# I/O Reference Behavior of Production Database Workloads and the TPC Benchmarks - An Analysis at the Logical Level

*Windsor W. Hsu*[⋆†]
*Alan Jay Smith*[⋆]
*Honesty C. Young*[†]

[⋆]*Computer Science Division*
*University of California*
*Berkeley, CA 94720*
{*windsorh,smith*}*@cs.berkeley.edu*

[†]*IBM Research Division*
*IBM Almaden Research Center*
*San Jose, CA 95120*
{*windsor,young*}*@almaden.ibm.com*

**Report No. UCB/CSD-99-1071**

November 1999

Computer Science Division (EECS)
University of California
Berkeley, California 94720

# I/O Reference Behavior of Production Database Workloads and the TPC Benchmarks - An Analysis at the Logical Level

Windsor W. Hsu[⋆†]
Alan Jay Smith[⋆]
Honesty C. Young[†]

[⋆]Computer Science Division
University of California
Berkeley, CA 94720
{windsorh,smith}@cs.berkeley.edu

[†]IBM Research Division
IBM Almaden Research Center
San Jose, CA 95120
{windsor,young}@almaden.ibm.com

**Abstract**

As improvements in processor performance continue to far outpace improvements in storage performance, I/O is increasingly the bottleneck in computer systems, especially in large database systems that manage huge amounts of data. The key to achieving good I/O performance is to thoroughly understand its characteristics. In this paper, we present a comprehensive analysis of the logical I/O reference behavior of the peak production database workloads from ten of the world's largest corporations by focusing on how these workloads respond to different techniques for caching, prefetching and write buffering. Our findings include several broadly applicable rules of thumb that describe how effective the various I/O optimization techniques are for the production workloads. For instance, our results indicate that the buffer pool miss ratio tends to be related to the ratio of buffer pool size to data size by an inverse square root rule. A similar fourth root rule relates the write miss ratio and the ratio of buffer pool size to data size.

In addition, we characterize the reference characteristics of workloads similar to the Transaction Processing Performance Council (TPC) benchmarks C (TPC-C) and D (TPC-D), which are de facto standard performance measures for on-line transaction processing (OLTP) systems and decision support systems (DSS) respectively. Since benchmarks such as TPC-C and TPC-D can only be used effectively if their strengths and limitations are understood, a major focus of our analysis is on identifying aspects of the benchmarks that stress the system differently than the production workloads. We discover that for the most part, the reference behavior of TPC-C and TPC-D fall within the range of behavior exhibited by the production workloads. However, there are some noteworthy exceptions that affect well-known I/O optimization techniques such as caching (LRU is further from the optimal for TPC-C while there is little sharing of pages between transactions for TPC-D), prefetching (TPC-C exhibits no significant sequentiality) and write buffering (write buffering is less effective for the TPC benchmarks). While the two TPC benchmarks generally complement one another in reflecting the characteristics of the production workloads, there remain aspects of the real workloads that are not represented by either of the benchmarks.

This work has been supported by the State of California under the MICRO program, and by IBM, Cisco Corporation, Fujitsu Microelectronics, Intel Corporation, Microsoft Corporation, Quantum Corporation, Sun Microsystems and Toshiba Corporation.

This report is also available as Research Report RJ 10166, IBM Almaden Research Center, San Jose, CA.

## 1 Introduction

I/O is increasingly the bottleneck in computer systems as processor performance continues to improve at a much faster rate than storage performance [24]. This is especially the case for large database systems that reference a lot of data. There are several well-known approaches to improving I/O performance. Among them are caching, prefetching and write buffering. The effectiveness of these general approaches depends very much on the characteristics of the reference stream. In addition, if the reference characteristics are well understood, these techniques can be customized to further improve performance. Nevertheless, there has not been much work on analyzing the reference characteristics of production database workloads. This reflects the fact that production systems are by definition critical to the proper functioning of an organization so that it is very difficult to get access to them for the purpose of conducting a scientific study, especially if the study requires any software changes or if data is to be collected and removed from the system.

In this paper, we examine the logical I/O reference behavior of the peak production database workloads from ten of the world's largest corporations. Our primary focus is on analyzing the factors that affect how these workloads respond to different techniques for caching, prefetching and write buffering. We evaluate many previously published algorithms and techniques and also develop several new ones based on our insights. Our production workloads are based on IBM's industrial-strength DB2 relational database management system (DBMS) and to the best of our knowledge, represent by far the most complete and diverse set of production workloads ever reported on in the literature. Since it is rare to have access to such a large collection of production workloads, an emphasis of this paper is on establishing broadly applicable rules of thumb with regards to the effectiveness of caching, prefetching and write buffering.

Though the Transaction Processing Performance Council (TPC) benchmarks C (TPC-C) [70] and D (TPC-D) [71] have become the de facto standard benchmarks for on-line transaction processing (OLTP) systems and decision support systems (DSS) respectively, and are heavily used for both systems design and marketing, there has not been any major focus on their I/O characteristics. Therefore, in this paper, we also evaluate the logical reference behavior of workloads similar to the TPC-C and TPC-D benchmarks[1]. While benchmarks such as TPC-C and TPC-D are important for progress in the field in that they define the playing field by establishing objectives that are easily measurable and repeatable, they may impede real progress in the field if they are not realistic and end up focusing energy and attention on issues that do not often arise in production environments. To effectively use a benchmark, we should carefully evaluate its characteristics against those of real production workloads to identify both its strengths and limitations.

[1]Because our TPC benchmark setups have not been audited per the benchmark specifications, our workloads are technically not TPC benchmark workloads and should only be referred to as TPC-like. In the rest of this paper, when the terms TPC-C and TPC-D are used to refer to our benchmark workloads, they should be taken to mean TPC-C-like and TPC-D-like respectively.

Therefore, one of the main objectives of this study is to determine whether the I/O reference behavior of the TPC benchmarks reflect that of the production workloads, especially with respect to the various techniques such as read caching, prefetching and write buffering that can be applied at the logical I/O level to improve overall DBMS performance. In a companion paper [26], we analyze and compare the system-level characteristics of the production and benchmark workloads. System-level characteristics are those that can be readily understood and perhaps observed by the user or system administrator and are therefore good features to use for comparing and understanding workloads. Although the current paper is self-contained, readers are encouraged to also read the companion paper.

The rest of this paper is organized as follows. Section 2 contains a brief overview of previous work in characterizing I/O reference behavior. Section 3 discusses our methodology and describes the traces that we use. In Section 4, we characterize the reference behavior of the workloads. Concluding remarks appear in Section 5 and acknowledgements, in Section 6. Because of the huge amount of data that is involved in this study, we can only present a characteristic cross-section in the main text. The rest of the results are presented in the Appendix. In addition, more detailed graphs and data are available from our web site [28].

## 2 Related Work

There have been numerous published studies of the reference behavior of hierarchical and network databases. See for instance [9, 18, 32, 33, 34, 48, 54, 57, 64, 74, 73, 76, 82]. However, these studies are rather limited in scope, often relying on data collected at only one or two installations. In several cases, the database was real but was driven by small contrived programs. The empirical reference behavior of relational databases has received even less attention [17, 23, 32, 82]. In such databases, users state their processing requirements using high level language interfaces, leaving the database system to select the best strategy or query plan for accessing the data [12, 61]. Since the pattern of data references can be predicted from the query plan [67], research on buffering in relational databases has for the most part focused on using semantic information derived from the query plan optimizer to direct buffer management [11, 14, 19, 45, 59, 60, 80]. Instead of relying on query plan information, a profiling approach that uses prior executions of a query to characterize its access patterns has also been proposed [10].

In general, the approaches that rely on the query plan work only for specific patterns such as sequential and cyclic sequential. All other references are simply considered random. Another shortcoming of these methods is that the query plan is based on estimates such as row cardinality, predicate selectivity and clustering factor and these may not be accurate. Furthermore, for complex queries, the accurate prediction of reference patterns from the query plan is non-trivial. To make matters worse, in multi-user situations, the query plans can overlap in complicated ways and this overlap is not accounted for by the query plan directed algorithms. In reality, these algorithms are best used together with techniques based on run time access characteristics as in [22, 68].

For the most part, studies of the I/O behavior of the TPC-C [70] and TPC-D [71] benchmarks have been limited to analysis of query plans [27], static analysis of accesses to tables [37] and empirical measurement of buffer hit rate [72]. File reference patterns in academic and research environments have been more extensively studied. See for example [5, 20, 49, 69, 78, 81]. An analysis of file usage patterns in commercial environments is presented in [55]. There has also been a large body of recent work on characterizing the I/O behavior of scientific applications in parallel and supercomputing environments. Among them are [6, 15, 41, 47, 50, 51, 53]. For a more detailed discussion of related work, the reader is referred to Section A1 in the Appendix.

## 3 Methodology

The methodology used in this paper is trace-driven simulation [66, 75]. In trace-driven simulation, relevant information about a system is collected while the system is handling the workload of interest. This is referred to as tracing the system and is usually achieved by using hardware probes or by instrumenting the software. In the second phase, the resulting trace of the system is played back to drive a model of the system under study. In other words, trace-driven simulation is a form of event-driven simulation where the events are taken from a real system operating under conditions similar to the ones being simulated. More comprehensive discussions of this technique and its strengths and weaknesses can be found in [66, 75].

The traces used in this study were collected by instrumenting commercial DBMSs. Instrumenting the DBMS allows the trace information to be collected at a logical level. This reduces dependencies on the system being traced and allows the trace to be used in a wider variety of studies, including those in which the models are somewhat different from the original system. In this study, we examined a total of 14 traces representing both industry standard benchmarks (TPC-C and TPC-D [70, 71]) and the production workloads of ten of the world's largest corporations. The benchmark traces were collected on a multiprocessor Personal Computer (PC) Server running DB2/Universal Database (DB2/UDB) V5 [30] on Windows NT 4.0. The production traces were collected on IBM mainframes running various versions of DB2/MVS, now known as DB2/390 [29]. More information about how the traces were collected can be found in [26].

In order to make our characterization more useful for subsequent mathematical analyses and modeling by others, we fitted our data to various functional forms through non-linear regression which we solved by using the Levenberg-Marquardt method [52]. When appropriate, we also fitted standard probability distributions to our data by using the method of maximum likelihood to obtain parameter estimates and then optimizing these estimates by the Levenberg-Marquardt algorithm [52].

### 3.1 Workload Description

The TPC-C benchmark models the operational end of the business environment where real-time transactions are processed [70]. It is set in the context of a wholesale supplier and is centered around its order processing operations which consist of business transactions that enter new orders, query the status of existing orders, deliver outstanding orders, enter payments from customers and monitor warehouse stock levels. The TPC-C performance metric is the number of orders processed per minute. The benchmark specifies a method for scaling the database which is based on an assumed business expansion path of the supplier. Our particular trace was collected on a benchmark setup with a scale of 800 warehouses.

The TPC-D benchmark models the analysis end of the business environment where trends are analyzed and refined to support sound business decisions [71]. The TPC-D database is a decision support database that tracks, possibly with some delay, the OLTP database through batch updates. The benchmark consists of 17 read-only queries that are far more complex than most OLTP transactions and typically examine large volumes of data using a rich set of operators and selectivity constraints. To exercise the update functionality of the DBMS, the benchmark includes two update functions that modify a small percentage of the database. The TPC-D benchmark defines both a power test to measure the raw query execution power of a system with a single active user and a throughput test that may be omitted. Our trace captures the entire run of a power test. This test starts off with the first update function (UF1). Next, the 17 queries are processed in a sequence specified by the benchmark. Finally, the second update function (UF2) is executed. As with TPC-C, the TPC-D benchmark specifies a method for scaling the database. Our trace was collected on a system with a scale factor of 30.

More details about the benchmarks can be found in [26] and in the benchmark specifications [70, 71]. Note that the

| Trace | Aerospace | Bank | ConsGds | DirMktg1 | DirMktg2 | FinSvcs | Insurance | Retail | TelecomA | TelecomB1 | TelecomB2 | Utility | Prod. Ave. | TPC-C | TPC-D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Source** | Aerospace company | Banking corp. | Consumer goods company | Direct mail marketing firm | Direct mail marketing firm | Financial services firm | Insurance company | Discount store | Telecom. Company A | Telecom. company B | Telecom. company B | Utility company | - | TPC benchmark C | TPC benchmark D |
| **Platform** | MVS on IBM S/370 | MVS on IBM S/370 | MVS on IBM S/370 | MVS on IBM S/370 | MVS on IBM S/370 | MVS on IBM S/370 | MVS on IBM S/370 | MVS on IBM S/370 | MVS on IBM S/370 | MVS on IBM S/370 | MVS on IBM S/370 | MVS on IBM S/370 | MVS on IBM S/370 | WinNT on Intel X86 | WinNT on Intel X86 |
| **DBMS** | DB2/MVS | DB2/MVS | DB2/MVS | DB2/MVS | DB2/MVS | DB2/MVS | DB2/MVS | DB2/MVS | DB2/MVS | DB2/MVS | DB2/MVS | DB2/MVS | DB2/MVS | DB2/UDB | DB2/UDB |
| **Date Collected** | 2/3/1992 | 5/13/1991 | 9/8/1992 | 9/18/1991 | 9/19/1991 | 6/6/1991 | 10/7/1992 | 7/1/1992 | 4/15/1992 | 10/8/1990 | 10/9/1990 | 5/14/1991 | - | 2/10/1998 | 3/8/1998 |
| **Duration (h:m)** | 2:29 | 22:57 | 1:59 | 1:03 | 2:02 | 3:54 | 2:41 | 4:52 | 1:40 | 2:27 | 1:42 | 3:16 | 4:15 | (withheld) | (withheld) |
| **# Objects** | 2203 | 1281 | 626 | 1446 | 1446 | 3124 | 1953 | 434 | 521 | 255 | 255 | 1139 | 1224 | 101 | 192 |
| **Data Size (MB)** | 33558 | 53079 | 3423 | 18191 | 18191 | 10064 | 38095 | 72188 | 197422 | 15114 | 15114 | 39070 | 42792 | 70246 | 77824 |
| **Footprint (MB)** | 1397 | 9600 | 726 | 1137 | 1362 | 2127 | 1732 | 6769 | 2986 | 947 | 976 | 5727 | 2957 | 13267 | 51580 |
| **# References** | 7779007 | 35916414 | 7133845 | 6401880 | 14396125 | 15664004 | 20648874 | 38646360 | 13072916 | 11531195 | 13757374 | 37653369 | 18550114 | 196067649 | 218130354 |
| **# Xacts** | 98931 | 85173 | 66102 | 11892 | 14906 | 20956 | 70242 | 797637 | 84378 | 36508 | 25899 | 118191 | 119235 | 890885 | 230 |
| **Read Ratio (%)** | 93.8 | 90.6 | 86.9 | 95.4 | 95.6 | 90.9 | 84.8 | 86.9 | 85.9 | 93.0 | 98.1 | 89.3 | 90.9 | 87.4 | 97.8 |

Table 1: Summary of Trace Characteristics.

TPC benchmark rules prohibit publicly disclosing TPC performance figures that have not been independently audited. Therefore, we withhold from this paper any data that may be used to derive our TPC metrics. This omission of absolute TPC performance numbers should not compromise our understanding of the logical reference behavior of the benchmarks.

Our other traces were collected in the day-to-day production environments of a diverse group of very large corporations. The industries represented include aerospace, banking, consumer goods, direct mail marketing, financial services, insurance, retail, telecommunications and utilities. In all cases, our traces include the peak production database workload as identified by the system managers. This is typically a combination of transaction processing and long-running queries. The trace referred to as Telecom in [82] and Phone in [62] is the first 30 minutes of the trace we call TelecomB1.

### 3.2 Trace Description

Table 1 summarizes the characteristics of the various traces that are used in this paper. Because of the large number of production workloads, we often also present the arithmetic mean of their results. This is denoted as “Prod. Ave.” In the table, the term *object* refers to a logical collection of data, such as a database table or an index, that is managed as an entity in much the same way as a file. *Data size* represents the total size of all the objects in the system and was obtained from the catalog dumps that were taken when the systems were traced. The *footprint* of a trace is defined as the amount of data referenced at least once in the trace. The traces record information from the perspective of the DBMS. Therefore, the object count includes DBMS system objects like catalogs, views and plans. In addition, the transactions recorded are database transactions, several of which may be needed to perform a single business transaction. The production traces were taken off the primary systems in use at some of the world’s largest corporations in the early nineties. Though these databases were considered large a few years ago, they are comparable in size to the TPC benchmark databases that can be supported on a high-end multiprocessor PC server today.

In the course of this research, there were situations where the state of various simulators had to be established before meaningful statistics could be collected. This is often referred to as warming up the simulator. For instance, the buffer pool in a real system is seldom empty, except during start up. Therefore, if we simulate the buffer pool miss ratio starting with an empty buffer pool, the results will be skewed by the extra misses that are needed to fill the buffer pool. A more meaningful approach is to collect the statistics after the buffer pool has been filled or warmed up. Such statistics are known as *warm* statistics. Unless otherwise stated, we used half of the trace for such warm-up purposes for most of the traces. Because the footprint of Bank increases abruptly around the middle the trace, we prolonged the warm-up period for Bank to slightly beyond the halfway mark. For the TPC-D trace, we used only a quarter of the trace to warm up our simulators because this already achieved a large enough footprint. The various warm-start points are presented in Table 2.

In this paper, we generally present only the results for buffer pools that are filled at the predetermined warm-start point. However, in computing the average result of the production workloads (denoted “Prod. Ave.”), we consider all the results, including those for buffer pools that are not full by the warm-start point. This ensures that we are always taking the arithmetic mean of 12 results (one for each production trace) so that the average is continuous. Note that some of the traces contain references to large pages, *i.e.,* those with

| Trace | Aerospace | Bank | ConsGds | DirMktg1 | DirMktg2 | FinSvcs | Insurance | Retail | TelecomA | TelecomB1 | TelecomB2 | Utility | TPC-C | TPC-D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # References | 3889504 | 20000000 | 3566923 | 3200940 | 7198063 | 7832002 | 10324437 | 19323180 | 6536458 | 5765598 | 6878687 | 18826685 | 98033825 | 54532589 |
| % References | 50.0 | 55.7 | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 25.0 |
| % Trace Time | 42.6 | 65.8 | 49.1 | 45.0 | 40.1 | 51.4 | 50.4 | 50.5 | 64.6 | 45.3 | 50.3 | 50.3 | 51.9 | 42.0 |

Table 2: Warm-Start Point.

sizes that are multiples of the 4KB base page size. For consistency, we converted these to refer to 4KB pages. More detailed description of the traces can be found in [26].

## 4 I/O Reference Characteristics

### 4.1 Overview

The key to good I/O performance lies in discovering useful reference patterns in the workload and effectively exploiting such patterns. Imagine the references trudging through I/O space and leaving footprints wherever they have visited. The highly-frequented regions will have deep footprints. Figures 1.i and 1.ii plot the footprints left by the references over time. In these plots, the y-axis represents the address modulo 32MB while the x-axis represents time in terms of the number of references. Each dot in the plots represents a space-time region of 256KB by 4096 references. In such plots, hot spots will appear as horizontal lines while sequential reference patterns will be manifested as lines running upwards at an angle. In order to show the relative depth of the footprints, the dots are plotted on a 256-level gray scale with a 5% saturation level, *i.e.,* any dot with 5% or more of the maximum intensity is treated as having the maximum intensity. We apologize for the small scale of these plots; a scale that permitted one to distinguish individual pages would require wallpaper for display.

Observe that TPC-C's reference behavior is markedly different from that of the production workloads. TPC-C's references are random and noise-like with relatively few obvious hot spots. Also, unlike the production workloads, which clearly exhibit sequentiality of reference, TPC-C does not exhibit any significant sequentiality. There does exist a faint line that slopes up very gradually in TPC-C's plot. This results from appending a row to the ORDER-LINE table for every item ordered [26, 70]. Because a page contains many rows, the page references increase very slowly. In this case, temporal reuse rather than sequentiality is the dominant characteristic. On the other hand, the reference patterns exhibited by the TPC-D queries are clearly sequential and are more structured and regular than those of the production workloads, perhaps because the TPC-D queries are run serially. Most of the production workloads have regions with distinct reference patterns. This reflects the fact that real production environments are typically much less controlled than benchmark environments and therefore have a greater variation in their workloads. Results from [26] indicate that TPC-C is made up of small transactions while TPC-D is dominated by very large transactions. The production workloads contain a combination of small and large transactions but their reference behavior appears to be more complex and varied than a simple superposition of the reference streams of TPC-C and TPC-D.

A closer examination of Figures 1.i and 1.ii shows that the production workloads exhibit what appears to be cyclic sequential or looping reference patterns. In Section 4.3, we will consider how this affects attempts to reduce buffer pool pollution by purging pages that have been sequentially accessed. Notice also that the plot for TPC-D's queries contains clearly sequential patterns with two or three dark horizontal lines. This sort of reference pattern is the result of sequential or near sequential index probes where an index is repeatedly used to look up keys that are ordered or nearly ordered. The dark horizontal lines reflect references to the root and intermediate nodes of the index while the sequential patterns reflect references to the leaf nodes. There are index lookaside techniques, such as [3], that avoid complete traversal of the index in these situations but we did not enable them in our TPC-D run so as to reduce the effect of any DBMS-specific optimizations. This sort of access pattern, though less prevalent, is also observed in the production workloads and suggests that the index lookaside techniques will be useful for the production workloads too. With this qualitative overview of the reference patterns as a backdrop, we will quantitatively characterize the reference behavior of the various workloads in the next few sections.

### 4.2 Locality and Skew

In this section, we evaluate the amount of temporal reuse that is exhibited by the various workloads. Because variations of the Least-Recently-Used (LRU) replacement algorithm are widely used in commercial systems [22, 68], we present the LRU miss ratio in Figure 2. Some more recently proposed replacement algorithms have been reported to offer a 5-10% improvement in hit rate over pure LRU for relatively small buffer sizes [32, 48, 56].

That the workloads exhibit locality of reference is evident from Figure 2. All the workloads, with the exception of Bank, have LRU miss ratios of less than 15% with a 100MB buffer pool. Bank's LRU miss ratio remains well above that of the other workloads until a buffer pool of 7GB at which

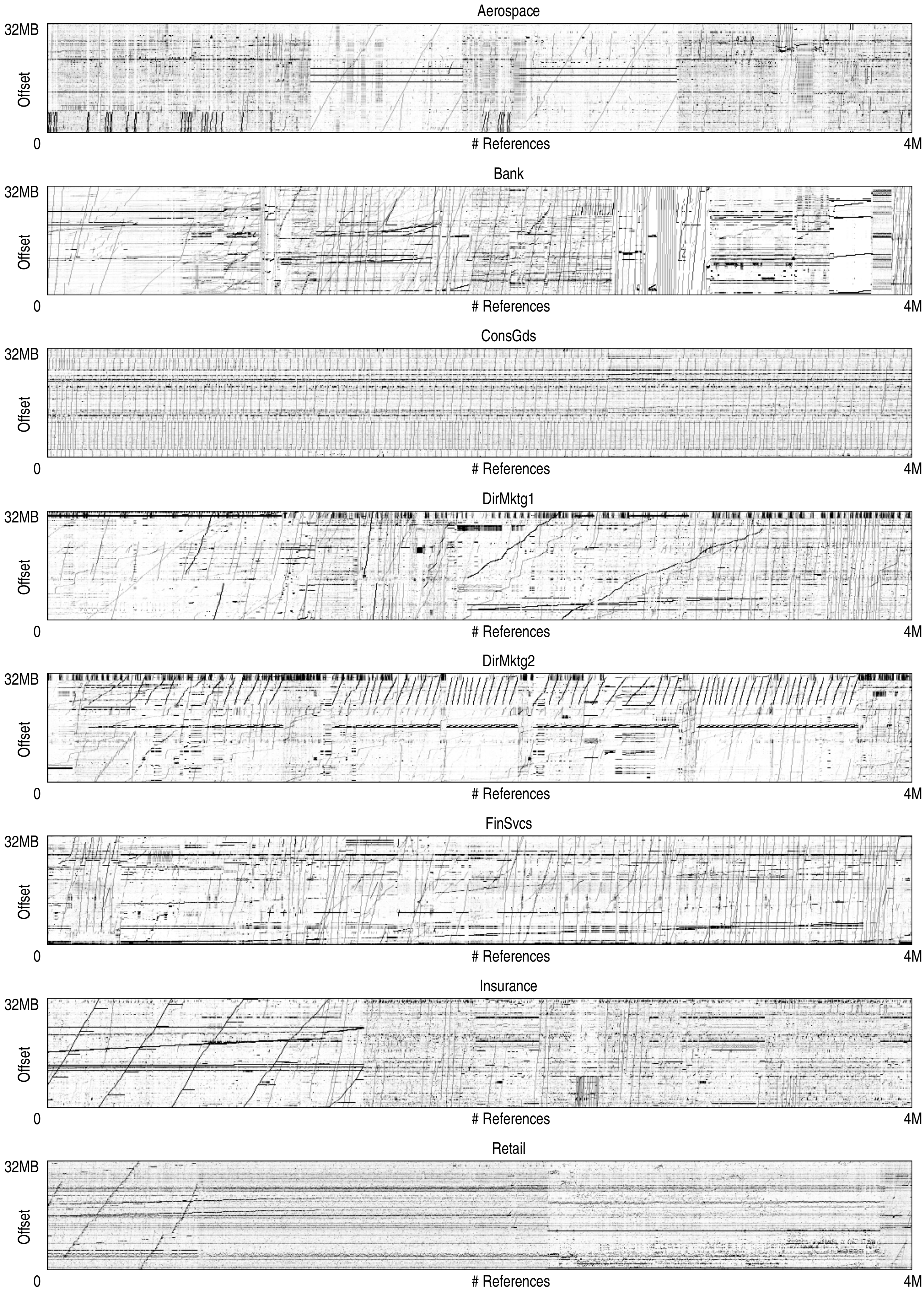



Figure 1.i: Reference Map. The addresses that are referenced are plotted modulo 32MB. Each dot in the figure represents a space-time region of 256KB by 4096 references. Any dot with 5% or more of the maximum intensity is treated as having the maximum intensity.

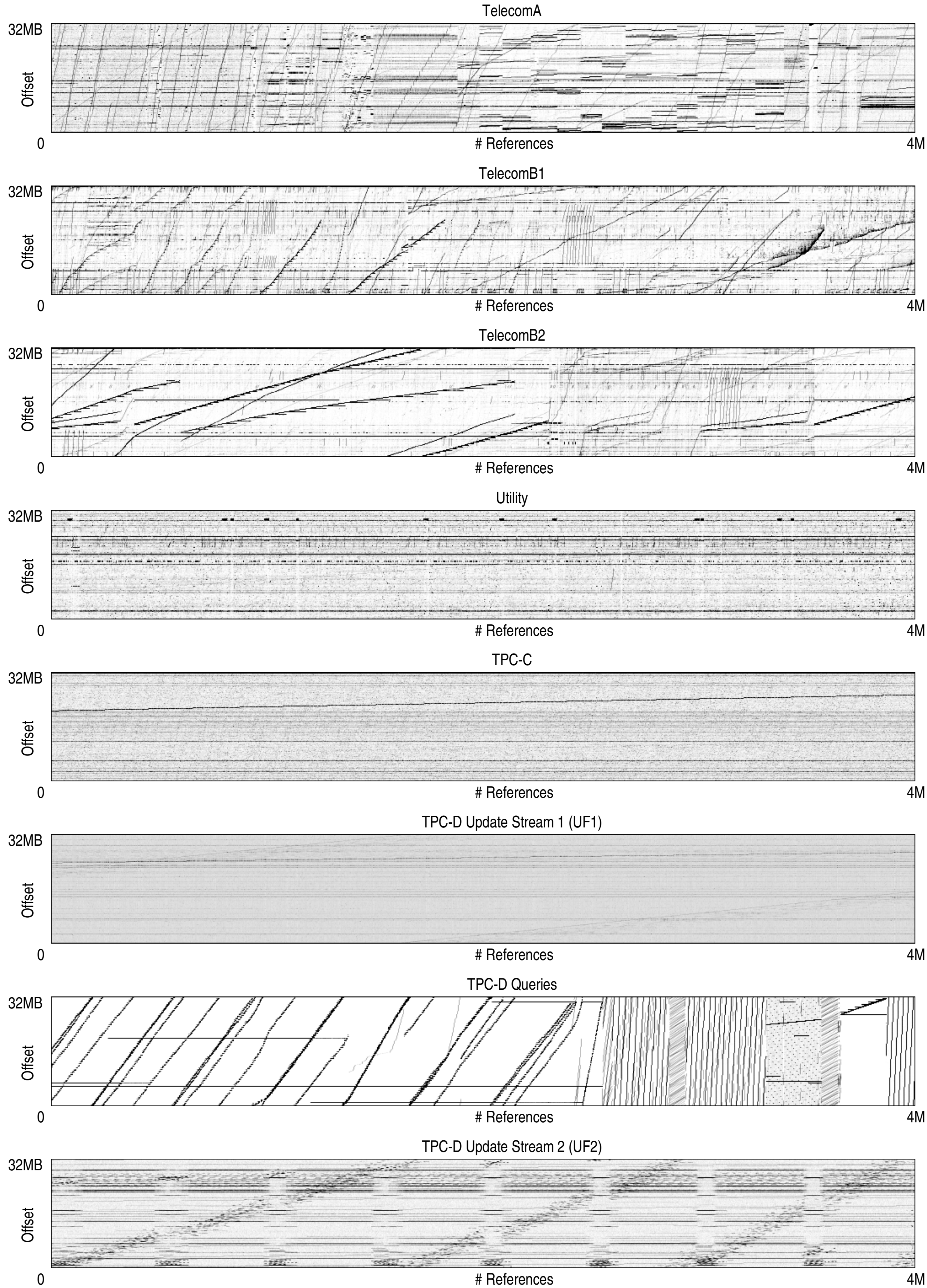



Figure 1.ii: Reference Map. The addresses that are referenced are plotted modulo 32MB. Each dot in the figure represents a space-time region of 256KB by 4096 references. Any dot with 5% or more of the maximum intensity is treated as having the maximum intensity.

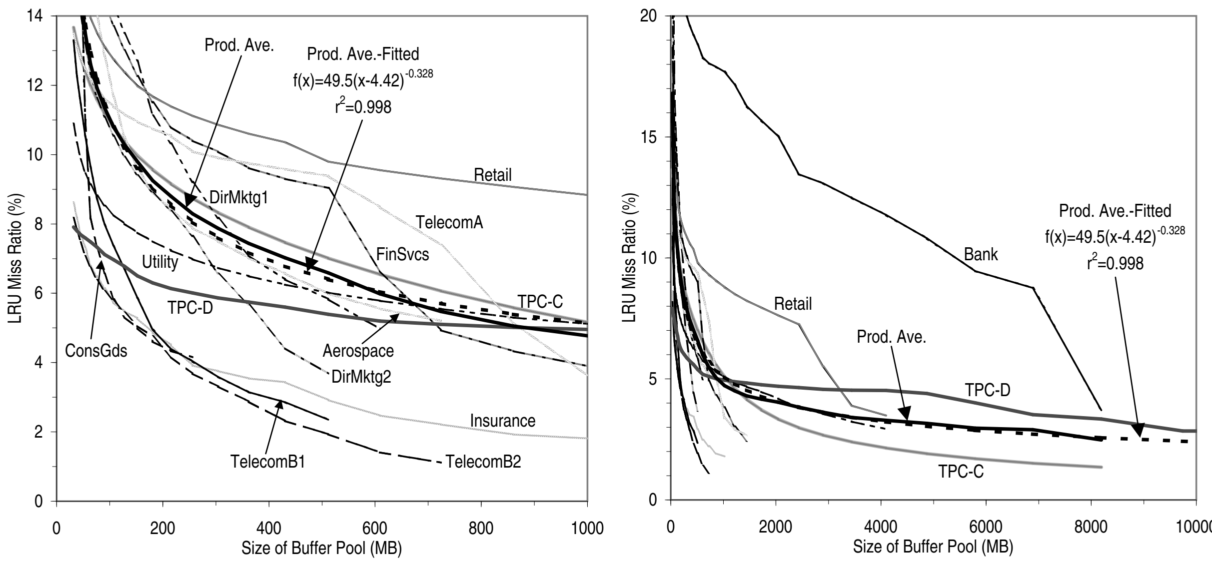



Figure 2: LRU Miss Ratio.

point it plummets. This sort of behavior is primarily due to loops that just fit within a 7GB buffer pool. Compared to the production workloads on average, TPC-C's miss ratio improves more quickly with increases in buffer pool size while the opposite is true of TPC-D. For the most part, the average miss ratio of the production workloads falls between that of TPC-C and TPC-D.

We can often obtain the miss ratio for a given workload at a given buffer pool size. For instance, DB2/UDB maintains performance counters that can be used to calculate the buffer pool miss ratio [30]. The interesting question is whether such data can be used to project the miss ratio at larger buffer pool sizes. More generally, an analytical model of the relationship between miss ratio and buffer pool size will be extremely useful for guiding both the design of future systems and the upgrade decisions for current systems. To this end, we fitted the data for our production workloads to various functional forms. As shown in Figure 2, we found that the relationship between miss ratio and buffer pool size is accurately described by a function of the form $f(x) = a \cdot (x-b)^c$, where $a$, $b$ and $c$ are constants. In this case, $c$ is approximately $\frac{1}{3}$. Interestingly, [38] presents a similar cube root rule ($f(x) \approx d \cdot x^{-\frac{1}{3}}$ where $d$ is a constant) which was obtained by examining I/O workloads at 11 commercial installations using a simple statistical model of cache reference locality. The difference between the two results is that our function is shifted to the right by $b$ units which probably reflects that the minimum buffer space required is $b$ units. In this case, $b$ is about 4MB.

The main criticism of using LRU in database buffer management is that it will not perform well for a cyclic or looping reference pattern until the buffer pool is large enough to hold the entire loop, at which point the miss ratio will suddenly improve dramatically. This sort of behavior can be seen as a concave kink in the miss ratio plots in Figure 2. Among the production workloads, Bank, Retail, TelecomA, FinSvcs and Insurance clearly show this behavior. To a much lesser extent, TPC-D also has this behavior but not TPC-C. Possible ways to improve the handling of loops include caching a loop only if it will fit within the buffer pool and using the Most Recently Used (MRU) replacement policy to handle the loops. The first technique hinges on the ability to determine whether a loop will fit within the buffer pool while the second requires knowledge of the marginal benefit of allocating buffer space between the loop and other competing needs. On a per transaction basis, both can be estimated to a certain extent using the query plan optimizer [11], However, when there are other transactions in the system, as is typically the case in real production systems, the problem becomes much harder.

In Figure 3, we plot the LRU miss ratio against the ratio of buffer pool size to data size. The intention is to establish a rule of thumb to determine what are reasonable buffer pool sizes relative to the data size. Again, we find that the data tends to follow a function of the form $f(x) = a \cdot (x - b)^c$ where $a$, $b$ and $c$ are constants. As shown in the figure, $c$ is approximately $-\frac{1}{2}$, resulting in a square root rule in this case. Note, however that the production workloads exhibit somewhat diverse behavior. Most of them can effectively utilize buffer pools that are on the order of 3-10% of their data size. The notable exceptions are TelecomA, which has a miss ratio curve that bottoms out at less than 1%, and Bank, which continues to improve in miss ratio with buffer pools that are bigger than 15% of its data size. Although TPC-C's miss ratio continues to improve at buffer pool sizes beyond 10% of its data size, the knee in its miss ratio curve is

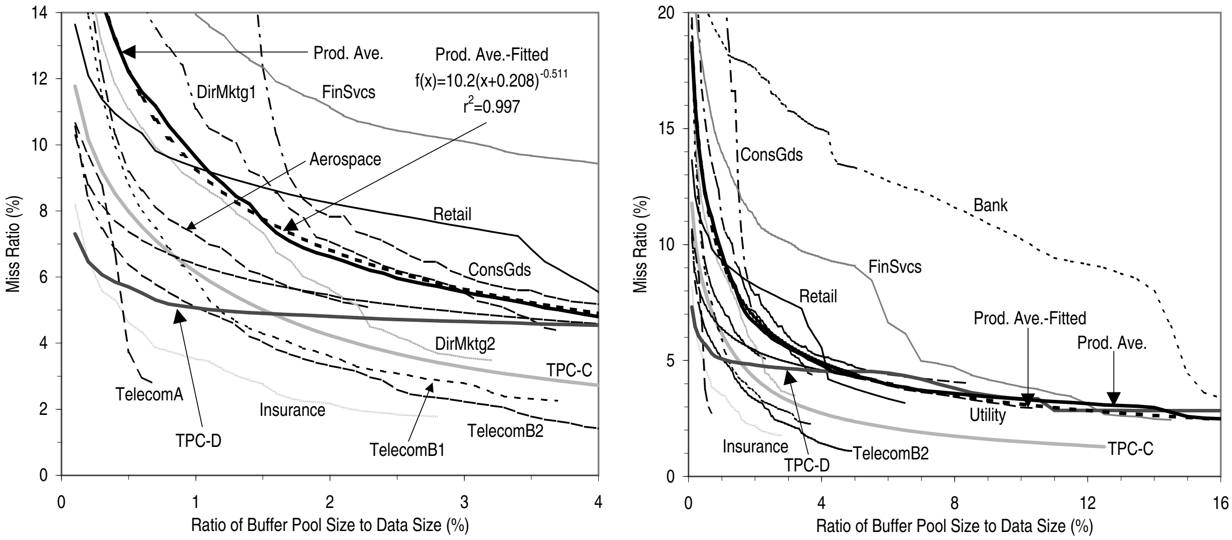



Figure 3: LRU Miss Ratio as a Function of Ratio of Buffer Pool Size to Data Size.

achieved at below 5% of the data size. For TPC-D, the knee in its miss ratio curve occurs at less than 1% but its miss ratio continues to improve at buffer pool sizes beyond 10% of its data size.

Such results are important because they indicate what a balanced database server should look like. For instance, storage-wise, a PC server today can easily accommodate databases that are hundreds of gigabytes in size. The barrier to good performance is likely to be the relatively small amount of memory that can be addressed by today's PC operating system to serve as the buffer pool. For instance, Windows NT 4.0 has a main memory addressing limit of 4GB of which 1GB is reserved for the operating system. This constraint has been deemed serious enough that Intel has announced a hardware architecture that allows main memory beyond the 4GB barrier to be used as a RAM disk [31].

In Figure 4, we consider how long pages tend to remain in the buffer pool by plotting the average buffer pool residency time as a function of the buffer pool size. Because of the large number of production workloads, we also plot the arithmetic mean of their residency time. This is labeled as "Prod. Ave." in Figure 4. Since the production traces are of different sizes, if we simply take the arithmetic mean of the residency time, the number of traces being averaged will decrease with the buffer pool size so that the resulting curve will contain discontinuities. Therefore, we take the mean of the rate of increase of the residency time and then integrate the resulting expression. More formally, we define the arithmetic mean of the average residency time for the production workloads with a buffer pool of size $X$ as $\int_0^X \overline{\frac{d}{dx}(f_i(x))}\,dx$, where $f_i(x)$ denotes the average residency time of trace $i$ with a buffer pool of size $x$.

We find that for the production workloads on average, the relationship between the average buffer pool residency time and the size of the buffer pool can be accurately described by the Hill equation which was originally proposed for modeling the absorption of oxygen by hæmoglobin [25]. The Hill model, $Hill(f_{max}, k, n)$, represents a family of sigmoidal saturation curves defined by $f(x) = \frac{f_{max} \cdot x^n}{k+x^n}$ where $f_{max}$ is the asymptotic value of $f(x)$ and $k$ and $n$ are parameters that determine the shape and slope of the curve. The values of these parameters in our current context are presented in Figure 4. Observe that for the TPC benchmarks, pages tend to remain in the buffer pool for a shorter duration than for the production workloads. This is probably because the production traces were collected on older and slower systems.

For the purpose of establishing a baseline level of miss ratio, we also performed simulations using Belady's MIN, the optimal lookahead or offline page replacement policy [7]. Figure 5 plots the ratio of LRU miss ratio to MIN miss ratio for the various workloads. Note that as discussed earlier, for the individual workloads, we only plot the results for buffer pools that are filled by the warm-start point. However, in computing the average of the production workloads, we always take the arithmetic mean of the results for all the production workloads, regardless of whether the buffer pool is full at the warm-start point. This ensures that the average curve is continuous but it may give the illusion that the average is lower than the curves of which it is the average.

Notice that the ratio of LRU miss ratio to MIN miss ratio is not very stable. This suggests that there are well-defined working set boundaries so that the miss ratio changes abruptly at certain buffer pool sizes. Observe that for buffer pool sizes ranging from 100MB to 1GB, the average LRU miss ratio for the production workloads tends to be almost 40% higher than the MIN miss ratio. This is slightly higher than the 30% difference reported in [82] but is still reasonably consistent with the difference in miss ratio between the

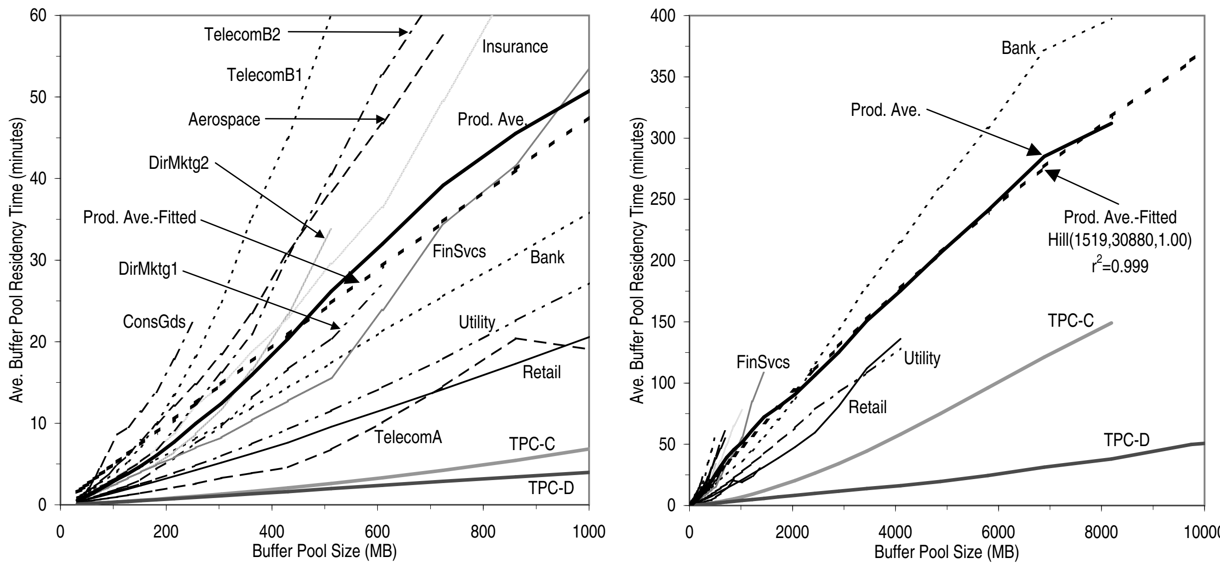



Figure 4: Average Buffer Pool Residency Time.

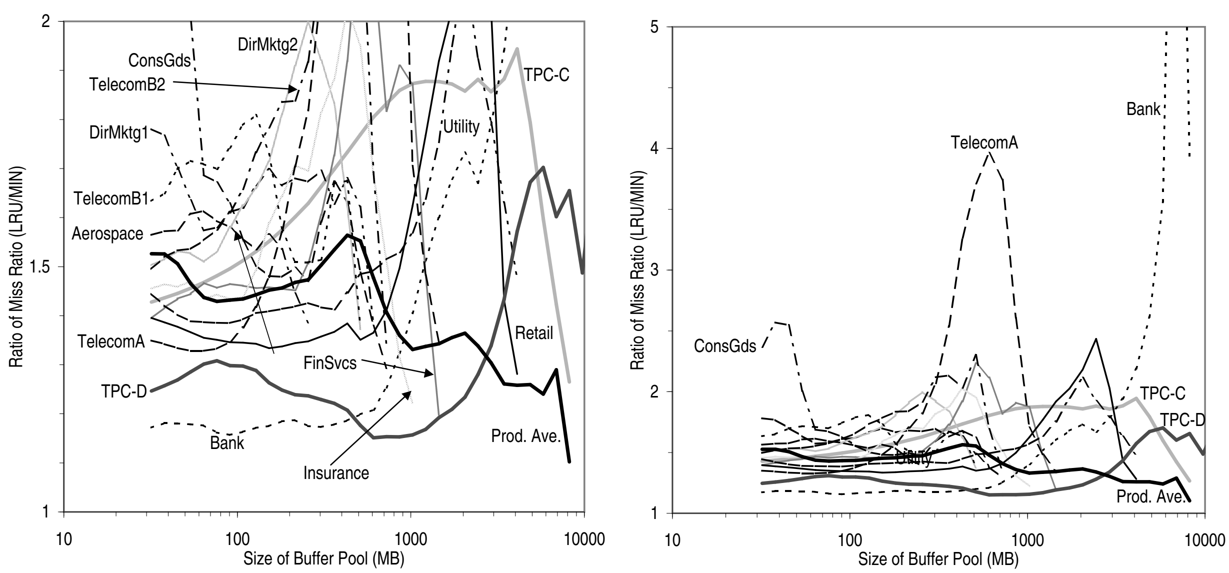



Figure 5: Ratio of LRU Miss Ratio to MIN Miss Ratio.

optimal realizable algorithm and the optimal lookahead algorithm which is reported to be about 35% in [63]. This suggests that on average, LRU is close to the optimal realizable algorithm for the production workloads.

The behavior of TPC-C is different in that its LRU miss ratio is more than 50% higher than its MIN miss ratio for a wide range of buffer pool sizes. The implication of this is that there may be a realizable algorithm that performs significantly better than LRU for TPC-C. On the other hand, TPC-D's LRU miss ratio tends to be much closer to its MIN miss ratio than the production workloads. This is because TPC-D has the tendency to sequentially scan a table or index. In such an operation, all the records on a page are sequentially read. Since each record read generates a page reference, this results in an access pattern where there are multiple references to the same page within a very short period of time. Such degenerate access patterns leave MIN with little advantage over LRU.

Some previous work on analyzing database reference streams, for instance [64], filtered out such degenerate access patterns by deleting immediate rereferences to the same page. However, as we shall see in Section 4.3, our workloads tend to contain complex interleavings of references from different transactions and to different objects so that it is very difficult to accurately identify the degenerate references. Therefore, we chose not to try to filter them out. Furthermore, as discussed earlier, many DBMSs including DB2/UDB maintain performance counters that can be queried and used to calculate the buffer pool miss ratio. If we were to try to remove the degenerate references, our analysis would not be consistent with these numbers which are easily obtainable and and are therefore widely used. See for example [72].

#### 4.2.1 Static vs Dynamic Buffer Management

The Independent Reference Model (IRM) of program behavior assumes that the probability of referencing page $i$ is always $p_i$ where $p_i$ is a constant, $0 \leq p_i \leq 1$ and $\sum_{all\,pages} p_i = 1$ [13]. Under such a model, an exact solution to the LRU miss ratio can be obtained analytically [13, 35]. Approximate solutions have also been presented in [16, 46]. In addition, if such a model is valid, then the buffer pool can be effectively managed by statically allocating space to pages with the highest rate of reference.

Let $dist(i)$ be the distance or time to the next reference of page $i$. The $A_o$ replacement algorithm [1, 13] always replaces the page with the largest expected value of $dist$. Therefore, if the IRM is valid, $A_o$ is the optimal realizable (non-lookahead) replacement algorithm. (Note that $A_o$ will require determination of the reference probabilities, which itself may require lookahead.) In Figure 6, we plot the ratio of $A_o$ miss ratio to LRU miss ratio. From the figure, the $A_o$ miss ratio for all the workloads and in particular the TPC

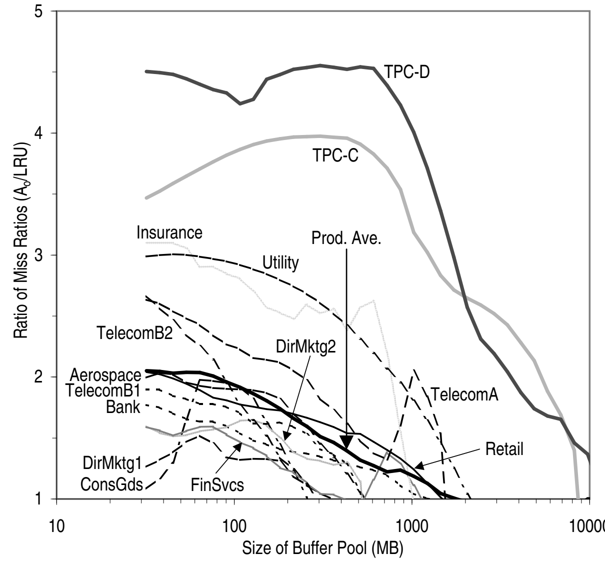



Figure 6: Ratio of $A_o$ Miss Ratio to LRU Miss Ratio.

benchmarks is clearly much higher than their LRU miss ratio. In other words, the $A_o$ algorithm is clearly not the optimal algorithm for these workloads. This implies that the independent reference model for program behavior is not valid for database reference streams and that mathematical models that are based on the IRM are also invalid. In addition, as far as miss ratio is concerned, a statically managed buffer pool is clearly a poor idea, especially for the TPC benchmarks. This is consistent with results presented in [65].

#### 4.2.2 Inter-Transaction Locality

As mentioned earlier, one of the primary weaknesses of buffer management strategies that are based solely on query plan information is that they do not account for interaction between transactions. Since the TPC-D queries are run serially in the power test, this shortcoming should not affect TPC-D. However, the production workloads typically have a high degree of concurrency and involve a complex mix of short transactions and long running queries [26]. Therefore, in this section, we try to quantify the amount of interaction that actually occurs between the transactions.

Let *page reuse* be the ratio of the number of references to the number of pages, *i.e.,*

$$page\ reuse = \frac{\#\ references}{\#\ pages} \quad (1)$$

On a per transaction basis,

$$page\ reuse_{per\ xact} = \frac{transaction\ size}{transaction\ footprint} \quad (2)$$

where following the terminology in [26], *transaction size* is the number of references in the transaction and *transaction* *footprint* is the number of pages referenced by the transaction.

To get an overall picture of the amount of page reuse on a per transaction basis, we compute the average over all the transactions, weighted by the transaction footprint so that transactions with larger footprints are counted more heavily. We refer to this as the intra-transactional page reuse.

$$\begin{aligned}
&\textit{page reuse}_{\textit{intra xact}} \\
&= \frac{\sum_{\textit{all xacts}} \textit{page reuse}_{\textit{per xact}} \cdot \textit{transaction footprint}}{\sum_{\textit{all xacts}} \textit{transaction footprint}} \\
&= \frac{\sum_{\textit{all xacts}} \textit{transaction size}}{\sum_{\textit{all xacts}} \textit{transaction footprint}} \\
&= \frac{\textit{\# references}}{\sum_{\textit{all xacts}} \textit{transaction footprint}} \qquad (3)
\end{aligned}$$

The difference between equations 1 and 3 is that the overall number of pages referenced is less than the sum of the transaction footprints because there is reuse of pages among the transactions. In other words, intra-transactional page reuse is the page reuse that would result if there is no page sharing among transactions; the remaining reuse can be considered to be page reuse between transactions or inter-transactional page reuse:

$$\begin{aligned}
&\textit{page reuse}_{\textit{inter xact}} = \\
&\qquad \textit{page reuse} - \textit{page reuse}_{\textit{intra xact}} \qquad (4)
\end{aligned}$$

In Figure 7, we break down the page reuse in our various workloads into inter-transactional and intra-transactional components. Among the production workloads, total page reuse varies from 13 in ConsGds to over 55 in TelecomB1. On average, total page reuse for the production workloads is about 25 of which 15% can be attributed to page reuse within the same transaction. TPC-C's total page reuse at almost 60 is the highest among all the workloads while TPC-D's reuse at about 17 ranks among the lowest. Observe that with the exception of TPC-D, most of the reuse is the result of page sharing among transactions. As expected, TPC-D's reuse behavior is rather different from that of the other workloads — it has very low inter-transactional page reuse but very high intra-transactional page reuse. TPC-D's high intra-transactional page reuse results from its tendency to sequentially read all the records on a page before moving on to the next page. As discussed earlier, each record read generates a page reference. Therefore, reading all the records on a page will result in multiple references to the same page.

Having seen that most of the page reuse for TPC-C and the production workloads results from interaction between transactions, we now relate this to buffer pool management. In particular, we consider the average *inter-transactional page sharing*, which we define to be the percentage of a transaction's footprint that is already present in the buffer pool as a result of accesses by other transactions. The results assuming an LRU buffer pool are summarized in Figure 8. Comparing the unweighted average in Figure 8(a) to the average that is weighted by the transaction footprint in Figure 8(b), we observe that, as expected, the larger transactions tend to exhibit less inter-transactional sharing. Figure 8(b) shows that with a 1GB buffer pool, we will overestimate buffer space requirements for most of the workloads by more than 80% if we ignore inter-transactional page sharing. The only exceptions are for TPC-D and to a lesser extent, Bank. TPC-D's low inter-transactional page sharing is expected because it contains very long queries that are run serially. Similarly, Bank's behavior can be anticipated from results presented in [26] that show Bank to have long queries and relatively low concurrency. As shown in Figure 8, for the production workloads on average, the relationship between inter-transactional page sharing and the size of the buffer pool can again be accurately described by the Hill equation.

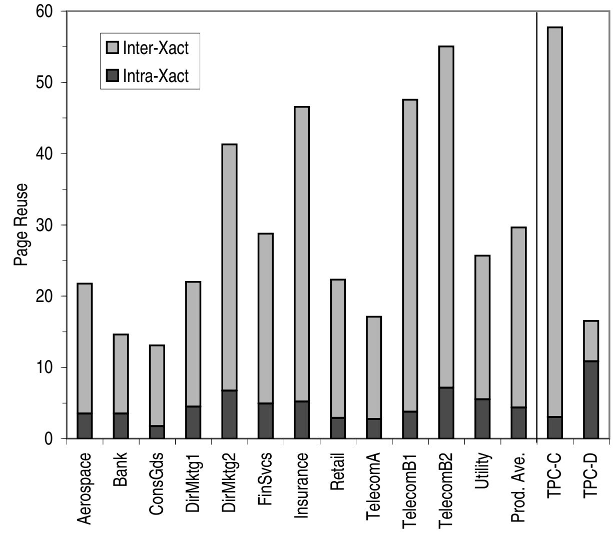
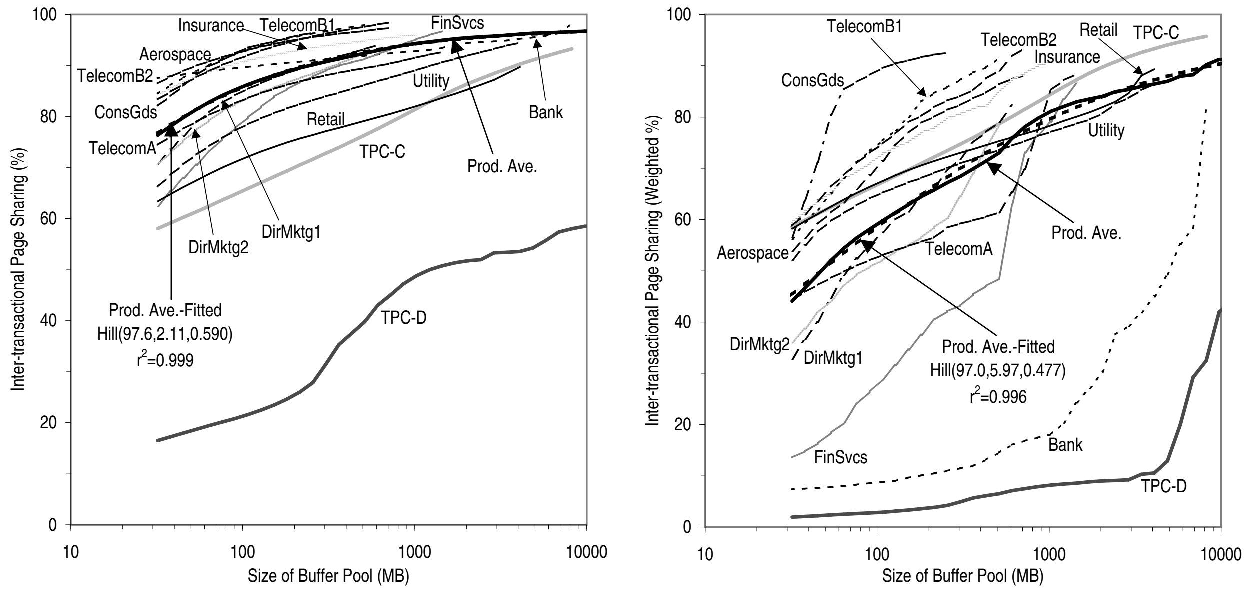



Figure 7: Inter and Intra Transaction Page Reuse.

### 4.3 Sequentiality

Sequentiality is the characteristic that the pages referenced tend to be increasing in page number. Sequential patterns of access allow us to anticipate which pages are likely to be accessed next and to fetch them before they are needed. Such anticipatory fetching of pages is commonly referred to as sequential prefetch. Sequential reference patterns may also allow us to identify pages that are less likely to be reused. In addition, if we discover that the reference




(a) Unweighted Average over all Transactions.

(b) Average Weighted by Transaction Footprint.

Figure 8: Degree of Page Sharing Among Transactions.

pattern is sequential when we are repeatedly using an index to look up a set of keys, then the keys must be sorted or nearly sorted. In this case, we may be able to avoid repeatedly traversing the root and upper levels of the index through the use of index lookaside techniques mentioned in Section 4.1.

This paper focuses on sequentiality in the logical page numbers. However, we note that because the characteristics of I/O devices are such that they operate most efficiently when fetching big blocks, the effectiveness of sequential prefetch depends very much on whether the logically sequential pages are physically sequential on the I/O devices. As a database is updated, data pages may overflow and index nodes may have to be split. In such cases, the physical reference stream may not be strictly increasing. However, if the database is reorganized or dumped and reloaded periodically, logical sequentiality will for the most part correspond to physical sequentiality.

[64] briefly discusses why sequentiality is common in the database reference stream. Other work such as [23, 57, 82] also found sequentiality but several empirical studies of database reference behavior found little or no sequentiality [18, 33, 76]. Whether sequentiality is present in the reference stream clearly depends on the database workload. For relational databases, long-running queries that examine a large number of records and those that involve joins of multiple relations will typically exhibit sequentiality of reference. While the query plan can usually provide some indication of sequentiality, especially in straightforward cases like table scans, some sort of run-time detector is generally needed to fully capture the sequentiality in complex queries.

Most of the previous work used rather strict definitions of sequentiality, the most common of which is that page references are consecutively numbered in ascending order or are separated by a fixed interval [17, 18, 23, 57, 64, 82]. However, in relational databases, the page references may occasionally run backwards in the middle of a forward run. This may occur as the result of a merge join operation that encounters duplicate keys. In addition, it is possible for the page reference stream to be generally increasing but be interspersed by gaps. This typically results from scanning a table using a clustered index as for instance, in an index nested-loop join operation. Moreover, with intra-query parallelism in which a query is concurrently worked on by more than one database agent or thread, references seen on a transactional level may not be strictly increasing.

In order to determine whether such pseudo-sequential reference patterns occur in practice, we consider a reference $r$ to page $p$ to be part of a sequential run $R$ if $p$ lies within $-extent_{backward}$ and $+extent_{forward}$ of the largest page number so far in $R$, where $extent_{backward}$ and $extent_{forward}$ are positive constants. As illustrated in Figure 9, such a definition establishes a high watermark from which the subsequent references may deviate. It captures the case where the page references are generally increasing but may occasionally back up. This definition of a sequential run is a generalization of that defined in [64] where $extent_{backward} = 0$ and $extent_{forward} = 1$. Later, we will

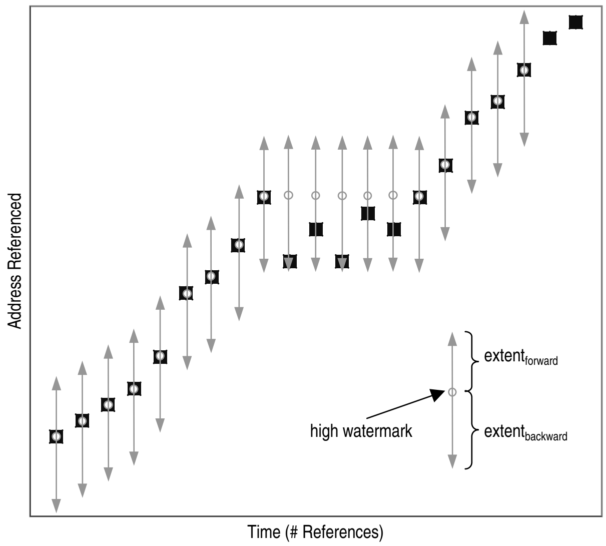



Figure 9: Detecting Psuedo-Sequential Reference Patterns.

examine our workloads to figure out appropriate values for $extent_{backward}$ and $extent_{forward}$ and thereby determine whether pseudo-sequential reference patterns occur in practice. As in [64], we define the *run length* to be the number of references in the run. In addition, we define the *size* of a run to be the number of unique pages in the run, its *span* to be the difference between the largest and smallest page numbers, and its *density* to be its size to span ratio.

It is generally not easy to discover sequentiality in the aggregate reference stream because of the complex interleaving of references from different transactions and to different objects. This is especially the case when there is a high degree of concurrency as is common in production workloads [26]. More recently, [34, 82] examined sequentiality on a per object (file) and/or per transaction basis. But this is still not sufficient because even on a per object and transaction basis, the sequential references may still be interspersed by other references. For instance, in an index scan, the references to the leaf pages of the index will be sequential but will be interleaved with accesses to the root and intermediate nodes of the index. In addition, the accesses to the leaf pages themselves may not be totally sequential if the index has not been reorganized for a while. Moreover, with intra-query parallelism, a simple scan of an object may be broken down into multiple concurrent partitioned scans. Therefore, the reference stream on a per transaction and per object basis may still contain multiplexed pseudo-sequential streams.

This suggests that we need to maintain a buffer of possible sequential runs for each transaction and object combination. In this study, we use an LRU list to implement the run buffer. On a reference, we march down the LRU list to determine the first run that the reference belongs to. If no such run is found, a new run is created and the least recently used run is replaced if necessary. The operation of the run buffer is similar to that of the segmented cache that is implemented in many of today's disks. The number of entries in the run buffer determines the number of runs that can be tracked. This approach is similar to the Sequential Working Set idea in [82] where a page is considered sequential if its predecessor exists in the cache. In our case, we use a separate run buffer instead of the cache directory to remember the recently referenced pages. We also explicitly identify the sequential run so that we can maintain some state for each run. This is useful not only for understanding the characteristics of the runs but also for prefetching where it allows us to condition on the run size.

In Figure 10, we investigate appropriate values for $extent_{backward}$ and $extent_{forward}$. In these simulations, we use a run buffer with 64 entries. We will examine suitable sizes for the run buffer later. TPC-C stands out among the workloads in that it does not show any significant sequentiality and will be omitted from the analysis in this section. Observe from Figure 10 that by relaxing the definition of sequentiality so that the page numbers may occasionally run backwards ($extent_{backward} > 0$), substantially more sequentiality can be detected. When $extent_{backward}$ is twice $extent_{forward}$, even more of the references can be considered sequential. The motivation for having $extent_{forward} > 1$ is that the page numbers may be generally increasing but be interspersed by gaps. In addition to handling such gaps, the backward extent also takes care of situations where the reference stream backs up and where there is a trailing set of subagents working on the same query. Therefore, having the backward extent to be larger than the forward extent is justifiable. For the rest of this paper, we will assume that $extent_{backward}$ is twice $extent_{forward}$.

Observe further that with $extent_{backward} = 0$, increasing $extent_{forward}$ results in the detection of significantly less sequentiality. This is because with $extent_{backward} = 0$, once we make a mistake in considering a forward reference to be part of a run, we cannot go backwards so that the run is essentially terminated prematurely. For instance, with $extent_{backward} = 0$ and $extent_{forward} = 4$, if the reference stream is "1, 2, 5, 3, 4, ...", we would consider "5" to be a continuation of the run "1, 2" and this would prevent "3, 4" from being part of the run. Therefore, having $extent_{backward} \geq extent_{forward}$ is extremely important in that it is forgiving of such mistakes.

In Figure 11, we further examine the relative significance of forward and backward reference patterns by classifying references into different categories depending on whether they can be considered part of an existing run and if so, whether the page numbers referenced are increasing, decreasing or stationary. From Figure 11(a), on average, 21% of the references in the production workloads belong to an existing run and reference a page number that is smaller than the largest page number already in the run. The corresponding figure for TPC-D is 12%. The percentage of references

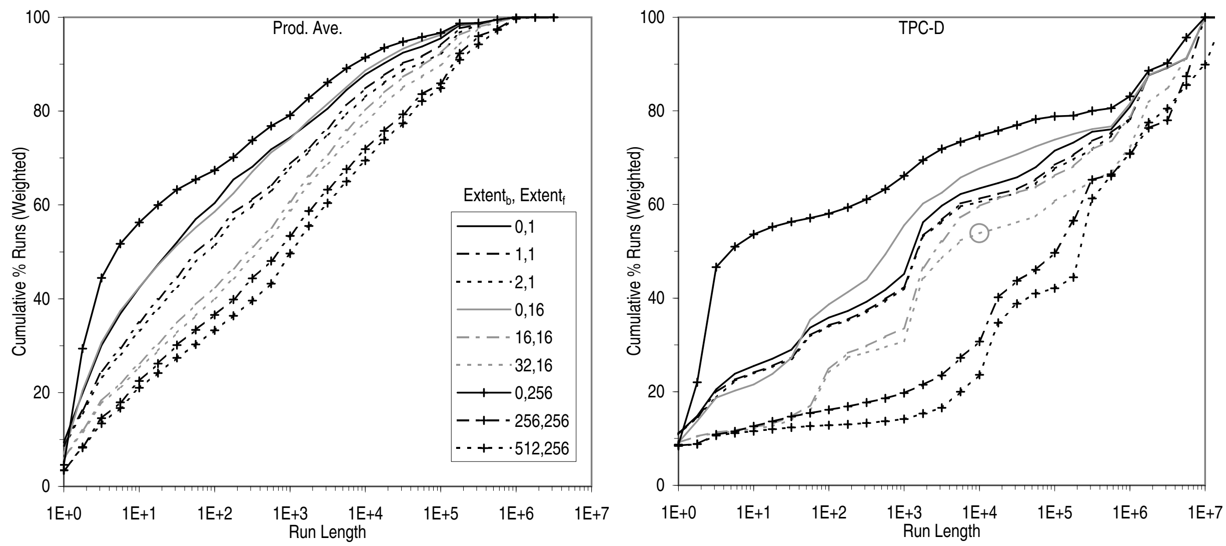
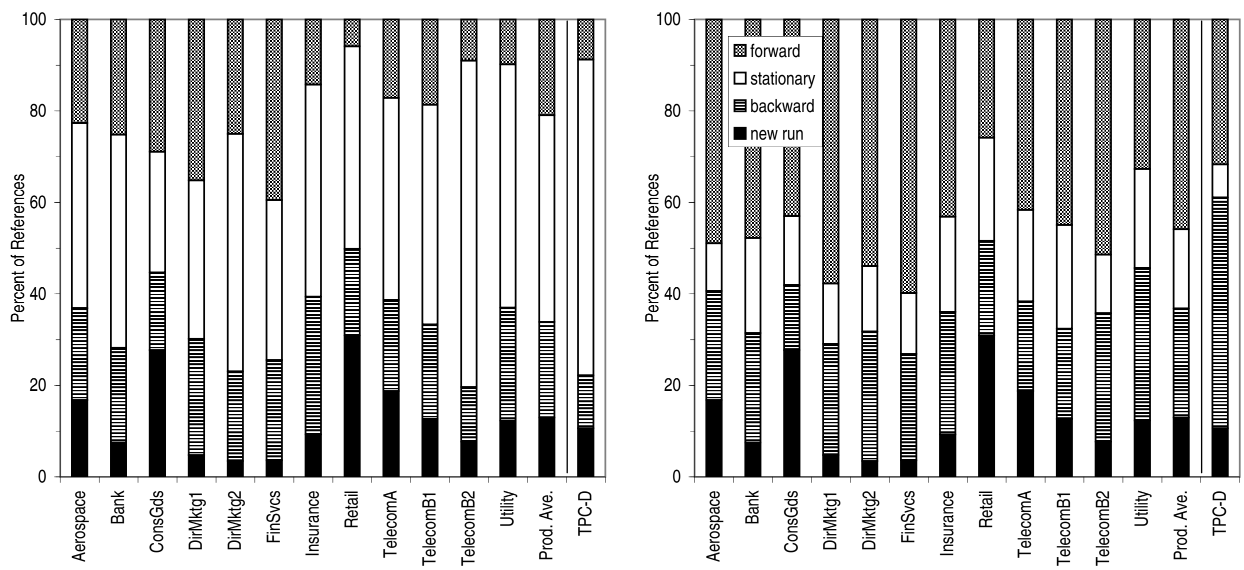


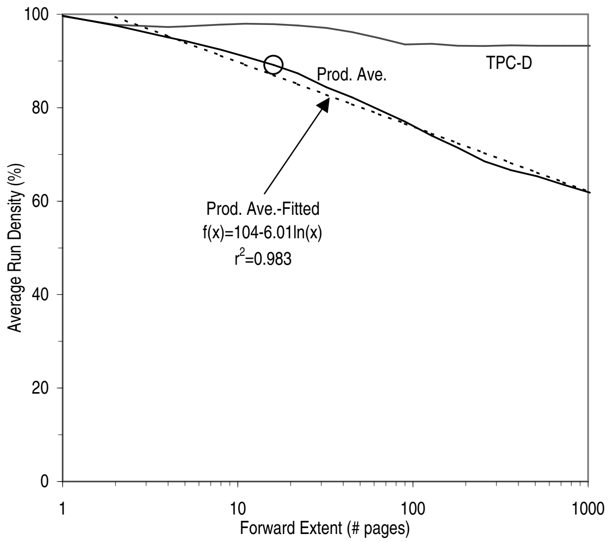

Figure 10: Sensitivity to Extent Size. The distribution of run length weighted by the run length is plotted. The circled point indicates that for TPC-D with a 64-entry run buffer, $extent_{backward} = 32$ and $extent_{forward} = 16$, 54% of the references occur in sequential runs with fewer than 10,000 references.



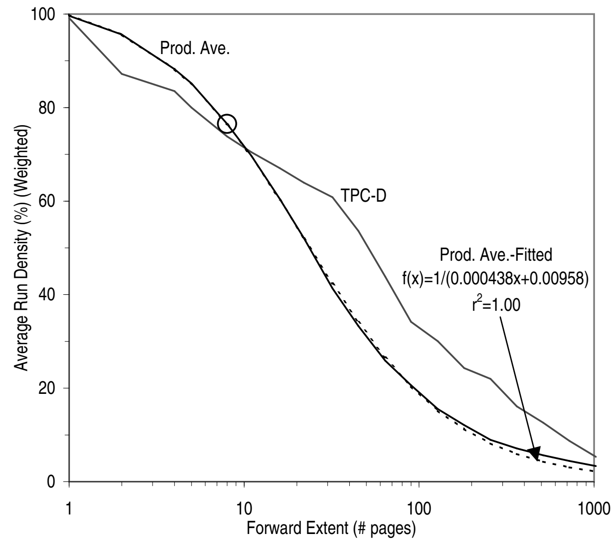

(a) In this figure, we classify the references that are part of an existing run with respect to the largest page number in the run. For instance, if a reference accesses page $p$ where $p <$ *largest page number so far in the run*, it is considered a backward reference.

(b) In this figure, we classify the references that are part of an existing run with respect to the last referenced page number in the run. For example, a reference to page $p$ where $p <$ *last page number accessed in the run* is classified as a backward reference.

Figure 11: Relative Significance of Forward and Backward Reference Patterns. We assume that $extent_{backward} = 16$, $extent_{forward} = 8$ and that the run buffer can track 64 concurrent runs. The references that cannot be considered part of any existing run are denoted as "new run." The remaining references are further classified into those that are part of a backward pattern, those that are stationary and those that are going forward.




(a) Unweighted Average. The circled point indicates that with $extent_{forward} = 16$ and $extent_{backward} = 32$, the production workloads have an average run density of 89%.




(b) Average Weighted by Run Span. In this plot, a run with span $s$ is counted $s$ times. The circled point shows that on average for TPC-D with $extent_{backward} = 16$ and $extent_{forward} = 8$, about 77% of the pages in a run span are actually referenced.

Figure 12: Run Density.

that increase the largest page number in the run is roughly the same. In Figure 11(b), we determine whether the page numbers are increasing or decreasing with respect to the last page number referenced in the run. In this case, 24% of the references jump backwards while 46% of them go forward for the production workloads on average. For TPC-D, the corresponding numbers are 50% and 32%. In either case, backward reference patterns are very significant.

In general, increasing the values of $extent_{backward}$ and $extent_{forward}$ relaxes the definition of a sequential run and enables more sequentiality to be detected. However, the problem with too relaxed a definition is that we may end up with very sparse runs. Recall that we define the density of a run to be the ratio of its size (number of unique pages referenced) to span (difference between largest and smallest page numbers). In Figure 12(a), we present the average run density as a function of the extent size. As shown in the figure, the data follows approximately a logarithmic function of the form $f(x) = a - b \cdot \ln x$ where $a$ and $b$ are constants. Note that in this figure, the runs in each workload are equally weighted. In Figure 12(b), we weight the average by the run span. The rationale for such a weighting is that when performing sequential prefetch, we usually fetch the run span, in which case the average run density weighted by the run span indicates the percentage of useful pages that will be prefetched. We fitted several functional forms to the data and found that a reciprocal function of the form $f(x) = \frac{1}{a \cdot x + b}$ where $a$ and $b$ are constants is a very good fit. Such a fitted function can be used to analytically determine the optimal prefetch policy using a cost and benefit model. See for instance [64]. Observe that with $extent_{backward} = 16$ and an $extent_{forward}$ of eight, about one in four pages in a run span are not referenced, which is pretty reasonable. We will therefore use these values for the rest of the paper.

Finally, we determine suitable sizes for the run buffer; the run buffer keeps track of the length, size and the lowest and highest numbered referenced pages for each run. The results are summarized in Figure 13. For the production workloads on average, a run buffer with four entries is able to capture most of the sequentiality. For TPC-D, a 16-entry run buffer is more appropriate; that size will be used for the rest of the paper.

### 4.3.1 Sequential Prefetch

Having seen that all the workloads except TPC-C exhibit strong sequentiality of reference, in this section we consider how to exploit the sequential patterns of reference to fetch pages before they are needed. Such prefetching of pages has the potential to increase I/O efficiency by transforming several small block I/Os into one large block I/O which can be more efficiently handled by the I/O device. In addition, prefetching may reduce CPU overhead by decreasing the number of start I/Os and the number of transaction blocks

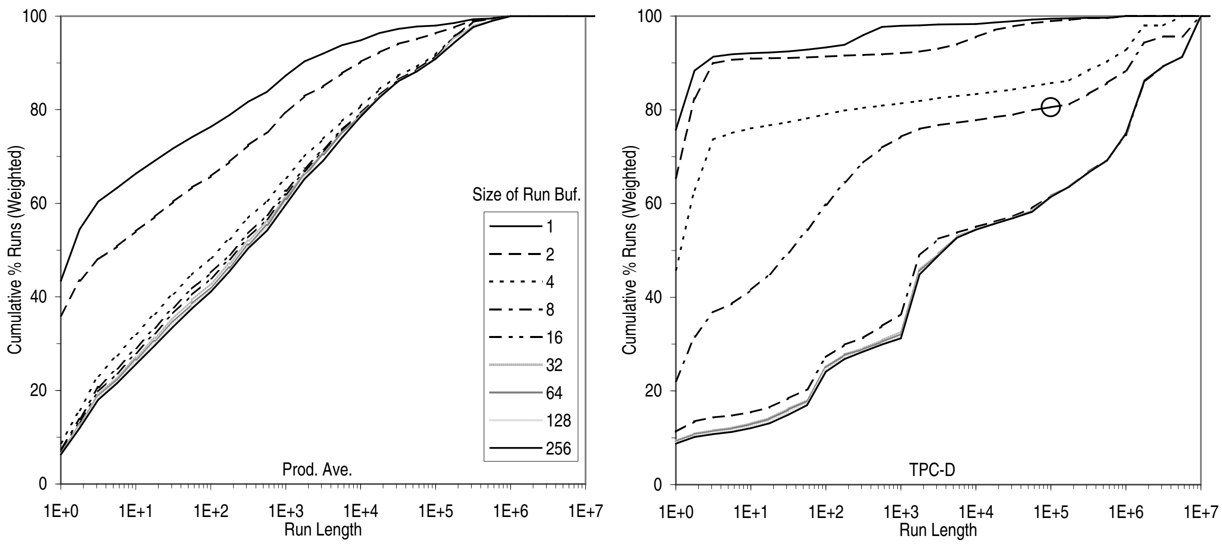



Figure 13: Sensitivity to Run Buffer Size. The distribution of run length weighted by the run length is plotted. The circled point indicates that for TPC-D with an eight-entry run buffer, $extent_{backward} = 16$ and $extent_{forward} = 8$, 81% of the references occur in sequential runs with fewer than 100,000 references.

due to I/O wait. However, inaccurate prefetching can waste resources if pages that are not needed are fetched. In general, the amount of resources that we commit to prefetching should increase with the likelihood that the reference pattern is sequential.

There are generally two different ways to prefetch. In in-line or synchronous prefetching, the prefetch request is issued as a tag along request when a demand fetch occurs. Synchronous prefetching incurs no extra overhead except that the transaction is blocked until the I/O request is completed. It is useful for prefetching small amounts in cases where we are not very certain whether the access pattern is sequential. In asynchronous prefetch, the prefetch request is carried out in the background while the transaction is processed. Asynchronous prefetch of large amounts can be used to speed up a transaction by reducing the need to wait for I/O. However, it tends to be more costly because of the need to initiate and manage asynchronous I/Os. In addition, it has to be initiated well in advance so that pages are prefetched early enough. Therefore, it is generally more useful in situations where we are certain that there is strong sequentiality in the reference stream.

Results presented above show that the various workloads, especially TPC-D, contain some very long sequential runs. In such cases, it generally makes sense to asynchronously prefetch large amounts to keep a transaction fed so as to speed it up. However, this may affect other transactions in the system. In a production environment where there is a complex mix of short transactions and long queries, the overall impact of such aggressive prefetching has to be carefully evaluated. In the TPC-D power test where the performance metric is the run time of a single stream of queries, committing otherwise unused resources to speed up the single stream of queries is clearly beneficial.

The prefetching strategy to use depends critically on the size of the sequential runs. In Figure 14, we plot the distribution of the run size weighted by the run size for the selected configuration (16-entry run buffer, $extent_{backward} = 16$, $extent_{forward} = 8$). Such a weighting allows one to consider the number of pages in the runs rather than the number of runs and is more indicative of the importance of sequential prefetch. To make the data more useful for mathematical modeling and analysis, we fitted it with standard probability distributions. As shown in the figure, the weighted run size distribution for the production workloads tends to follow the lognormal distribution (denoted LogNorm($\mu$, $\sigma$) where $\mu$ is the mean and $\sigma$ is the variance). Note that because the data are plotted on a logarithmic scale, the fit at small values of run size appears poorer than it actually is.

In Figure 15, we plot the expected future run size, $E[FRS(x)]$. This is defined as the expected remaining run size given that the run size has already reached $x$. More formally,

$$E[FRS(x)] = \sum_{i=x+1}^{\infty} \frac{(i-x)l(i)}{1-L(i)} \tag{5}$$

where $l(\cdot)$ is the probability distribution of the run size, *i.e.,* $l(j)$ is the probability that a run has a size of $j$ and $L(\cdot)$ is the cumulative probability distribution of the run size, *i.e.,* $L(j) = \sum_{i=1}^{j} l(i)$. Note that each of the production workloads has a different maximum run size so that the expected remaining run size becomes zero for the different workloads at different values of run size. In Figure 15, we only plot the

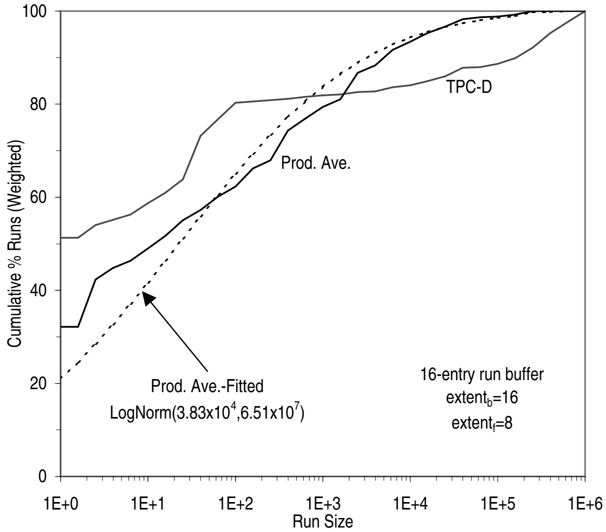



Figure 14: Distribution of Run Size Weighted by Run Size.

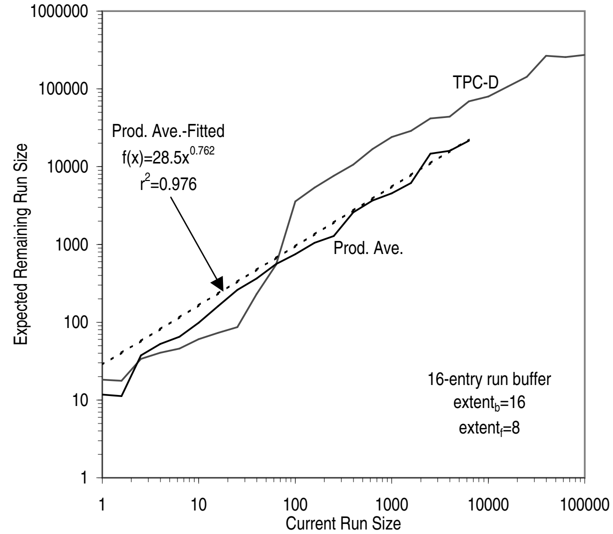



Figure 15: Expected Remaining (Future) Run Size.

average of the production workloads up to the point where the expected remaining run size of one of the workloads become zero. Observe that as in [63, 82], E[FRS(x)] is generally increasing. In other words, the larger the current run size, the longer the run is likely to continue. This suggests prefetch policies that progressively fetch larger numbers of pages as the run size increases.

With the intent of better understanding how the various workloads respond to prefetch, we generalize the hazard rate used in [64]. We define the *prefetch Hazard* $H_{pf}(k, r)$ to be the probability that a sequential run ends with size $\leq k + r$ given that its size is at least $k$. In other words, given a run that has reached a size of $k$, $H_{pf}(k, r)$ is roughly the chance that we will overfetch if we prefetch more than $r$ pages. More formally,

$$H_{pf}(k, r) = \frac{\sum_{i=0}^{r} l(k+i)}{1 - L(k-1)} \tag{6}$$

Figure 16 summarizes the average prefetch hazard for the production workloads and for TPC-D. The individual plots for the production workloads can be found in Figures A-1.i and A-1.ii in the Appendix. Observe that the hazard rate is generally declining as the run size increases. A high hazard rate indicates that the prefetched blocks will not be used and conversely. Therefore these results again support the idea of determining prefetch amounts by conditioning on the run size seen so far [64]. Notice that as expected, the hazard rate increases with $r$. In other words, the chances that the prefetched blocks will not be used increases with the prefetch amount. As mentioned earlier, in order to keep a query fed, asynchronous prefetch typically has to be initiated well in advance. Suppose that in order for the prefetch to be completed on time, we need to initiate it $x$ pages in advance. In this case, if the run is already of size $k$, then $H_{pf}(k, r+x)$ is approximately the probability that a prefetch amount of $r$ will result in an overfetch. In view of these considerations and the shape of the curves in Figure 16, a good prefetch strategy is to perform in-line prefetching of small amounts initially and to switch to asynchronous prefetching of large amounts when a long run is detected.

Although all the workloads except TPC-C share the characteristic that their reference behavior is highly sequential in nature, their exact behavior varies from one workload to another, as is apparent from Figures A-1.i and A-1.ii in the Appendix. This suggests that it may be worthwhile to have an adaptive prefetch strategy that dynamically accounts for these differences to determine, for instance, the prefetch amount. Such an adaptive strategy will be especially useful if the sequential behavior is non-stationary, as has been suggested in [64], Generally, the prefetch policy to use depends, among other things, on the system and workload characteristics as well as the performance metric. Determining the optimal prefetch policy is beyond the scope of the current study, which only examines the I/O characteristics at the macroscopic level and leaves the refinement of specific techniques to more focused future research.

In Figure 17, we examine the effectiveness of a prefetch policy that on encountering a run of size at least 16, fetches the next eight pages if fewer than four of these pages are already in the buffer pool. The production workloads again exhibit a wide range of responses. Prefetching reduces the LRU miss ratio of Bank by about 90% but reduces that of Retail by only less than 20%. On average, the miss ratios of the production workloads are decreased by almost 60%. This is significantly better than the 30% improvement that has previously been reported for commercial database

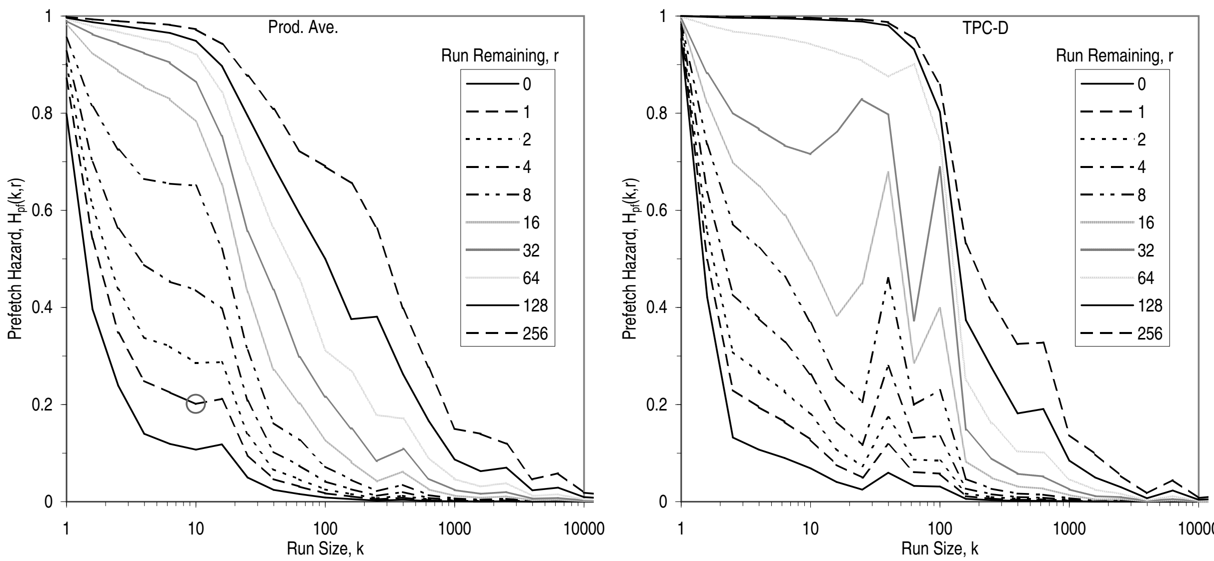



Figure 16: Prefetch Hazard. The circled point indicates that for the production workloads on average, there is a 20% chance that a run is of size 11 given that it has reached a size of 10.

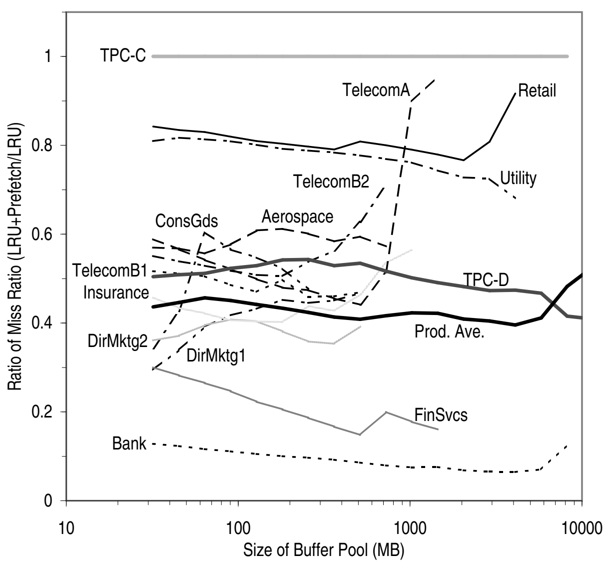



Figure 17: Miss Ratio Improvement with Prefetch. In these simulations, we fetch the next 8 pages whenever a run that has a size of at least 16 is encountered.

workloads[64, 82] for simpler sequential prefetching policies. We believe part of the reason for the better results lie in our more elaborate sequential detector. Observe further from Figure 17 that TPC-D's miss ratio is reduced by 50%, which is well within the spectrum defined by the production workloads. As expected, TPC-C, which shows no significant sequentiality, is not affected by the prefetch.

Note that the LRU miss ratio generally decreases with bigger buffer pools. Therefore, if prefetching reduces the number of misses by a constant amount as the buffer pool is increased in size, we expect the ratio of prefetch miss ratio to LRU miss ratio to decrease with buffer pool size. This is the case for some of the workloads. For most of the workloads, the curves are relatively flat. For the remaining workloads, notably, TelecomA, the opposite happens. For these workloads, the prefetch benefit is greatly reduced with the bigger buffer pools because of the presence of loops that just fit within the bigger buffer pools.

A major problem with fetching pages before they are really needed is that such prefetching may dramatically increase the amount of I/O traffic because of useless prefetches. We consider this in Figure 18. Observe that our prefetch criteria seems to be very accurate for practically all the workloads. This is consistent with results presented in [64, 82] for a prefetch policy that determines prefetch amount by conditioning on the run length. The only exceptions are TPC-D and TelecomA. The behavior of TelecomA as the buffer pool is increased in size is especially disturbing. A deeper analysis reveals that TelecomA contains some very sparse sequential runs that are not effectively handled by the buffer pool. As the buffer pool is increased in size, more and more of the other runs, which are denser, are cached within the buffer pool. This causes the overall prefetch accuracy to become less and less accurate and eventually leads to the rather dramatic increase in traffic ratio. A more sophisticated prefetch mechanism that dynamically monitors the prefetch accuracy on a per run basis would help to solve this problem with TelecomA.

Another issue in prefetching is to decide how to treat the pages that have been prefetched with respect to replacement. These pages can be entered into the buffer pool as if they have been referenced but this may pollute the buffer pool

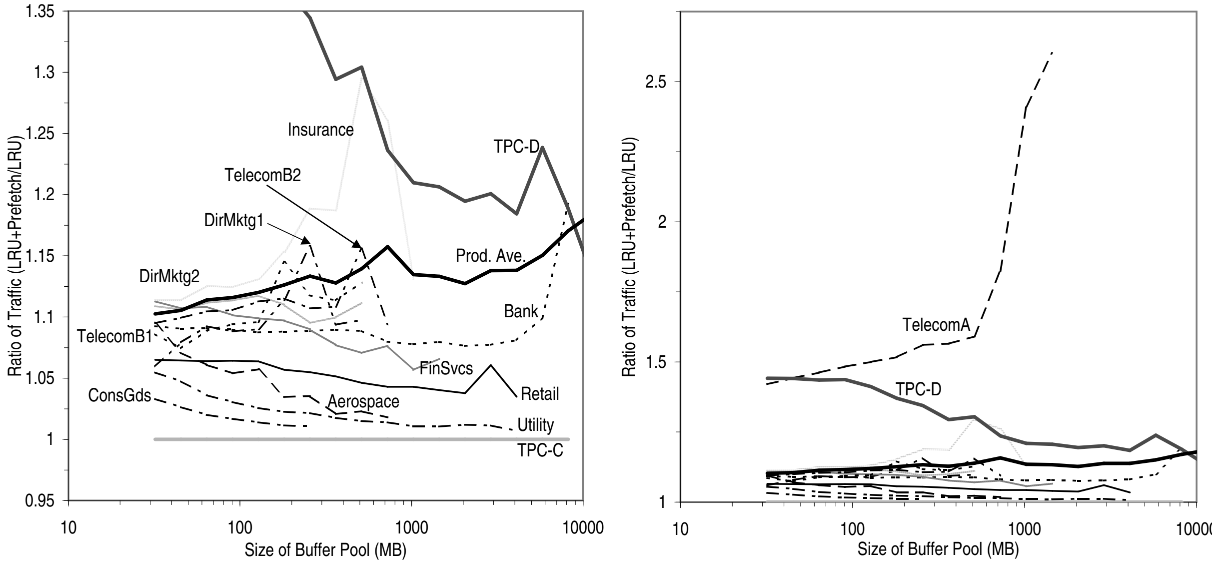



Figure 18: Increase in I/O Traffic with Prefetch. In these simulations, when a run of size at least 16 is encountered, we fetch the next eight pages if fewer than four of these pages are already in the buffer pool.

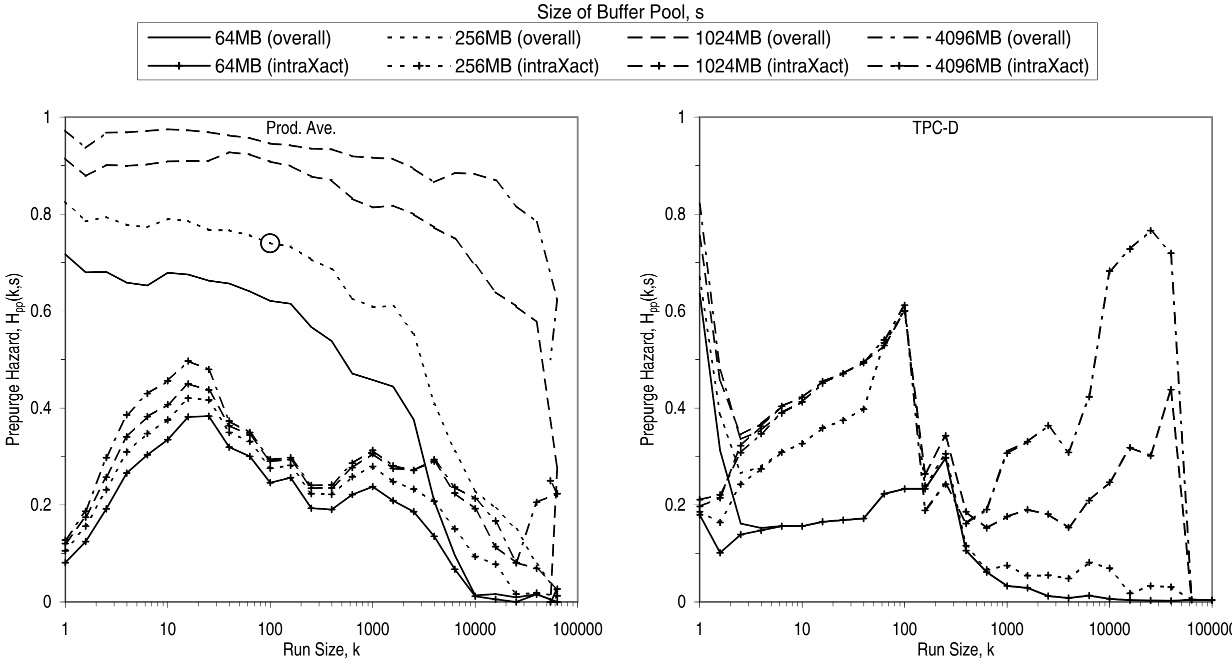
with pages that will never be accessed. An alternative is to hold these pages in a separate prefetch buffer but then the issue becomes one of allocating space between the prefetch buffer and the buffer pool. A general approach is to enter the prefetched pages into the buffer pool at a certain depth in the LRU stack to control pollution. We refer to this depth as the *prefetch depth*. The idea was investigated in [64] and no noticeable difference was found. Nevertheless, recent patents such as [77] suggest that a pollution prevention mechanism for prefetched pages may be important. Therefore, we reexamine the issue by varying the prefetch depth from zero, *i.e.,* the MRU position, to three-quarters of the LRU stack. The only workload to display any noticeable response was TelecomA. Even then the difference was not significant. Since varying the prefetch depth is essentially a way to control pollution of the buffer pool by the pages that have been prefetched, that there is no significant effect is a testament to the accuracy of our prefetch.

### 4.3.2 Sequential Prepurge

[68] suggests that sequentially accessed pages may be much less likely to be rereferenced than randomly accessed pages and should therefore be handled differently by the buffer pool replacement algorithm. In this section, we consider whether it is beneficial to purge a page after it has been sequentially accessed. We refer to this as prepurging and define the *prepurge hazard* $H_{pp}(k, s)$ to be the probability that a page referenced in a sequential run of size $k$ is referenced again in another run before it is evicted from a buffer pool of size $s$. We further define intra-transactional prepurge hazard to be the special case where the rereference is by the same transaction. Figure 19 presents the average prepurge hazard for the production workloads and for TPC-D. The individual plots for the production workloads can be found in Figures A-2.i and A-2.ii in the Appendix.

Note that $H_{pp}(1, s)$, the prepurge hazard corresponding to a run of size one, is essentially the probability that a randomly referenced page will be reused before it is evicted from the buffer pool. On average for the production workloads, this value is higher than that for the sequentially accessed pages but not by much (Figure 19(a)). In other words, although it is true that sequentially accessed pages are less likely to be rereferenced than randomly accessed pages, the probability for them to be rereferenced is generally not low, especially for the larger buffer pools. Hence we should exercise caution in prepurging sequentially accessed pages. Moreover, Figures A-2.i and A-2.ii in the Appendix show that the individual behavior of the production workloads is rather diverse in this regard, suggesting that no single prepurge policy will work well across all these workloads and that any prepurge mechanism will probably have to be tuned to each environment or be made self-tuning. Notice that TPC-D's prepurge hazard tends to be much lower than that of the production workloads. This is not surprising since the TPC-D workload is dominated by very long queries that are run serially so that there is little chance for a sequentially referenced page to be still in the buffer pool when it is next needed. Nevertheless, the prepurge hazard for TPC-D is still significant, indicating that prepurge is probably not a good idea for TPC-D either.

In relational databases, the query optimizer can usually provide some indication as to whether a sequential run is part of a cyclic reference pattern and should therefore not be pre-




(a) The circled point indicates that for the production workloads on average, there is a 74% chance that a page referenced in a run of size 100 will be rereferenced before it is evicted from the buffer pool.

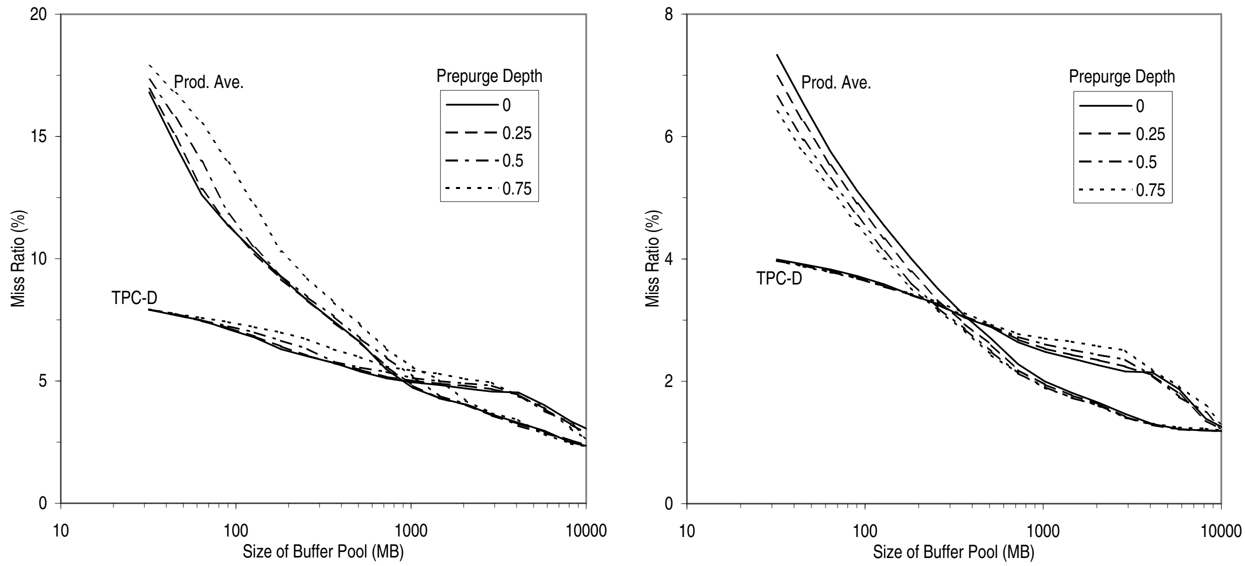
(b) The curves for the overall prepurge hazard and the intra-transactional prepurge hazard are overlaid because there is very little interaction between transactions in TPC-D.

Figure 19: Prepurge Hazard.

purged. However, this sort of indication only covers rereferences by the same transaction. As is apparent from Figure 19 and Figures A-2.i and A-2.ii in the Appendix, there is actually quite a bit of inter-transactional reuse of sequentially accessed pages in the production workloads. TPC-D stands out clearly among the workloads in that it exhibits very little inter-transactional reuse of sequentially accessed pages. As discussed earlier, this can be attributed to the fact that the queries in TPC-D access a lot of data and are run one after another in a serial fashion, thereby reducing any chance of inter-transactional reuse. The implication of this is that strategies that consider only reference behavior on a per transaction level will perform disproportionately well for TPC-D.

It is generally not a good idea to immediately evict the prepurged pages because they may still be needed. One approach is to place them in a separate buffer like a victim buffer but as in the case of the prefetch buffer, the issue becomes one of allocating space between the buffer pool and the separate buffer. In Figure 20(a), we investigate the idea of placing prepurged pages at a certain depth in the LRU stack. In these simulations, whenever a page is accessed in a run that has a size of at least 16, we place it at depths ranging from zero (MRU position) to three-quarters in the LRU stack . Observe that for the most part, the idea of prepurging pages performs poorly for both the production workloads and for TPC-D, which is consistent with results reported in [65] and is not surprising in view of our earlier results on the prepurge hazard.

In Figure 20(b), we perform both prefetching and prepurging, in which case prepurging yields a lower miss ratio for the production workloads. This indicates that the sequentially accessed pages are reused sequentially so that the prepurged pages can be effectively prefetched if necessary. Whether this is actually a performance improvement requires a more detailed study using a timing model but we believe that it is generally not worthwhile unless the system has a lot of I/O bandwidth but is memory limited. Observe that TPC-D is unlike the production workloads in that prepurging, even with prefetching, increases the miss ratio. This is because in our TPC-D setup, we aggressively have multiple database agents working concurrently on the same query and these agents may go out of synchronization in complex join operations. If prepurging is performed in this case, the same page may have to be fetched more than once by the different agents.




(a) In these simulations, a page is prepurged if it is accessed in a run that has a size of at least 16

(b) In addition to prepurge, we also prefetch the next eight pages when a run that has a size of at least 16 is encountered and if fewer than four of the pages to be prefetched are already in the buffer pool.

Figure 20: Effect of Prepurge Depth.

### 4.4 Write Behavior

From a performance perspective, the write behavior of a workload is an extremely important characteristic because writes or update operations complicate a system and throttle its performance. For instance, a static database can be simply replicated to improve not only its performance but also its scalability and durability. But once there are writes in the system, the system has to ensure that the writes occur in the correct order and has to propagate the results of each write to all possible replicated or derived copies such as the parity in a parity-protected striped disk (*e.g.,* RAID-5) storage system. The intensity of these operations will depend largely on the percentage of writes in the reference stream. From the read ratio reported in Table 1, the production workloads are very dissimilar in this regard. Insurance has the lowest read to write ratio of 5.6:1 while TelecomB2 has the highest of 51.4:1. TPC-C and TPC-D lie within this very broad spectrum with 6.9:1 and 45.0:1 respectively.

These numbers are in line with those reported in [82] but they are significantly higher than what has been measured at the physical level. For instance, measurements conducted at the physical level at 12 moderate-to-large MVS installations found the read-to-write ratio to be about 3.5:1 [39]. Part of the reason for this difference is that main memory buffering is more effective at reducing read traffic than write traffic, especially when modified pages must be written to disk in a timely fashion to reduce the possible loss of data in case of a system crash. We will examine the effectiveness of write buffering later in this section. Another reason for our higher read-to-write ratio is that we did not consider activity to the database logs which are known to be dominated by writes. In addition, our trace data reflects only database system I/O and not whatever I/O may have been generated by the operating system or other applications. Measurements reported in [39] show that the read-to-write ratio for storage used by DB2 is 20:1, which is much higher than the overall.

[26] shows that the fraction of data that is updated in the TPC benchmarks is much higher than in the production workloads. The fraction of dynamic data is affected by the relative length of our traces although Bank, which has a duration of nearly a day, still has a very small fraction of dynamic data. In general, if the dynamic portion of the database is relatively small and stable, handling dynamic data differently may result in better performance and lower overall cost. For instance, one technique is to place only the dynamic portions of the database in RAID-1, also known as mirrored disks, and the rest in RAID-5 [79]. Though this may be an interesting idea, our traces are not long enough for us to investigate this in enough detail.

Perhaps the most important technique to improve write performance is to delay the writes so as to allow write coalescing to take place. Write coalescing can generally occur in two ways. First, multiple logical updates of the same page

| Trace | Aerospace | Bank | ConsGds | DirMktg1 | DirMktg2 | FinSvcs | Insurance | Retail | TelecomA | TelecomB1 | TelecomB2 | Utility | Prod. Ave. | TPC-D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Inst. | 4.48 | 322 | 0.406 | 17.4 | 43.7 | 96.6 | 26.7 | 20.0 | 15.1 | 43.1 | 5.57 | 4.20 | 49.9 | 1555 |
| 100-ms | 4.44 | 322 | 0.398 | 17.4 | 43.6 | 96.5 | 26.6 | 20.0 | 14.8 | 43.0 | 5.52 | 4.20 | 49.9 | 1554 |
| 1-s | 4.20 | 322 | 0.397 | 17.4 | 43.4 | 96.1 | 25.9 | 20.0 | 13.1 | 42.4 | 5.05 | 4.13 | 49.5 | 1548 |
| 10-s | 2.41 | 321 | 0.346 | 11.3 | 40.4 | 92.6 | 18.5 | 19.3 | 8.47 | 41.0 | 3.69 | 4.11 | 46.9 | 1500 |
| 1-min | 0.81 | 316 | 0.118 | 8.54 | 33.1 | 76.5 | 14.0 | 18.3 | 2.60 | 31.4 | 2.65 | 3.92 | 42.3 | 1391 |
| 10-min | 0.14 | 302 | 0.0212 | 1.34 | 10.2 | 70.4 | 4.08 | 10.6 | 0.266 | 10.6 | 1.74 | 1.36 | 34.4 | 1093 |
| 100-min | 0.0452 | 150 | 0.0081 | 0.781 | 5.62 | 51.3 | 0.101 | 2.94 | 0.0535 | 4.08 | 0.695 | 1.24 | 18.1 | 553 |
| Trace Len. | 0.0324 | 18.0 | 0.0103 | 0.750 | 3.26 | 34.2 | 0.329 | 1.83 | 0.0437 | 1.76 | 0.420 | 0.572 | 5.10 | 234 |

Table 3: Temporary Space Usage (MB) Averaged over Various Time Intervals. The table shows the peak or maximum value observed for each interval size. The instantaneous maximum is denoted by “Inst.”. The row labeled “Trace Len.” is essentially the temporary space utilization time-averaged over the entire trace.

may be combined into a single physical write of the page, thereby reducing the number of actual physical writes that have to be performed by the system. This is especially beneficial to log-structured filesystems/arrays [40, 58] because it reduces the need to perform garbage collection. Second, writes to sequential pages may be merged into a big block I/O, which can be more efficiently handled. In particular, writing big chunks at a time helps to reduce the penalty for small writes in a RAID-5 storage system. Since our focus in this paper is on the logical I/O characteristics, we will concentrate on understanding the first type of write coalescing. In a subsequent study, we intend to investigate the second type of coalescing and its effect on the physical storage system.

#### 4.4.1 Persistent vs Temporary Writes

The write operations in a system can usually be divided into *persistent writes*, or writes to permanent objects, and *temporary writes*, or writes to temporary objects. In order to buffer persistent writes in memory without opening up a window within which committed updates may be lost, most database management systems log the updates to stable storage such as disk before allowing the changes to be committed. This is known as Write Ahead Logging (WAL) [43]. With WAL, the destage policies used in the database buffer pool can be more elaborate than those used in filesystem caches, where the age of “dirty” or modified pages must be bounded in order to restrict the loss of data in case of a system crash.

Nevertheless, it is still prudent to not hold dirty data indefinitely in main memory. The primary reason is that database recovery time depends on how old the memory pages are at the time of a crash. Furthermore, when the page to be replaced is dirty, it has to be written back before the buffer space can be reused. Moreover, the storage system may perform better when a page is updated soon after it is read. For instance, in a RAID-5 storage system, the write penalty for generating the new parity can be reduced if the page is still present in the storage cache at the time of a write. Therefore, as a general rule, the dirty pages should be kept in memory only to the point where write coalescing becomes insignificant. In Section 4.4.2, we will try to determine this point for the various workloads.

Temporary objects are used in the processing of certain SQL statements that require working storage space. For instance, when a sort requires more memory than has been allocated for the sort heap, a multi-phase sort [36] is typically used and the intermediate runs of the sort operation are stored in temporary objects. Writes to such temporary objects do not have to be recovered in a system crash. Furthermore, if the temporary writes are held in memory beyond the lifetime of their corresponding objects, they do not have to be written to disk. In Table 3, we summarize the amount of temporary or working space that is used by the various workloads. The values in this table are obtained by time-averaging the temporary space utilization over various time intervals and then taking the maximum values observed for each of the interval sizes. In Figure 21, we show the temporary space usage as it varies over time. As mentioned earlier, TPC-C is unique among the workloads in that it does not contain any references to the temporary objects. It is therefore omitted from both Table 3 and Figure 21.

Results presented in [26] indicate that temporary writes account for a very significant portion of the write activity. Specifically, 41.5% of the writes in TPC-D are to temporary objects. The corresponding figure for the production workloads on average is 47.6%. Nevertheless, the maximum temporary space requirement for all the workloads except Bank, FinSvcs and TPC-D is well under 50MB at any one time. This small amount of working space can be easily kept in main memory. However, Bank, perhaps FinSvcs and especially TPC-D have large temporary space requirements that cannot be easily accommodated in main memory. In these cases, we have to decide how to allocate buffer space between the various competing demands and in particular,

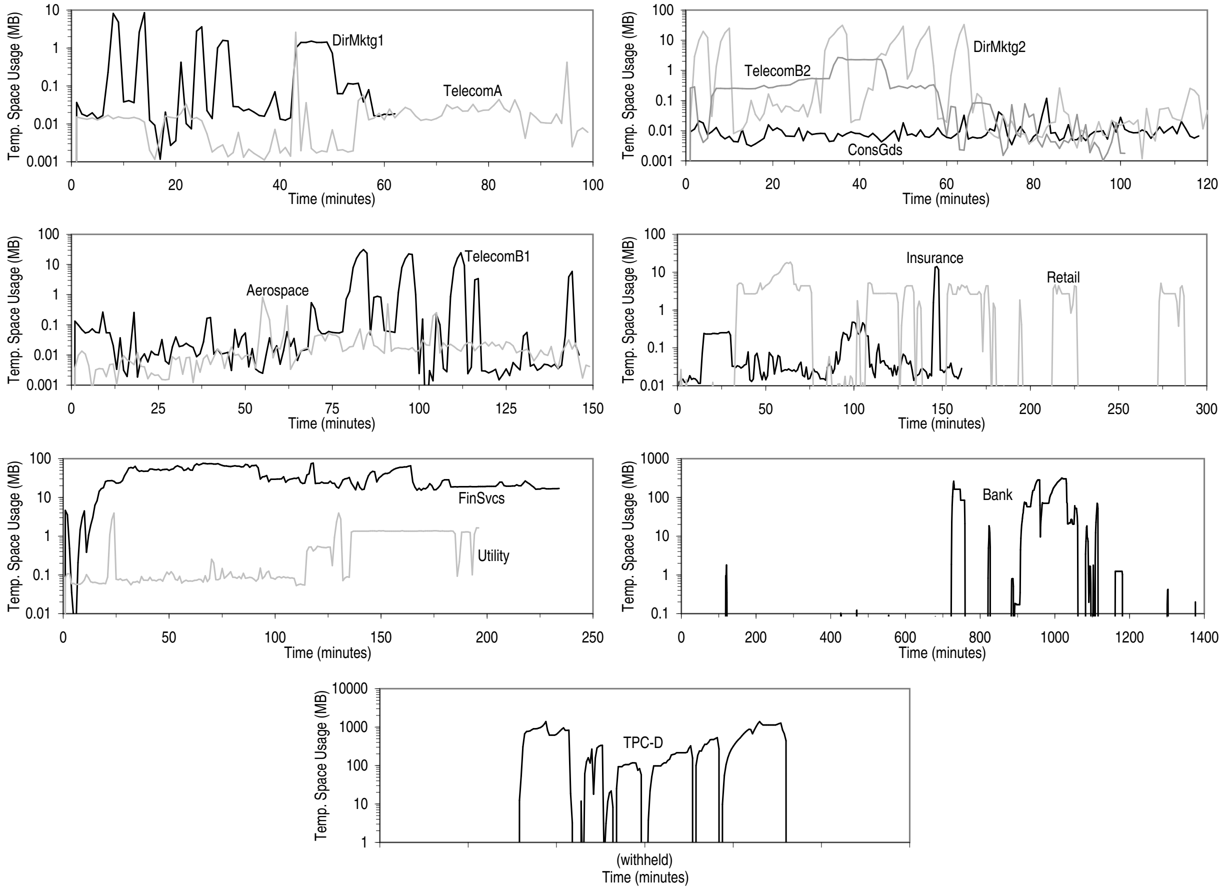



Figure 21: Temporary Space (Work File) Usage over Time. The data in this figure have been smoothed by averaging over one-minute intervals.

when to spill the temporary writes to disk and in what order. Note that, for a given memory size, the I/O behavior corresponding to temporary objects is generally well-understood on a per database operation basis. See for example [21]. Therefore, the temporary objects can be managed using hints based on the database operation as in [22]. However, as we have discussed, production workloads tend to have a complex mixture of concurrent transactions competing for resources so that it is difficult to reliably estimate the amount of memory that will be available for a particular database operation.

### 4.4.2 Write Buffering

In database systems, the technique of delaying or buffering the writes is typically implemented by allowing pages in the buffer pool to exist in a modified state. This allows the write buffer to vary in size as needed up to the size of the buffer pool. Such a design makes sense because the set of pages that are read tends to overlap with the set of pages that are written. Furthermore, sharing a common pool of buffer space between caching reads and buffering writes allows the allocation to dynamically adjust to demands.

For the purpose of understanding the effectiveness of write buffering at reducing the amount of physical writes, we introduce the metric *write miss ratio*. This is defined as the fraction of logical writes that ultimately have to result in physical I/Os for writing or destaging the modified pages to persistent storage. More specifically,

$$\textit{write miss ratio} = \frac{\#\,\textit{write buffer misses} - \#\,\textit{modified pages purged}}{\#\,\textit{logical writes}} \quad (7)$$

A write is considered to be a write buffer miss if the page being written to is either not in the buffer pool or is in a clean state in the buffer pool. Pages are purged when the object to which they belong is deleted. This typically happens only for temporary objects. Notice that the definition of write miss ratio is similar to that usually used for miss ratio in that it measures the ratio of physical to logical I/O operations.

Because variations of the LRU replacement algorithm are often used in database buffer pools, we first investigate how a write back policy will work with such a buffer pool. In this design, the victim page, *i.e.,* the page to be replaced, is

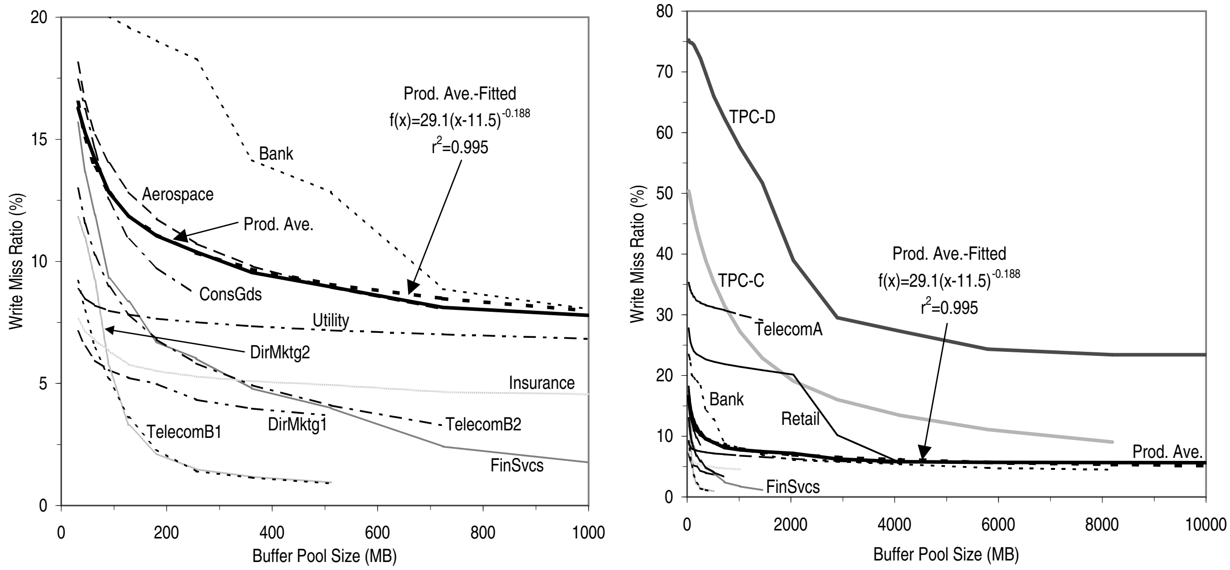



Figure 22: Write Miss Ratio with LRU Write Back Buffer Pool.

the least recently used page. If the victim page is dirty, it is destaged to persistent storage before the buffer space is reused. To avoid having to wait for the destage to be completed, the destage can be performed asynchronously by always keeping the bottom few pages of the LRU stack clean. In Figure 22, we show the write miss ratio for such a design as a function of the buffer pool size.

Notice that Retail stands out among the workloads in that its write miss ratio plot has an obvious concave kink. A deeper analysis reveals that Retail contains cyclic patterns of sequential updates. Observe also that the write miss ratio for TPC-C and especially TPC-D is much higher than that of the production workloads. In addition, we find that except for TPC-C, the write miss ratio for all the workloads tends to bottom out at a smaller buffer pool size than the reference miss ratio, which is presented in Figure 2. Results in [82] indicate that write miss ratio as low as 20% can be achieved with an LRU write back cache. However, [82] did not take into account the purging of temporary objects, which may explain why our write miss ratio numbers are lower. Compared to UNIX filesystem workloads, the production database workloads appear to be more responsive to write buffering. For instance, [49] reports that only up to 75% of the data written in a UNIX filesystem environment are overwritten or deleted before they are replaced.

As in the case of the reference miss ratio (Section 4.2), we use our data for the production workloads to derive an analytical model of the relationship between write miss ratio and buffer pool size. We find that the write miss ratio, like the reference miss ratio, can be accurately modeled by a power function of the form $f(x) = a \cdot (x - b)^c$, where $a$, $b$ and $c$ are constants. As shown in Figure 22, the values of $a$, $b$ and $c$ in this case are 29.1, 11.5 and -0.188 respectively. To make our data more generally useful, we also plot the write miss ratio as a function of the ratio of buffer pool size to data size. This is presented in Figure 23. The average of the production workloads again follows a function of the form $f(x) = a \cdot (x - b)^c$, where $a$, $b$ and $c$ are constants. In this particular case, $c$ is about $-\frac{1}{4}$, meaning that the relationship can be approximately described by a fourth root rule.

Results presented earlier in Figure 4 indicate that pages may remain in large buffer pools for many tens of minutes. If such pages have been modified, the updates may be lost or may have to be recovered in case of a system crash. A straightforward approach to limiting the amount of data loss is to flush the dirty pages periodically. Variations of this policy are used in several flavors of the UNIX filesystem cache [4, 44]. However, periodically flushing all the dirty pages has some bad side effects such as increasing the burstiness of the write traffic. This has been found to lengthen the mean response time for read operations and to add to its variance [8, 42]. Therefore, we consider cleaning a dirty persistent page only when it is older than a specified age limit. Figure 24 presents the write miss ratio that can be achieved with an LRU write back cache for different age limits.

On average, the write behavior of the TPC benchmarks is clearly unlike that of the production workloads. Among all the workloads, TPC-D has the highest write miss ratio and is the least responsive to more relaxed age limits. This can be attributed to the fact that most of TPC-D's writes are to temporary objects which are not affected by the age limit. TPC-C responds well to larger age limits but its write miss ratio, even at huge buffer pool sizes, is dramatically higher than that of the production workloads.

One of the drawbacks of letting pages exist in a modified state in the buffer pool is that when such a page is

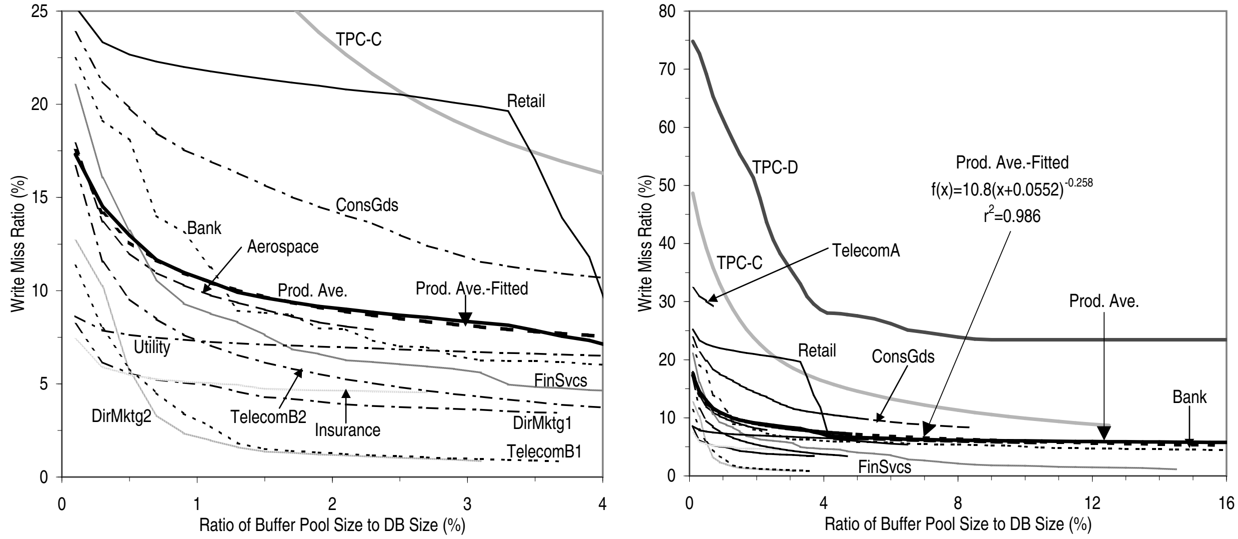



Figure 23: Write Miss Ratio as a Function of Ratio of Buffer Pool Size to Database Size (LRU Write Back Buffer Pool).

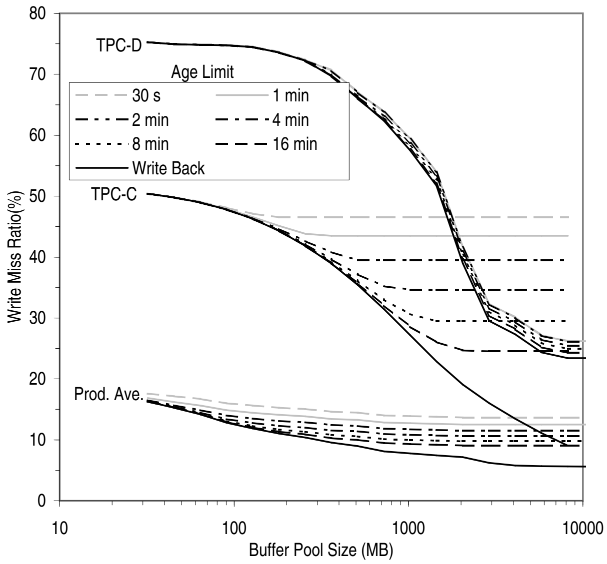



Figure 24: Effect on Write Miss Ratio of Imposing Age Limit on Dirty Pages.

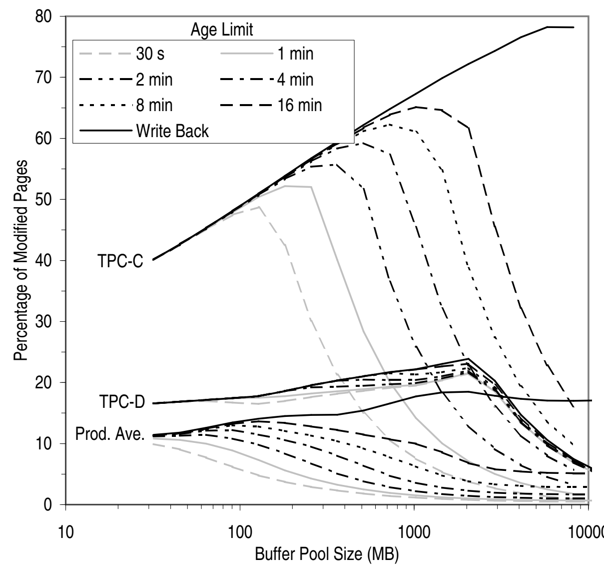



Figure 25: Percentage of Pages that are Dirty (Modified) in LRU Write Back Buffer Pool.

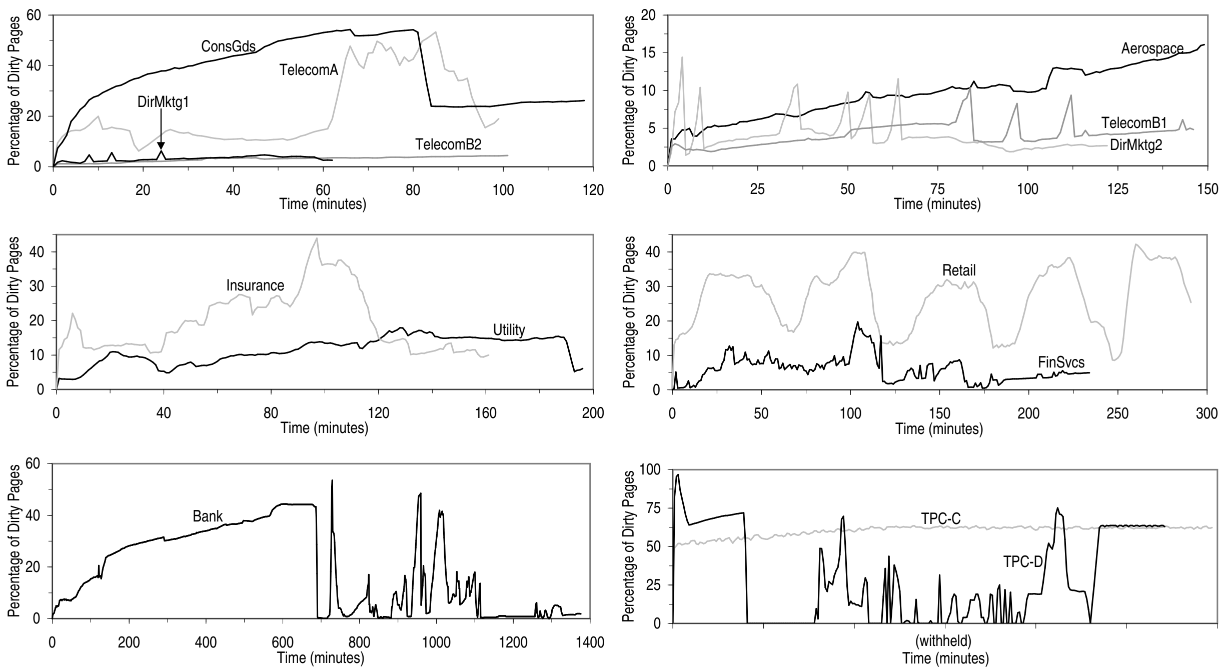



Figure 26: Profile over Time of Percentage of Dirty Pages in 512MB LRU Write Back Buffer Pool.

replaced, it has to be destaged before the buffer space can be reused. Therefore, we next investigate the percentage of pages in the buffer pool that are dirty. In Figure 25, we plot the average dirty percentage which is computed by observing the number of dirty pages in the buffer pool on every reference. For both the production workloads and TPC-D, only about 20% of the pages in the buffer pool are dirty. On the other hand, for TPC-C, more than 50% of the pages in the buffer pool can be modified. To better understand the effects of write buffering, it is useful to consider how the dirty percentage varies over time. The results for a 512MB LRU write back cache are presented in Figure 26. Notice that the dirty percentage for the production workloads and especially for TPC-D tend to be rather bursty. Therefore, allowing the write buffer to vary in size as needed is generally a good idea for these workloads. On the other hand, the dirty percentage for TPC-C is very constant, suggesting that a fixed-size dirty buffer will work well.

In order to establish a lower bound for the write miss ratio, we next modify the MIN [7] algorithm to consider only writes and to account for both the lifetime of dirty temporary pages and the age limit on dirty persistent pages. We refer to this new algorithm as *WMIN*. WMIN is a fixed-space algorithm that always destages the dirty page that will not be updated again before it is purged or before it has to be destaged. It should be apparent that WMIN is the optimal lookahead destage algorithm. In Figure 27, we compare the LRU write miss ratio with the WMIN write miss ratio. Note that in this figure, the write miss ratio is plotted as a function of the write buffer pool size and not the total buffer pool size, as is the case for the other plots. To obtain the LRU numbers for this figure, we simulated a buffer pool that only caches writes but that updated the LRU information on every hit, read or write.

Notice from the figure that as expected, the potential for improving the destage algorithm increases as the age limit is relaxed. With a write back cache, the performance difference between LRU and WMIN for the production workloads is only about 25% on average. This is somewhat lower than the corresponding number for the reference miss ratio and suggests that there is probably not a lot of room for improving the destage policy for these workloads. The same can be said for TPC-D. On the other hand, the difference in write miss ratio for TPC-C with relaxed age limits can be more than 50%. It might be possible to design a more elaborate destage policy for TPC-C but we believe that this would be of little value for the real workloads.

## 5 Conclusions

In this paper, we empirically examine the logical I/O reference behavior of the peak production database workloads from ten of the world's largest corporations. In particular, we analyze how these workloads respond to different techniques for caching, prefetching and write buffering by evaluating many previously published algorithms and techniques and several new ones that we developed. Since, it is extremely rare to have access to real production workloads, let alone such a large collection of them, we also establish several general rules of thumb with regards to the effectiveness

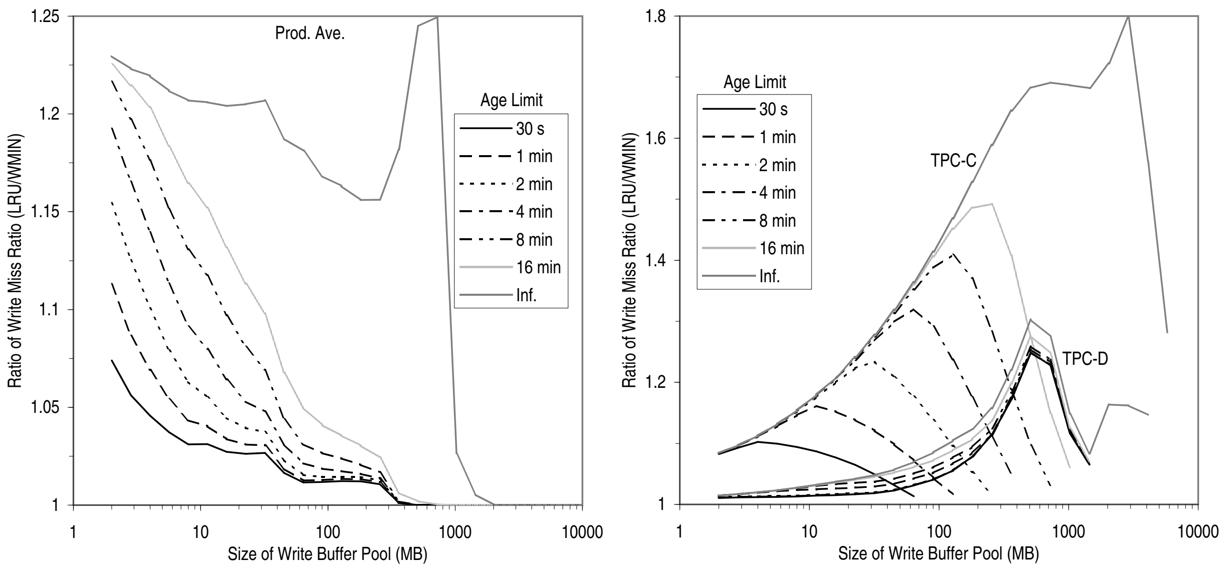



Figure 27: Ratio of LRU Write Miss Ratio to WMIN Write Miss Ratio.

of caching, prefetching and write buffering. For instance, we discover that the relationship between the buffer pool miss ratio and the ratio of buffer pool size to data size can be modeled by a function of the form $f(x) = a \cdot \frac{1}{\sqrt{x}}$, where $a$ is a constant. We refer to this rule of thumb as the square root rule. For the write miss ratio, we find that a fourth root rule is more appropriate.

We also analyze the reference characteristics of the de facto standard benchmarks for both on-line transaction processing and decision support systems, namely TPC-C and TPC-D. Comparing the reference behavior of these benchmarks with that of the production workloads, we find that, for the most part, the reference behavior of TPC-C and TPC-D fall within the spectrum of behavior exhibited by the production workloads. However, there are some noteworthy exceptions that have implications for well-known I/O optimization techniques such as caching, prefetching and write buffering. Although TPC-C and TPC-D are generally complementary in that they tend to be representative of different aspects of the production workloads, there remain some characteristics of the real workloads that are not reflected by either of the benchmarks.

More specifically, we find that the difference between the optimal lookahead and LRU miss ratios for TPC-C is significantly higher than the corresponding difference for the production workloads on average. In fact, the gap is much larger than the difference between the optimal realizable and optimal lookahead algorithms presented in [63]. This seems to suggest that for TPC-C, there may be considerable room for improving the page replacement algorithm beyond LRU. Expending effort in this direction, however, will primarily benefit TPC-C. Our analysis also clearly shows that there is a lot of interaction between the reference streams of the transactions in the production workloads. TPC-C is generally consistent in this regard but not TPC-D. This indicates that strategies that only consider per-transaction reference behavior will perform disproportionately well for TPC-D.

As has been suggested in previous studies such as [64], all the production workloads clearly exhibit significant amounts of sequentiality in their reference streams. In addition, we find that there is a lot of pseudo-sequentiality where the page references are generally increasing but may be interspersed by gaps and may occasionally back up. Our results indicate that a simple sequential prefetching scheme can reduce the average miss ratio of these workloads by more than half. The behavior of TPC-D in this regard is in-line with that of the production workloads. On the other hand, TPC-C has no significant sequentiality in its reference stream, meaning that it will not exercise one of the most effective techniques for improving the performance of production database workloads.

From a performance perspective, one of the most important characteristics of a workload is its write behavior. But it is in this aspect that differences between the production workloads and the TPC benchmarks are most apparent. We find that for almost all the production workloads, write buffering is an effective technique for reducing the number of physical writes to the storage system. However, for TPC-C and especially TPC-D, write buffering is far less effective. Furthermore, we find that for the production workloads on average, the percentage of buffer space that is modified tends to be small but this is not true for TPC-C. In addition, our analysis suggests that there may be more room to optimize the destage policy for TPC-C than for the other workloads.

The use of temporary or workfile space is another area where the TPC benchmarks do not track the behavior of the production workloads. Although temporary objects account for a significant portion of the write traffic in the production workloads, TPC-C has no such activity. While TPC-D has references to temporary objects, the behavior is different from that of the production workloads. More specifically, we find that the temporary space required by the production workloads tends to be small and to fit within the buffer pool but this is not the case for TPC-D. Deciding how to balance the demands for temporary buffer space with that for caching requires more work in TPC-D but this will tend not to matter in the production environments.

Finally, as we discussed in [26], the behavior of the production workloads tends to be dynamic while that of TPC-C is very static and regular. For example, we find that unlike the production workloads which have bursty demands for write buffer space, TPC-C's write buffer space requirement is practically constant. This indicates that TPC-C will tend not to reward dynamic or adaptive strategies that are useful in production environments. The fact that TPC-C's behavior is stationary also means that it is relatively easy and tempting to analyze TPC-C for the sole purpose of designing custom policies for achieving good benchmark numbers.

## 6 Acknowledgments

Hsu gratefully acknowledges the support of IBM during this research. This work was initiated as a joint project through an IBM Fellowship. Hsu is now supported by IBM Research. Smith's research in this and related areas is also funded by the State of California under the MICRO program, IBM, Cisco Corporation, Fujitsu Microelectronics, Intel Corporation, Microsoft Corporation, Quantum Corporation, Sun Microsystems and Toshiba Corporation.

Collecting traces and getting them into a form suitable for analysis constitute a very long and involved process. This research would not have been possible without the help of many. In particular, the authors would like to thank Eric Louie and Alex Hazlitt for their prompt answers and solutions to the many questions and problems that arose during the setting up of the TPC-C and TPC-D benchmarks. The authors would also like to thank Jyh-Herng Chow, Amit Somani and Surrendra Verma for helping the authors to navigate through the DBMS source code to determine the appropriate instrumentation points. In addition, the authors are much indebted to the many people who contributed to getting the production traces. Special mention is made of Ted Messinger who, even in retirement, provided the authors with help on processing and interpreting the production traces. Barbara Tockey and James Teng also provided invaluable help in this regard. The authors would also like to express their gratitude to Chuck Tribolet who offered to take on the arduous task of getting the raw production traces off more than 600 tape cartridges and onto the authors' computer system. Finally, the authors would like to recognize Joseph M. Hellerstein for his helpful comments on early versions of this paper.

## References

[1] A. V. Aho, P. J. Denning, and J. D. Ullman, "Principles of optimal page replacement," *Journal of the ACM*, vol. 18, no. 1, pp. 80–93, Jan. 1971.

[2] Anonymous, "Errata," *ACM Computing Surveys*, vol. 29, no. 4, pp. 428–428, Dec. 1997. See [75].

[3] C. Atul, H. J. Donald, A. Shibamiya, R. W. Lyle, and S. J. Watts, "System and method for avoiding complete index traversals in sequential and almost sequential index probes." U.S. Patent 5748952. Filed May 10, 1995. Issued May 5, 1998.

[4] M. J. Bach, *The Design of the UNIX Operating System*. Prentice Hall, 1986.

[5] M. G. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shirriff, and J. K. Ousterhout, "Measurements of a distributed file system," in *Proceedings of ACM Symposium on Operating Systems Principles*, (Pacific Grove, CA), pp. 198–212, Oct. 1991.

[6] S. J. Baylor and C. E. Wu, "Parallel I/O workload characteristics using Vesta," in *Input/Output in Parallel and Distributed Computer Systems* (R. Jain, J. Werth, and J. C. Browne, eds.), vol. 362 of *The Kluwer International Series in Engineering and Computer Science*, ch. 7, pp. 167–185, Kluwer Academic Publishers, 1996.

[7] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems Journal*, vol. 5, no. 2, pp. 78–101, 1966.

[8] S. D. Carson and S. Setia, "Analysis of the periodic update write policy for disk cache," *IEEE Transactions on Software Engineering*, vol. 18, no. 1, pp. 44–54, Jan. 1992.

[9] I. R. Casas and K. C. Sevcik, "A buffer management model for use in predicting overall database system performance," in *Proceedings of IEEE International Conference on Data Engineering.*, (Los Angeles, CA), pp. 463–469, Feb. 1989.

[10] C. M. Chen and N. Roussopoulos, "Adaptive database buffer allocation using query feedback," in *Proceedings of International Conference on Very Large Data Bases (VLDB)*, (Dublin, Ireland), pp. 342–353, Aug. 1993.

[11] H. T. Chou and D. J. DeWitt, "An evaluation of buffer management strategies for relational database systems," in *Proceedings of International Conference on Very Large Data Bases (VLDB)*, (Stockholm, Sweden), pp. 127–141, Aug. 1985.

[12] E. F. Codd, "A relational model for large shared data banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, June 1970.

[13] E. G. Coffman, Jr. and P. J. Denning, *Operating Systems Theory*. Prentice-Hall, Inc., 1973.

[14] D. W. Cornell and P. S. Yu, "Integration of buffer management and query optimization in relational database environment," in *Proceedings of International Conference on Very Large Data Bases (VLDB)*, (Amsterdam, The Netherlands), pp. 247–255, Aug. 1989.

[15] P. E. Crandall, R. A. Aydt, A. A. Chien, and D. A. Reed, "Input/output characteristics of scalable parallel applications," in *Proceedings of Supercomputing '95*, (San Diego, CA), Dec. 1995.

[16] A. Dan and D. Towsley, "An approximate analysis of the lru and fifo buffer replacement schemes," in *Proceedings of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, (Boulder, CO), pp. 143–52, May 1990.

[17] A. Dan, P. S. Yu, and J.-Y. Chung, "Database access characterization for buffer hit prediction," in *Proceedings of IEEE International Conference on Data Engineering*, (Los Alamitos, CA), pp. 134–144, Apr. 1993.

[18] W. Effelsberg and M. E. S. Loomis, "Logical, internal, and physical reference behavior in CODASYL database systems," *ACM Transactions on Database Systems*, vol. 9, no. 2, pp. 187–213, June 1984.

[19] C. Faloutsos, R. Ng, and T. Sellis, "Predictive load control for flexible buffer allocation," in *Proceedings of International Conference on Very Large Data Bases (VLDB)*, (Barcelona, Spain), pp. 265–274, Sept. 1991.

[20] R. A. Floyd and C. S. Ellis, "Directory reference patterns in hierarchical file systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 1, no. 2, pp. 238–247, June 1989.

[21] G. Graefe, "Query evaluation techniques for large databases," *ACM Computing Surveys*, vol. 25, no. 2, pp. 73–170, June 1993.

[22] L. Haas, W. Chang, G. Lohman, M. McPherson, P. Wilms, G. Lapis, B. Lindsay, H. Pirahesh, M. Carey, and E. Shekita, "Starburst mid-flight: As the dust clears," *IEEE Transactions on Knowledge and Data Engineering*, vol. 2, no. 1, pp. 143–160, Mar. 1990.

[23] P. Hawthorn and M. Stonebraker, "Performance analysis of a relational data base management system," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, (Boston, Massachusetts), pp. 1–12, May 1979.

[24] J. L. Hennessy and D. A. Patterson, *Computer Architecture A Quantitative Approach*. Morgan Kaufmann Publishers, Inc. San Francisco, CA, second ed., 1996.

[25] A. V. Hill, "The combinations of hæmoglobin with oxygen and carbon monoxide," *Biochemistry Journal*, vol. 7, pp. 471–480, 1913.

[26] W. W. Hsu, A. J. Smith, and H. C. Young, "Analysis of the characteristics of production database workloads and comparison with the TPC benchmarks." Technical Report CSD-99-1070, Computer Science Division, University of California, Berkeley, Nov. 1999. Also available as Research Report RJ 10165, IBM Almaden Research Center, San Jose, CA, Nov. 1999.

[27] W. W. Hsu, A. J. Smith, and H. C. Young, "Projecting the performance of decision support workloads on systems with smart storage (SmartSTOR)." Research Report RJ 10145, IBM Almaden Research Center, San Jose, CA, July 1999. Also available as Technical Report CSD-99-1057, Computer Science Division, University of California, Berkeley, CA, Aug. 1999.

[28] W. W. Hsu, A. J. Smith, and H. C. Young, "Results and data for 'Analysis of the I/O characteristics of production database workloads and the TPC benchmarks'," 1999. http://www.cs.berkeley.edu/~windsorh/DBChar.

[29] IBM Corporation, *DB2 for OS/390 V5 Installation Guide*. 1997.

[30] IBM Corporation, *DB2 UDB V5 Administration Guide*. 1997.

[31] Intel Corporation, "Intel Extended Server Memory Architecture (ESMA): Overcoming the 4 GB memory barrier," 1999. http://www.intel.com/procs/servers/pentiumiii/xeon/whitepapers/ESMA.htm.

[32] T. Johnson and D. Shasha, "2Q: a low overhead high performance buffer management replacement algorithm," in *Proceedings of International Conference on Very Large Data Bases (VLDB)*, (Santiago, Chile), pp. 439–450, 1994.

[33] J. P. Kearns and S. DeFazio, "Locality of reference in hierarchical database systems," *IEEE Transactions on Software Engineering (SE)*, vol. 19, no. 2, Mar. 1983.

[34] J. P. Kearns and S. DeFazio, "Diversity in database reference behavior," *Performance Evaluation Review*, vol. 17, no. 1, pp. 11–19, May 1989.

[35] W. F. King, "Analysis of paging algorithms," in *Proceedings of IFIP Congress*, (Ljubljana, Yugoslavia), pp. 485–490, Aug. 1971.

[36] D. E. Knuth, *The Art of Computer Programming: Sorting and Searching*, vol. 3. Addison-Wesley Publishing Company, second ed., 1998.

[37] S. T. Leutenegger and D. M. Dias, "A modeling study of the TPC-C benchmark," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, (Washington, D.C.), pp. 22–31, May 1993.

[38] B. McNutt, "A simple statistical model of cache reference locality, and its application to cache planning, measurement and control," *CMG (Computer Measurement Group) Transactions*, pp. 13–21, Winter 1993.

[39] B. McNutt, "MVS DASD survey: Results and trends," in *Proceedings of the CMG (Computer Measurement Group) Conference*, (Nashville, TN), pp. 658–667, Dec. 1995.

[40] J. Menon, "A performance comparison of RAID-5 and log-structured arrays," in *Proceedings of IEEE International Symposium on High Performance Distributed Computing*, (Washington, DC), pp. 167–178, Aug. 1995.

[41] E. L. Miller and R. H. Katz, "Input/output behavior of supercomputing applications," in *Proceedings Supercomputing '91*, (Albuquerque, NM), pp. 567–576, Nov. 1991.

[42] J. C. Mogul, "A better update policy," in *Proceedings of Summer 1994 USENIX Conference*, (Boston, MA), pp. 99–111, June 1994.

[43] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz, "ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging," *ACM Transactions on Database Systems*, vol. 17, no. 1, pp. 94–162, Mar. 1992.

[44] M. N. Nelson, B. B. Welch, and J. K. Ousterhout, "Caching in the SPRITE network file system," *ACM Transactions on Computer Systems*, vol. 6, no. 1, Feb. 1988.

[45] R. T. Ng, C. Faloutsos, and T. K. Sellis, "Flexible buffer allocation based on marginal gains," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, (Denver, Colorado), pp. 387–396, May 1991.

[46] V. F. Nicola, A. Dan, and D. M. Dias, "Analysis of the generalized clock buffer replacement scheme for database transaction processing," in *Proceedings of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, (Newport, RI), pp. 35–46, June 1992.

[47] N. Nieuwejaar, D. Kotz, A. Purakayastha, C. Schlatter Ellis, and M. Best, "File-access characteristics of parallel scientific workloads," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 10, pp. 1075–1089, Oct. 1996.

[48] E. J. O'Neil, P. E. O'Neil, and G. Weikum, "The LRU-K page replacement algorithm for database disk buffering," *SIGMOD Record (ACM Special Interest Group on Management of Data)*, vol. 22, no. 2, pp. 297–306, June 1993.

[49] J. Ousterhout, H. Da Costa, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson, "A trace-driven analysis of the UNIX 4.2 BSD file system," in *Proceedings ACM Symposium on Operating System Principles*, (Orcas Island, WA), pp. 15–24, Dec. 1985.

[50] B. K. Pasquale and G. C. Polyzos, "A case study of scientific application I/O behavior," in *Proceedings of the Second International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, (Durham, NC), pp. 101–106, Jan. 1994.

[51] B. K. Pasquale and G. C. Polyzos, "Dynamic I/O characterization of I/O intensive scientific applications," in *Proceedings of Supercomputing '94*, (Washington, DC), pp. 660–669, Nov. 1994.

[52] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1990.

[53] A. Purakayastha, C. Schlatter Ellis, D. Kotz, N. Nieuwejaar, and M. Best, "Characterizing parallel file-access patterns on a large-scale multiprocessor," in *Proceedings of IEEE International Parallel Processing Symposium*, (Santa Barbara, CA), pp. 165–172, Apr. 1995.

[54] N. Ragaz and J. Rodriguez-Rosell, "Empirical studies of storage management in a data base system." Research Report RJ 1834, IBM Research Laboratory, San Jose, CA, Oct. 1976.

[55] K. K. Ramakrishnan, P. Biswas, and R. Karedla, "Analysis of file I/O traces in commercial computing environments," in *Proceedings of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, (Newport, RI), pp. 78–90, June 1992.

[56] J. Robertson and M. Devarakonda, "Data cache management using frequency-based replacement," in *Proceedings of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, (Boulder, CO), pp. 134–142, May 1990.

[57] J. Rodriguez-Rosell, "Empirical data reference behavior in data base systems," *Computer Magazine of the Computer Group News of the IEEE Computer Group Society*, vol. 9, no. 11, Nov. 1976.

[58] M. Rosenblum and J. K. Ousterhout, "The design and implementation of a log-structured file system," in *Proceedings of ACM Symposium on Operating Systems Principles*, (Pacific Grove, CA), pp. 1–15, Oct. 1991.

[59] G. M. Sacco and M. Schkolnick, "A mechanism for managing the buffer pool in a relational database system using the hot set model," in *Proceedings of International Conference on Very Large Data Bases (VLDB)*, (Mexico City, Mexico), pp. 257–262, Sept. 1982.

[60] G. M. Sacco and M. Schkolnick, "Buffer management in relational database systems," *ACM Transactions on Database Systems*, vol. 11, no. 4, pp. 473–498, Dec. 1986.

[61] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, "Access path selection in a relational database management system," *Proceedings of ACM SIGMOD International Conference on Management of Data*, pp. 23–34, May 1979.

[62] V. Singhal and A. J. Smith, "Analysis of locking behavior in three real database systems," *The VLDB Journal*, vol. 6, no. 1, pp. 40–52, Jan. 1997. Extended version available as Technical Report CSD-94-801, Computer Science Division, University of California, Berkeley, CA, Apr. 1994.

[63] A. J. Smith, "Analysis of the optimal, look-ahead demand paging algorithms," *SIAM Journal on Computing*, vol. 5, no. 4, pp. 743–757, Dec. 1976.

[64] A. J. Smith, "Sequentiality and prefetching in database systems," *ACM Transactions on Database Systems*, vol. 3, no. 3, pp. 223–247, Sept. 1978. Also available as Research Report RJ 1743, IBM Research Laboratory, San Jose, CA, Mar. 1976.

[65] A. J. Smith, "Disk cache — miss ratio analysis and design considerations," *ACM Transactions on Computer Systems*, vol. 3, no. 3, pp. 161–203, Aug. 1985.

[66] A. J. Smith, "Trace driven simulation in research on computer architecture and operating systems," in *Proceedings of the Conference on New Directions in Simulation for Manufacturing and Communications*, (Tokyo, Japan), pp. 43–49, Aug. 1994.

[67] M. Stonebraker, "Operating system support for database management," *Communications of the ACM*, vol. 24, no. 7, pp. 412–418, July 1981.

[68] J. Z. Teng and R. A. Gumaer, "Managing IBM Database 2 buffers to maximize performance," *IBM Systems Journal*, vol. 23, no. 2, pp. 211–218, 1984.

[69] J. G. Thompson, *Efficient Analysis of Caching Systems*. PhD thesis, University of California, Berkeley, 1987. Available as Technical Report CSD 87/374, Computer Science Division, University of California, Berkeley, CA, Oct. 1987.

[70] Transaction Processing Performance Council, *TPC Benchmark$^{TM}$ C Standard Specification Revision 3.3.2*. June 1997.

[71] Transaction Processing Performance Council, *TPC Benchmark$^{TM}$ D Standard Specification Revision 1.3.1*. Dec. 1997.

[72] T.-F. Tsuei, A. N. Packer, and K.-T. Ko, "Database buffer size investigation for OLTP workloads," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, (Tucson, AZ), pp. 112–122, May 1997.

[73] W. G. Tuel, Jr., "An analysis of buffer paging in virtual storage systems," *IBM Journal of Research and Development*, vol. 20, no. 5, pp. 518–520, Sept. 1976. Also available as Research Report RJ 1421, IBM Research Laboratory, San Jose, CA, July 1974.

[74] W. G. Tuel, Jr. and J. Rodriguez-Rosell, "A methodology for the evaluation of data base systems." Research Report RJ 1668, IBM Research Laboratory, San Jose, CA, Oct. 1975.

[75] R. A. Uhlig and T. N. Mudge, "Trace-driven memory simulation: A survey," *ACM Computing Surveys*, vol. 29, no. 2, pp. 128–170, June 1997. See erratum [2].

[76] A. I. Verkamo, "Empirical results on locality in database referencing," in *Proceedings of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, (Austin, TX), pp. 49–58, 1985.

[77] N. Vishlitzky and Y. Ofek, "Sequential cache management system utilizing the establishment of a microcache and managing the contents of such according to a threshold comparison." U.S. Patent 5706467. Filed Sep 5, 1995. Issued Jan 6, 1998.

[78] B. B. Welch, "Measured performance of caching in the Sprite network file system," *Computing Systems*, vol. 4, no. 3, pp. 315–342, Summer 1991.

[79] J. Wilkes, R. A. Golding, C. Staelin, and T. Sullivan, "The HP AutoRAID hierarchical storage system," *ACM Transactions on Computer Systems*, vol. 14, no. 1, pp. 108–136, Feb. 1996.

[80] P. S. Yu and D. W. Cornell, "Buffer management based on return on consumption in a multi-query environment," *The VLDB Journal*, vol. 2, no. 1, pp. 1–38, Jan. 1993.

[81] S. Zhou, H. Da Costa, and A. J. Smith, "A file system tracing package for Berkeley UNIX," in *Proceedings of the 10th USENIX Conference*, (Portland, OR), pp. 407–419, June 1985.

[82] B. T. Zivkov and A. J. Smith, "Disk cache design and performance as evaluated in large timesharing and database systems," in *Proceedings of the CMG (Computer

*Measurement Group) Conference*, (Orlando, FL), pp. 639–658, Dec. 1997. Abridged version published as "Disk Caching in Large Database and Timeshared Systems", Proceedings of the Fifth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, (Haifa, Israel), pp. 184-195, Jan. 1997. Extended version available as Technical Report CSD-96-913, Computer Science Division, University of California, Berkeley, CA, Sep. 1996.

## Appendix

## A1 Related Work (Expanded)

### A1.1 Hierarchical and Network DBMS

There have been several published studies of the reference behavior of database workloads. However, most of these studies were conducted on hierarchical and network databases. In addition, the studies are rather limited in scope, often relying on data collected at only one or two installations. In several cases, the database was real but was driven by small contrived programs. These studies report conflicting results as to whether locality or sequentiality is present in the database reference stream. Clearly, the characteristics of the reference stream depend on the workload imposed on the database and the various studies should be interpreted with this in mind.

Some of the earliest empirical studies of database reference behavior are based on an IMS database used in an on-line manufacturing control system [54, 57, 64, 74, 73]. The system was observed to have strong sequentiality and weak locality. The run length was found to be a useful predictor of future access patterns [57]. A prefetch algorithm that selectively prefetches pages by using the run length distribution and conditioning on the current run length is proposed in [64]. In addition, a method for estimating the optimal prefetch amount is presented. In [33], a large commercial IMS database was faithfully reproduced and was observed while it was driven by two small non-production programs in batch mode. These two programs were supposed to be representative of the types of batch programs used to maintain a large database. The reference stream was found to have strong locality but no significant sequentiality [34].

A subsequent study of a small CODASYL database running retrieval transactions found moderate levels of locality but no sequentiality [18]. In this study, the database contained live data for a blood center and for a bibliographic system. A typical day-time query stream was analyzed. In [76], two batch non-production programs were run against a real CODASYL-like database. The first program listed a selected part of the database in a given order while the second program updated, inserted and deleted records from the database. The system was observed to exhibit weak locality and no significant sequentiality. More recently, a page reference trace taken from a production CODASYL OLTP system of a large bank was used to evaluate new page replacement algorithms [32, 48]. The results suggest that the reference stream does exhibit locality.

Based on IMS database references from two commercial installations, [34] discovered that the per transaction and per database reference behavior varied widely across the different transactions and databases but was stable over time. In [9], five hierarchic databases were observed in a production setting over a five day period. The block reference behavior was found to closely approximate a Bradford-Zipf (B-Z) distribution. [82] investigated design issues in disk caches using data from several commercial installations, including both IMS and DB2 customer sites. The reference streams clearly exhibited locality of reference and sequentiality was found to be prevalent in most of them.

### A1.2 Relational DBMS

There has been little published data on the empirical reference behavior of relational databases. In an early work based on INGRES, [23] looked at the reference behavior of several different types of queries using a course/room scheduling database and an accounting database. Using both user query streams and query streams assembled from user queries, [23] found localized reference behavior to the temporary relations in queries with aggregate functions. In addition, some short queries were found to frequently rereference system tables. Strong sequentiality was exhibited by queries that reference large number of records. For queries that involve multiple relations, pages were found to be cyclically re-referenced. More recently, [17] used reference traces taken from two commercial installations of DB2 to characterize the access skew of different data objects so as to predict buffer hit ratio. In [32], a DB2 trace from a commercial installation was used to evaluate the 2Q page replacement algorithm. Strong locality of reference was indicated in both these studies. Finally, as mentioned above, DB2 reference traces were used in [82] to study disk caching.

For the most part, research on buffering in relational DBMS has focused on using semantic information derived from the query plan optimizer to direct buffer management. In relational DBMS, users state their processing requirements using high level language interfaces, leaving the DBMS to select the best strategy or query plan for accessing the data [12, 61]. Since the pattern of data references can be predicted from the query plan, [67] suggests that a replacement algorithm specific to relational DBMS should outperform the LRU replacement policy. This approach of using the query plan to direct buffer management is developed in [11, 59, 60]. The issue of buffer allocation among multiple concurrent queries is subsequently addressed in [19, 45, 80]. [14] suggests that query optimization and buffer management should be integrated because the optimal query plan is a function of the available buffer size. Instead of relying on query plan information, a profiling approach that uses prior executions of a query to characterize its access patterns has also been proposed [10].

In general, the approaches that rely on the query plan work only for specific patterns such as sequential and cyclic sequential. These approaches consider all other references as simply random. Another problem with these methods is that the query plan is based on estimates such as row cardinality, predicate selectivity and clustering factor and these may not be accurate. Furthermore, for complex queries, the accurate prediction of reference patterns from the query plan is difficult. To make matters worse, in multi-user situations, the query plans can overlap in complicated ways and this overlap is not accounted for by the query plan directed algorithms. In reality, these algorithms are best used together with techniques based on run time access characteristics as in [22, 68].

### A1.3 TPC Benchmarks

Although he TPC-C [70] and TPC-D [71] benchmarks have become the de facto standard benchmarks for on-line transaction processing and decision support systems respectively, there has not been any major effort to empirically examine and understand their I/O characteristics. Based on static analysis of accesses to tables, [37] looked at the skewness in the data access of TPC-C. [72] contains an empirical study of how the database size, buffer size and the number of CPUs affect the throughput and buffer hit rate of TPC-C on symmetric multiprocessors (SMPs). Recently, [27] presents the query plans taken from a recently certified TPC-D setup and considers the potential benefit of offloading TPC-D operations to storage systems with embedded processors.

### A1.4 Filesystem Reference Behavior

[49, 69, 81] studied file reference characteristics on time-shared VAX-11/780s in an academic environment. The measurements show that the accesses tend to be bursty and highly sequential. A major finding is that most file data are deleted or overwritten within a few minutes of creation. In addition, caches of several megabytes can eliminate a large proportion of all disk traffic. Similar file usage patterns are reported in a subsequent study conducted on a collection of about 40 10-MIPS workstations running the Sprite operating system in a comparable academic environment [5]. [78] reports measurements on the performance of caching in the Sprite network filesystem. The patterns of reference to the UNIX filesystem directory is studied in [20]. The results show that the directory references exhibit strong locality and that caches are an effective way to decrease directory overhead. An analysis of the file usage patterns in commercial computing environments is presented in [55]. Unlike most other studies that are based on data from academic or research environments, this study was based on traces collected at eight different and relatively large VAX/VMS customer sites.

### A1.5 Parallel and Supercomputing I/O

There has been a lot of recent work on characterizing the I/O behavior of scientific applications in parallel and supercomputing environments. See for instance [6, 15, 41, 47, 50, 51, 53]. In general, the I/O reference patterns of these applications are more regular and predictable than those of commercial database workloads. This reflects the fact that these huge scientific applications tend to be highly structured and are designed to use carefully formatted data sets. More specifically, in scientific vector applications, the files tend to be large and are usually accessed completely and sequentially. For scientific parallel applications, the requests are smaller and tend to be non-consecutive but still generally sequential.

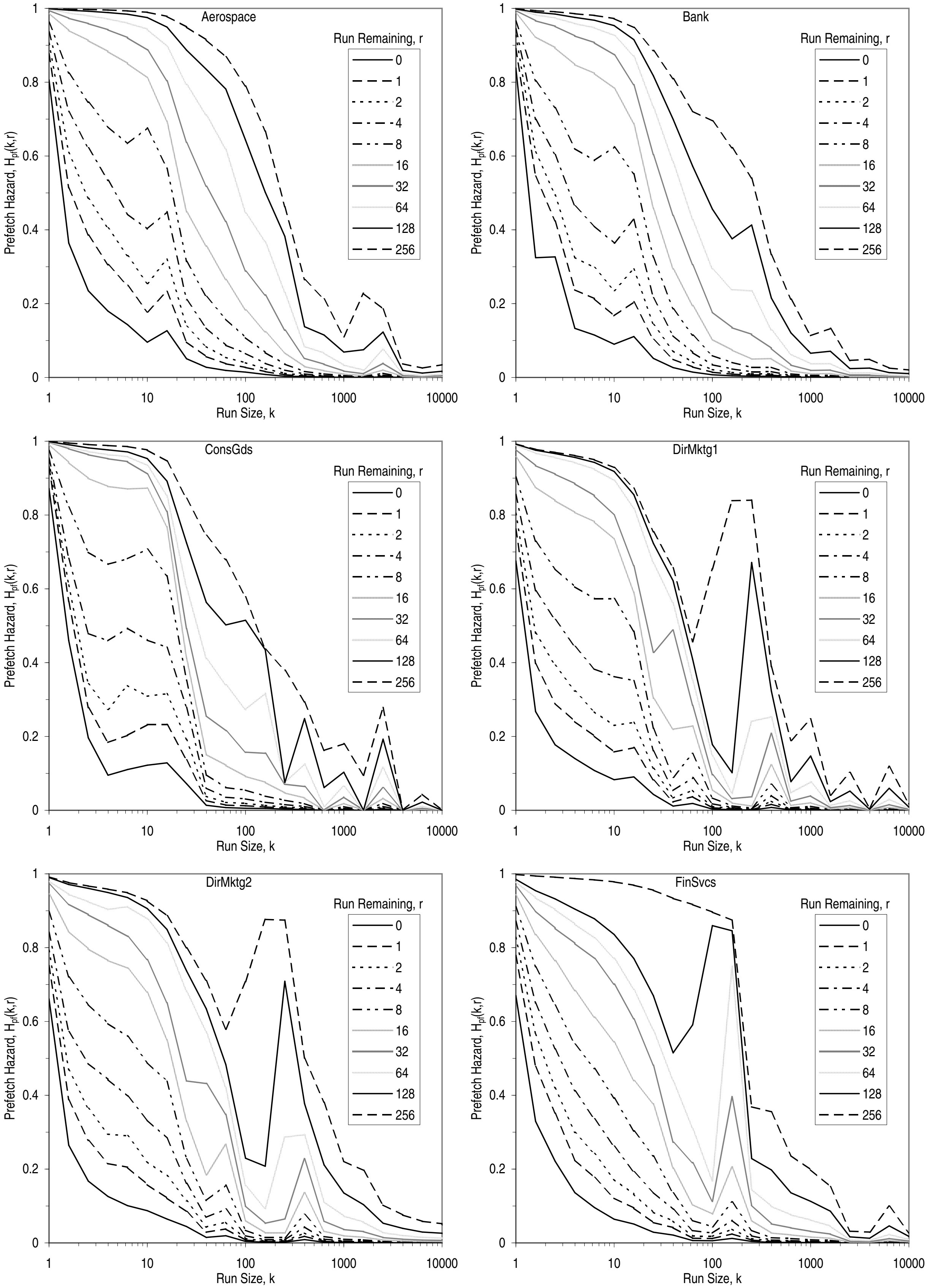



Figure A-1.i: Prefetch Hazard.

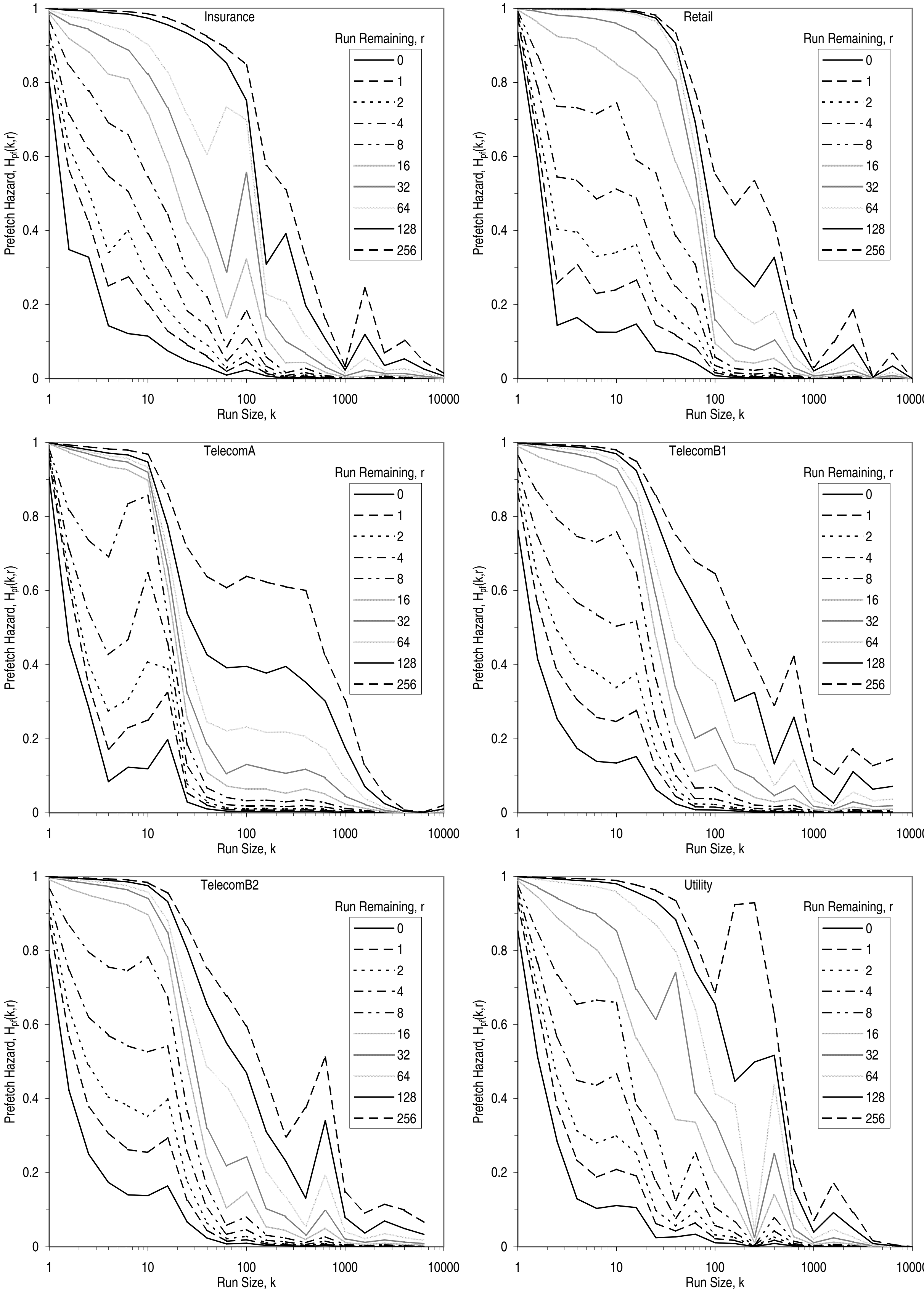



Figure A-1.ii: Prefetch Hazard.

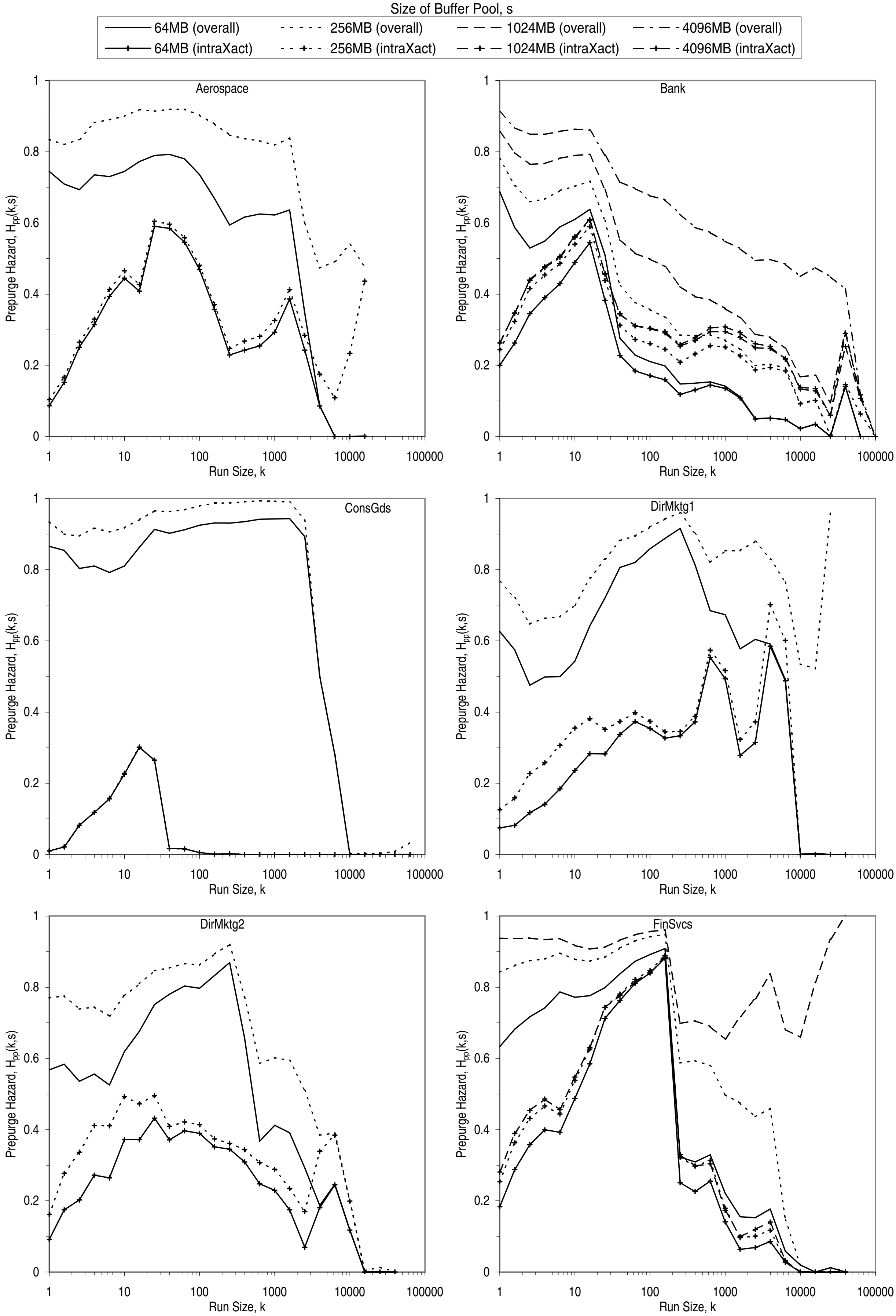





Figure A-2.i: Prepurge Hazard.

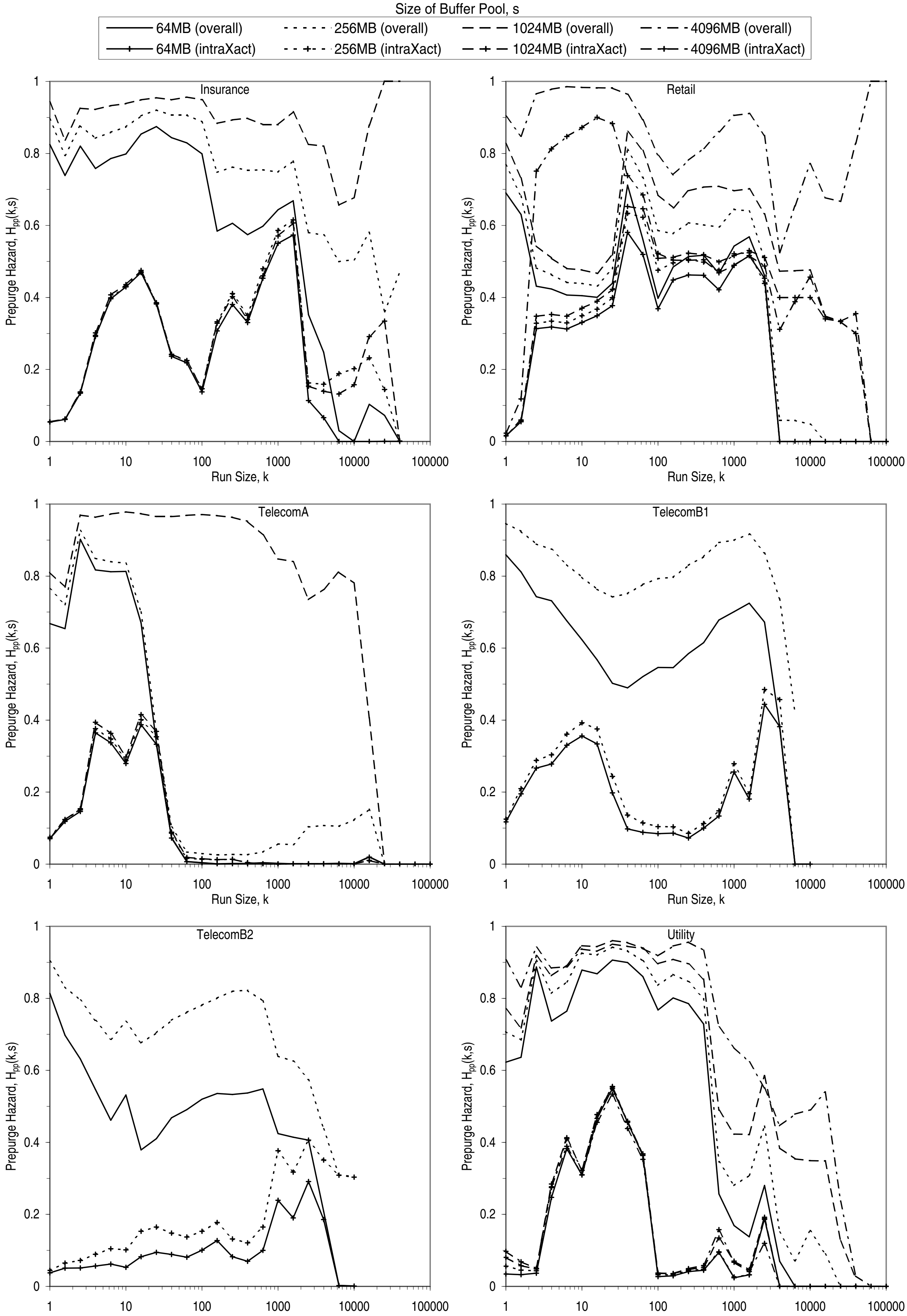





Figure A-2.ii: Prepurge Hazard.