan}$, which being sequential scan, is fairly small compared to other components. Consequently, the simplifications in modeling this part of the recovery do not lead to severe inconsistencies in the total recovery time, as we show later.

In order to see how the distribution of $A$ varies with $B$, Figure 3 plots the distributions of $A$ for $B$ = 500, 1000, 1500 and 2000, for $N$ = 10000. The expected value of $A$ increases faster than $B$, and in fact, so does its standard deviation. The reason for high expectation of $B$ even when $B/N = 0.05$ is not difficult to see. The value of $A$ is determined by the probability that **one** page is that old. So even though the probability that individual pages are that old is fairly low, the probability of at least one page being old is fairly high, and increases as $B$ increases, even if $B/N$ is kept constant. In [7], we show that $E(A)$ grows as $B \log B$. Thus we can conclude that the oldest page in the buffer is likely to be very old, even based on a simple LRU policy. When we include hot pages into consideration, the age of the oldest page will only increase, thus exacerbating the recovery time even further.

Based on the above, the expected length of the log examined is then given by:

$$E(L_{logscan}) = \sum_{k=1}^{\infty} k \Pr(A = k) + \frac{CP}{2} \quad (13)$$

because the length of the log examined before the last check-

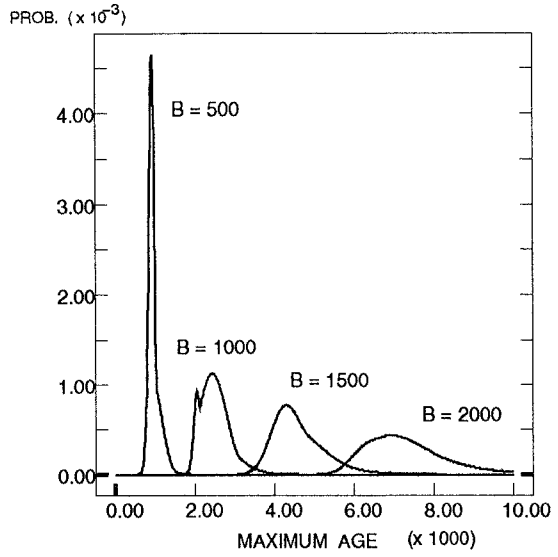


Figure 3: Distribution of MaxAge for various B

point is determined entirely by the DPT at the last checkpoint, and on the average, a crash will occur midway between two checkpoints, resulting in a log scan of $\frac{CP}{2}$ after the last checkpoint.

The log scanning time is given by:

$$E(T_{logscan}) = M_{logscan} * E(L_{logscan}) \quad (14)$$

We now turn to the other two components of recovery – the analysis and the undo phases.

### 3.2 Analysis and Undo Processing

It is easy to see that the number of log records scanned during the analysis phase is uniform in $[1, CP]$. For typical values of $CP$, we see that the time taken for analysis phase is small compared to the other component associated with reading the log records – the log scan time of the redo phase. Consequently, the recovery component due to analysis is ignored from now on.

The number of transactions that need to be undone because of a crash is given by $mpl$, the multi-programming level. Assuming that a transaction writes $lpt$ log records[2], uniformly spaced, the total number of log records that need to be applied during undo processing can be approximated as a uniform distribution between 0 and $lpt * mpl$, with an expected value of $0.5 * lpt * mpl$, and the time for this is given by:

$$\begin{aligned} E(T_{undologapply}) &= M_{logapply} * E(L_{undologapply}) \quad (15) \\ &= M_{logapply} * 0.5 * lpt * mpl \end{aligned}$$

Both the data I/O and log scan costs during undo can be ignored under the assumptions of short duration transactions, because the corresponding pages and log records would have been read in during the redo phase, and are likely to be around during the entire undo phase.

[2]These parameters are assumed to be independent of others for simplicity of analysis.

### 3.3 Total Recovery Time

In Figure 4a, we have plotted the recovery time components for the following parameters: $B = 1000$, $N = 10000$, $CP = 200$, $mpl = 10$, $lpt = 3$, $M_{logscan} = 0.4msec$, [3] $M_{IO} = 20msec$ and $M_{logapply} = 0.8msec$,[4] and under the assumption that only $B$ pages of buffer are available during recovery. The data I/O distribution corresponds to the assumption that the crash can occur anywhere between checkpoints. We see that the recovery time is dominated by the data I/O (about an order of magnitude more than the next major component), followed by the log application and the log scan time, with the time for undo being a small fraction of the total time spent during recovery. Figure 4b plots the expected values of the recovery time components as a function of $B$ for the above values of the other parameters. We see that the recovery time is around 26 seconds for a buffer size of 1k. Furthermore, we see that while both log scan and log application times are comparable, the former starts to dominate over the latter as the $B$ increases. We know that the expected log application time increases as $B/(1 - B/N)$, indicating that the expected MaxAge increases at least as fast as this.

## 4 DIFFERENT READ/WRITE RATIOS

Until now we have been considering a read/write ratio of one, i.e. every access involves an update. That is certainly true of the data pages of Account, Teller, and Branch relations touched in the TPCB benchmark, but not true when one considers the associated index pages. In more complex benchmarks read/write ratios are typically in the range of 2-10.

If every page that is accessed is not updated, then each of the three components of the recovery time (excluding undo processing, that is) gets affected. The age of a page is to be really counted from the first update, and not the first access, and this is likely to reduce the maximum age in the buffer, thus affecting the log scan time. The data I/O has to be paid for only those pages that were dirty at the time of the crash, and this is likely to be smaller than the total buffer size. Finally, the number of log records that need to be applied on a page is exactly equal to the number of accesses to it that were updates, in contrast to our previous analysis where every access resulted in a log application.

In this section we analyze the various components of recovery under varying read/write ratios. The access distribution is uniform as before. Let us also define $f_w$ to be the fraction of accesses which are also updates.

### 4.1 Data I/O Time

Here we describe the number of reads (i.e. number of I/O's) under the unlimited buffer assumption because that is the easiest to model. Furthermore, if $B/(B + CP) \geq f_w$, then it is expected that the total number of pages read in during recovery does not exceed $B$, and hence modeling unlimited buffers is just a convenience.

[3]Under the assumption that each log record is 200 bytes, and sequential reads can proceed at 1MByte/sec, it takes around 0.2 msec to each log record, and we add another 0.2 msec for the CPU processing on the log

[4]This is based on the assumption that applying a log takes around 4000 instructions, which on a 5 MIPS machine takes 0.8 msec.

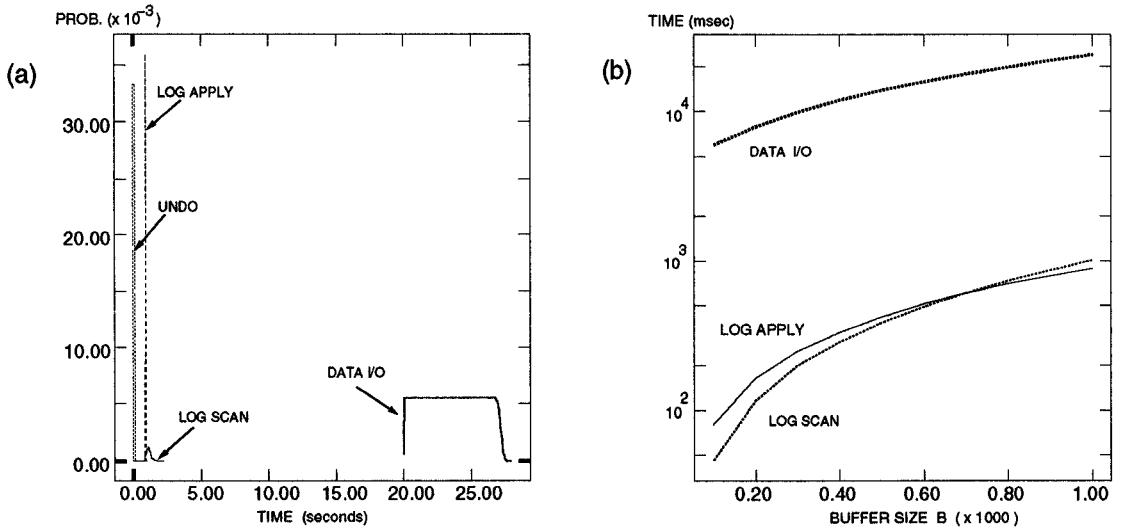


Figure 4: (a) Components of Recovery Time. (b) Expected Values of Recovery Components

Let us define $pc_i$ to be the probability that the page in rank $i$ is "clean". Then we have:

$$pc_i = \sum_{k=1}^{\infty} [\Pr(L_i = k)(1 - f_w)^k] \tag{16}$$

which follows from the fact that if the page is clean, then all accesses to it in the buffer must have been reads, and $L_i$ is the distribution of the number of accesses to it. Substituting for the distribution of $L_i$ derived earlier, we have,

$$\forall i, \quad pc_i = pc = \frac{(1 - \frac{B}{N})(1 - f_w)}{(1 - (1 - f_w)\frac{B}{N})} \tag{17}$$

since it is independent of $i$.

We then have the following distribution for $r_0$ (the number of data reads till the last checkpoint):

$$\Pr(r_0 = k) = \binom{B}{k}(1 - pc)^k pc^{B-k} \tag{18}$$

which is binomial in $B$ and $1 - pc$. This follows from the observation that $r_0$ is the number of dirty pages at the last checkpoint. Furthermore, we have:

$$\begin{aligned} \Pr(r_j = k) \quad = \quad & \Pr(r_{j-1} = k)[\frac{k}{N} + (1 - \frac{k}{N})(1 - f_w)] \\ & + \Pr(r_{j-1} = k - 1)(1 - \frac{k-1}{N})f_w \end{aligned} \tag{19}$$

which says that the number of pages read in does not increase if a page already read in is accessed or a page from outside is accessed for read. Otherwise, the number of pages read in increases by one.

Figure 5a plots the distribution of the number of I/O's (in this case, the number of reads), for the parameters used in Figure 3 and for $f_w = 0.1$, 0.5 and 1, assuming that the crash occurs randomly between checkpoints.

### 4.2 Log Application Time

Let $pr, pw$ be the respective probabilities of read or a write of a page in the buffer (where $pr + pw = B/N$ and $pw/(pr + pw) = f_w$). Then the distribution of $W_i$, the number of writes seen by the page in rank $i$ while in the buffer is:

$$\Pr(W_i = w) = \sum_{k \geq 1} \Pr(W_i = w | L_i = k) \Pr(L_i = k) \tag{20}$$

where $L_i$ is the number of accesses to the page. After some simplification, we get

$$\Pr(W_i = w) = \left(\frac{pw}{1 - pr}\right)^w \frac{pe}{(1 - pr)(1 - pe)}, \qquad w > 0 \tag{21}$$

For the special case $W_i = 0$,

$$\Pr(W_i = 0) = \frac{pe.pr}{(1 - pr)(1 - pe)} \tag{22}$$

This distribution is similar in form to a geometric distribution but not exactly so. Substituting $f_w = pw/(1 - pe)$, the mean of the above distribution is

$$E(W_i) = \frac{f_w}{1 - B/N} = f_w E(L_i) \tag{23}$$

and the variance can also be computed easily.

The total number of writes over all $B$ pages in the buffer will no longer be Pascal since $W_i$ is not pure geometric but using the Central Limit Theorem, the distribution of $W = \sum_{i=1}^{B} W_i$ is normal with mean $B.E(W_i)$ and standard deviation $SD(W_i)\sqrt{B}$.

Figure 5b plots the approximation for $W$ for the parameters used in Figure 5a.

### 4.3 Scanning the Log

We now allow the variables $A_j$ to be zero-valued, which corresponds to being clean in the buffer. So $A_j$ does not

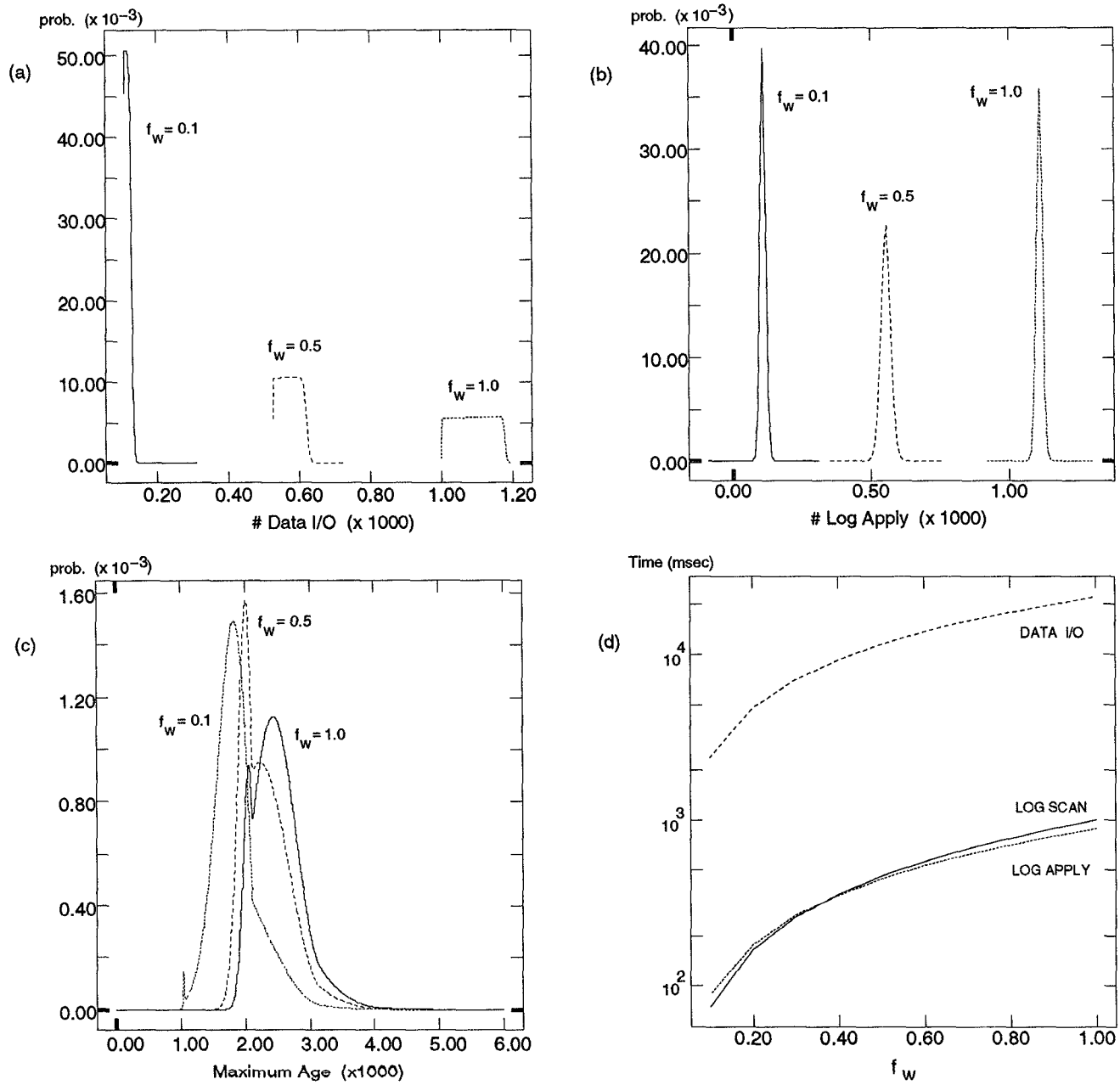


Figure 5: Different R/W ratios: (a) Data I/O Distribution. (b) Distribution of #Log Applications. (c) Distribution of Max Age. (d) Expected Values of Recovery Components vs. $f_w$

begin counting age until the page is updated at least once. As a result, $\Pr(A_j = k)$ is given by (for $k > 1$):

$$\begin{array}{ll} [(1 - \frac{B}{N}) + \frac{1}{N}\sum_{i=1}^{B} \Pr(A_i = 0)]f_w & \text{if } j = k = 1 \\ 0 & \text{if } j > k \\ \frac{1}{N}\sum_{i=1}^{B} \Pr(A_i = k-1) & \text{if } j = 1, k > 1 \\ \Pr(A_{j-1} = k-1)(1 - \frac{j-1}{N}) & \\ + \Pr(A_j = k-1)\frac{j-1}{N} & \text{o.w.} \end{array} \quad (24)$$

The first term follows from the observation that a page attains an age of 1 in rank 1 when it is either brought from outside for update, or a clean page is accessed from inside for update. The other terms remain the same because once the page has been dirtied (i.e. achieved an age greater than zero), its age will advance normally. Also, $\Pr(A_j = 0)$ is simply $pc$ derived earlier.

Figure 5c plots $\Pr(A = k)$ under the above formulations for parameters used in Figure 5a. We see that as $f_w$ decreases, the maximum age decreases. However, the distribution is to the right if 1000, even for $f_w = 0.1$.

The time for log scan decreases by another factor of $f_w$, since when the maximum age is $k$, the number of log records that are scanned during recovery is just $kf_w$.

### 4.4 Total Recovery Time

In Figure 5d, we have plotted the three major components of recovery time as a function of $f_w$. We see that the total recovery time when $f_w$ is decreased from 1 to 0.5 is reduced from 26 seconds to around 15 seconds.

## 5 NON-UNIFORM ACCESS PATTERNS

In this section we expand our model to include the x-y % rule, where x% of the accesses go to y% of the data. The analysis and undo times are unaffected by the hot-set model, and hence not discussed here. We define $p_h$ to be the probability of access to the hot-set (whose size is $H$), i.e. $x = 100p_h$ and $y = 100(H/N)$. Following [8], [5], [7], we can derive $bhp$, the buffer hit probability based on LRU replacement and a hot-set access pattern. Due to lack of space, we remark briefly on the three components and show their average behavior in Figure 6. See [7] for further details and figures.

We derive the data I/O time under the assumption that the amount of buffer space available during recovery is fixed at $B$. The equations are analogous to the uniform case with $B/N$ replaced by $bhp$. It is observed that the number of I/O's actually decreases as the dichotomy increases, since the number of data pages read in *after the checkpoint* actually decreases (as most of the pages are already in the buffer).

With the dichotomy, $L_i$ are no longer independent or identically distributed. However, the restricted variables ($L_i$|Rank $i$ has a hot page) are independent. Their distributions are approximately equal in the non-extremal cases (when $H/B$ is not too small). This can be used to show the expected result that the number of log records that need to be applied increases as the dichotomy increases.

The LogScan time rises sharply with a hot-cold dichotomy because the oldest page is hot and stays in the buffer for very long times. For example, when 50% of the accesses go to 1% of the data instead of 50%, the expected maximum age goes up from $2.6B$ to $5.8B$.

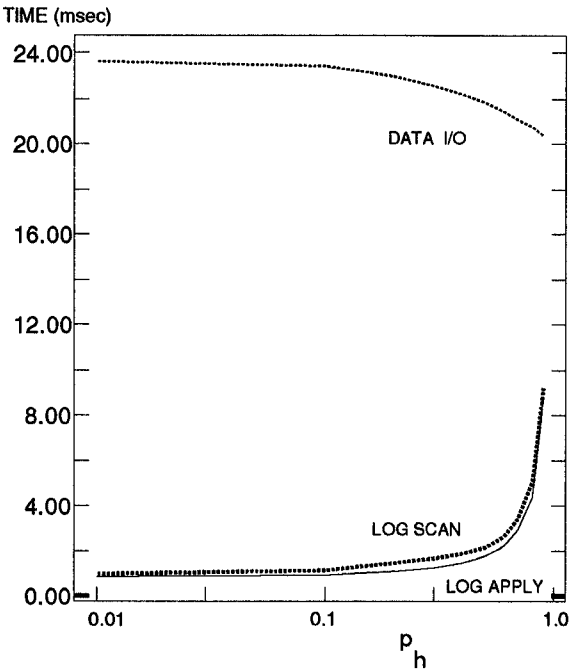


Figure 6: The Hot-Set Model: Expected Values of Recovery Components vs. $p_h$

For the constants used in Figure 4a, and the parameter values ($H = 100, N = 10000, c = 200$) Figure 6 plots the various components of the recovery time as a function of $p_h$. We see that not only does the total recovery time increase with an increase in the dichotomy, but also that the major contributions to this increase come from the log scan and log apply time, with the I/O time remaining fairly constant (actually decreasing somewhat). At 90% of the accesses going to 1% of the data, the total recovery time increases from around 26 seconds to 48 seconds.

## 6 CONCLUSIONS AND FUTURE WORK

In this paper we have studied the recovery processing using ARIES. Our initial model was based on certain simplifying assumptions, namely uniform access patterns and a read/write ratio of one. We derived the distributions for the time taken for all the components of recovery, but concentrated on the redo components:

- Log Scan, which was proportional to the age of the oldest page in the buffer.
- Data I/O, which depended on the amount of buffer space available during recovery.
- Log Application, which was a function of the number of times a page is accessed while in the buffer.

By substituting realistic numbers for our parameters, we showed that the recovery time is dominated by data I/O, and can be completed in about half a minute when the buffer size is 1000 pages.

Our first embellishment to the model was to include different read/write ratios, and we showed how the probability

distribution of the redo components could then be derived. We showed that a read/write ratio greater than one reduced all the recovery time components, and when it was equal to 2, the recovery took around 16 seconds.

We then expanded our model to include a "hot-set", and showed that this tended to lower the data I/O time, increase the log scan and log processing time, and not affect the undo time. It was shown that when 90% of the accesses went to 1% of the data, the recovery time rose to around three-quarters of a minute, with the major factors being the log scan and the log application times, which could be ignored when the hot-cold dichotomy does not exist.

As a part of further work, we are expanding the model to include the following:

- Some policies (in addition to LRU) that might be used to reduce various components of recovery time:
  1. Clip the age beyond certain value, in effect, clipping the maximum age, and thus limiting the log scan time. This will be achieved by forcing pages that have aged sufficiently to the disk. A worst case analysis for such a policy is presented in [2].
  2. Clip the log application beyond a certain value by forcing a page that has been updated a sufficient number of times to the disk.
  3. Clip the data I/O by a) taking checkpoints more frequently, or b) not allowing more than certain number of dirty pages in the buffer.

  As is obvious, these strategies are considerably more difficult to analyze, and furthermore, each strategy, while designed to affect one distribution, has an indirect affect on the other two components of redo.

- In schemes such as Asynchronous Replica Management [2], dirty pages are asynchronously spooled to a secondary node. Consequently, if a node goes down, the secondary node can run the log, recover the database, and start processing transactions. The one advantage of this recovery is that the secondary's buffer is not lost, and consequently, the data I/O requirements at the secondary might be reduced. The same effect can be observed on one site if some sort of battery-backed up buffer is used. The distributions under various policies for transferring dirty pages between a volatile buffer and a non-volatile buffer (either on a different node, or battery-backed up) need to be examined.

We conclude this paper with a brief discussion on the impact of the emerging main memory technologies, that are likely to push up reasonable values of $B$ in the range of a few tens of thousands, even on workstations. Assuming that $B/N$ remains constant (i.e. databases grow only as fast as the main memory), we see that the data I/O, might, at some point in time, be dominated by the log scan costs, since the former grows almost linearly with $B$, whereas the latter grows much faster. Figure 7 plots the three redo components (under a uniform access model) against $B$ for $B/N = .1$. We see that while the log scan components does increase, it is still well below the data I/O component. In fact, as $B$ increases to 50k, the difference between the two actually increases.[5] Based on this, we conclude that for the next few years, it is most important to limit the number of dirty pages in the memory at any given time if bounded recovery times are required.

[5] The effect is because $M_{IO}$ dominates over $M_{logscan}$ for these values of $B$, even though $E(IO)$ increases linearly and $E(A)$ increases superlinearly.

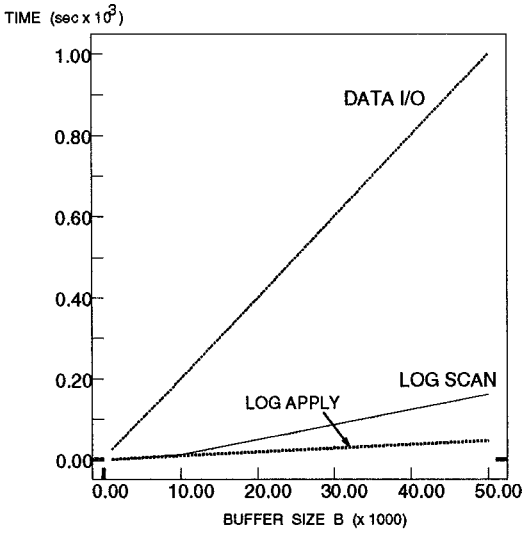


Figure 7: Impact of Main Memory Technologies

## Acknowledgments

Pratap Khedkar is supported by grants from MICRO and NASA (NCC-2-275).

## References

[1] Anonymous, "A Measure of Transaction Processing Power," Datamation, 1985.

[2] Bhide, A. et al., "An Effcient Scheme for Providing High Availability," Proc. ACM SIGMOD, June 1992.

[3] Chou, H. and DeWitt, D., "An Evaluation of Buffer Replacement Strategies for Relational Database Systems," Proc. 11th International Conf. on VLDB, 1985.

[4] Coffman, E. G. and Denning, P. J., "Operating Systems Theory", Prentice-Hall, Englewood Cliffs, N.J., 1973

[5] Dan, Asit and Towsley, Don, "An Approximate Analysis of the LRU and FIFO Buffer Replacement Schemes", ACM Sigmetrics, Boulder, Colorado, 1990

[6] Haerder, T. and Reuter, A., "Principles of Transaction-Oriented Recovery," ACM-TODS, Sept. 1983.

[7] Jhingran, A. and Khedkar, P., "Analysis of Recovery in a Database System Using a Write-Ahead Log Protocol," IBM Tech Report RC 17475, Dec. 1991.

[8] King, W. F., "Analysis of Paging Algorithms", Proceedings of the IFIP Congress, pp 485-490, Ljublanjana, Yugoslavia, Aug 1971.

[9] Mohan, C. et al., "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks using Write-Ahead Logging," To Appear in ACM TODS, March 1992.