Appeared in the *Proceedings of the 25th International Symposium on Computer Architecture*, June 1998.

Memory System Characterization of Commercial Workloads

Luiz André Barroso, **Kourosh Gharachorloo**, and **Edouard Bugnion***
Western Research Laboratory
Digital Equipment Corporation
{barroso,kourosh}@pa.dec.com
bugnion@cs.stanford.edu

Abstract

Commercial applications such as databases and Web servers constitute the largest and fastest-growing segment of the market for multiprocessor servers. Ongoing innovations in disk subsystems, along with the ever increasing gap between processor and memory speeds, have elevated memory system design as the critical performance factor for such workloads. However, most current server designs have been optimized to perform well on scientific and engineering workloads, potentially leading to design decisions that are non-ideal for commercial applications. The above problem is exacerbated by the lack of information on the performance requirements of commercial workloads, the lack of available applications for widespread study, and the fact that most representative applications are too large and complex to serve as suitable benchmarks for evaluating trade-offs in the design of processors and servers.

This paper presents a detailed performance study of three important classes of commercial workloads: online transaction processing (OLTP), decision support systems (DSS), and Web index search. We use the Oracle commercial database engine for our OLTP and DSS workloads, and the AltaVista search engine for our Web index search workload. This study characterizes the memory system behavior of these workloads through a large number of architectural experiments on Alpha multiprocessors augmented with full system simulations to determine the impact of architectural trends. We also identify a set of simplifications that make these workloads more amenable to monitoring and simulation without affecting representative memory system behavior. We observe that systems optimized for OLTP versus DSS and index search workloads may lead to diverging designs, specifically in the size and speed requirements for off-chip caches.

1 Introduction

During the past few years, shared-memory multiprocessor systems based on commodity microprocessors have emerged as the platform of choice for running demanding commercial database and Web workloads. Even though these systems have traditionally targeted engineering and numeric applications, the cost effectiveness of using commodity components has enabled them to successfully compete in a market which was once exclusively the domain of mainframes. In fact, with the increasing demand for database applications and the emergence of the Web, commercial workloads have quickly surpassed scientific/engineering workloads to become the largest market segment for multiprocessor servers.

The dramatic change in the target market for shared-memory servers has yet to be fully reflected in the design of these systems. The behavior of commercial workloads is known to be very different from numeric workloads [12]. Yet, current processors have been primarily optimized to perform well on the SPEC95 [18] benchmark suite. Similarly, design of multiprocessor architectures, along with academic research in this area, have been heavily influenced by popular scientific and engineering benchmarks such as SPLASH-2 [26] and STREAMS [13], with only a handful of published architectural studies that have in some way tried to address issues specific to commercial workloads [3, 7, 9, 12, 14, 16, 20, 21, 24].

The lack of architectural research on commercial applications is partly due to the fact that I/O issues have been historically considered as the primary performance bottleneck for such workloads. However, innovations in disk subsystems (RAID arrays, use of non-volatile memory) and software improvements that exploit the intrinsic parallelism of these workloads to hide I/O latencies have largely alleviated problems in this area. At the same time, the increasing gap between processor and memory speeds, and the higher memory latencies typical of multiprocessor systems, have made memory system design the critical performance factor for commercial workloads.

Even though the architecture community has recently developed a sense of urgency to pay more attention to commercial applications, a combination of factors complicate their study:

Lack of Availability and Restrictions. The biggest difficulty in studying commercial database engines is the lack of easy access due to the proprietary nature of the software. Software licenses typically prevent the disclosure and publication of performance information, and almost never include any access to the source code.

Large Scale. Commercial applications are often difficult and expensive to set up due to their large scale. For example, just the hardware cost of systems assembled for generating audit-quality performance results on database benchmarks is often several hundred thousand (and sometimes a few million) dollars.

Complexity. Commercial workloads are inherently complex, and often exhibit a large amount of non-trivial interaction with

*Edouard Bugnion is a graduate student in computer science at Stanford University.

the operating system and the I/O subsystem. The difficulty in the understanding and interpretation of the results is exacerbated by the lack of source code. These interactions also make such workloads an extremely challenging target for simulation studies.

Moving Target. Due to the competitive nature in some important commercial markets such as databases, software changes occur at a very fast pace, often providing large performance improvements with each new generation. For example, commercial database engines have improved tremendously during the past few years in scaling to more processors in shared-memory systems. Therefore, studies that are a few years old (e.g. [16]) or that are based on software that is not of commercial grade (e.g., the public domain Postgres database [21]) are likely to be outdated and non-representative. The above issues are even more pronounced for Web workloads since numerous applications in this area are yet to be developed, and the existing ones are only in their infancy.

The goal of this paper is to provide a detailed characterization of the memory system behavior of three important commercial workloads running on a shared-memory multiprocessor: (i) online transaction processing (OLTP), (ii) decision support systems (DSS), and (iii) Web index search. We have been fortunate to have access to the Oracle database engine [19] for running our database workloads, and the popular AltaVista search engine for our Web workload.

Our characterization results consist of two classes of experiments. The first set of results consists of a large number of monitoring experiments on Alpha multiprocessor platforms, using a wide range of hardware and software monitoring tools. The second set of results uses full system simulation (using our Alpha port of SimOS [15]) to study the effect of architectural variations. In dealing with the large scale and complexity of these workloads, we have identified a number of simplifications that make these workloads more amenable to monitoring and simulation studies (both full system and user-level only) without affecting their intrinsic memory behavior. Determining and validating the appropriate set of simplifications is a non-trivial task. Our techniques result from several months of studying Oracle, and we benefited from the expertise of the designer of the AltaVista index search [2].

One of the most interesting results of this paper is with respect to OLTP. Although OLTP is known to be memory intensive, emerging architectural trends will allow off-chip caches to capture significant working sets and eliminate most capacity and conflict misses, leaving only communication misses. The use of larger-scale shared-memory multiprocessors for running commercial workloads indicate that dirty cache misses will soon outnumber misses to memory, effectively forcing systems architects to reconsider trade-offs that penalize dirty misses. The same trade-off is not necessarily exposed by scientific applications which in general exhibit much lower communication miss rates, and for which communication data often gets replaced back to memory before being read by another processor. Our results also show that DSS and AltaVista behave quite differently from OLTP. Despite the size of their largest data structures, the critical working sets of these applications fit in reasonably sized on-chip caches.

The rest of paper is structured as follows. The next section provides a detailed description of the commercial workloads used in our study. Section 3 presents our experimental methodology and setup. Sections 4, 5, and 6 present our main results: our monitoring experiments, our simulation experiments, and the discussion of our scaling methodology. Finally, we discuss related work and conclude.

2 Background on Commercial Workloads

This section provides a detailed description of our two database workloads and our Web index search workload.

2.1 Database Workloads

The most important classes of applications in the database market are online transaction processing (OLTP) and decision support systems (DSS). Online transaction processing systems are used in day-to-day business operations (e.g., airline reservations), and are characterized by a large number of clients who continually access and update small portions of the database through short running transactions. On the other hand, decision support systems are primarily used for business analysis purposes, whereby information from the OLTP side of a business is periodically fed into the DSS database and analyzed. In contrast to OLTP, DSS is characterized by long running queries that are primarily read-only and may each span a large fraction of the database.

2.1.1 Oracle Database Engine

Our OLTP and DSS workloads run on top of the Oracle 7.3.2 DBMS. The Oracle 7.3.2 DBMS runs on both uniprocessors and multiprocessor shared memory machines, and recent benchmark results demonstrate that the software scales well on current SMP systems [4]. Figure 1 illustrates the different components of Oracle. The server executes as a collection of Unix processes that share a common large shared memory segment, called the *System Global Area*, or SGA. Oracle has two types of processes, *daemons* and *servers*. The daemons run in the background and handle miscellaneous housekeeping tasks such as checking for failures and deadlocks, evicting dirty database blocks to disk, and writing redo logs. The server processes are the ones that actually execute database transactions and account for most of the processing. Both daemon and server processes share the same code, but have their own private data segment called the *Program Global Area*, or PGA. Daemons and servers primarily use the SGA for communication and synchronization, but also use signals to wake up blocked processes.

The SGA is roughly composed of two regions, namely the *block buffer* and *meta-data* areas. The block buffer area is used for caching the most recently used database disk blocks in main memory, and typically accounts for 80% of the SGA size. The meta-data area contains the directory information that is used to access the block buffers, in addition to space for miscellaneous buffers and synchronization data structures.

Database engines maintain two important types of persistent data structures: the database tables and the redo log. The latter keeps a compressed log of committed transactions, and is used to restore the state of the database in case of failure. Committing only the log to disk (instead of the actual data) allows for faster

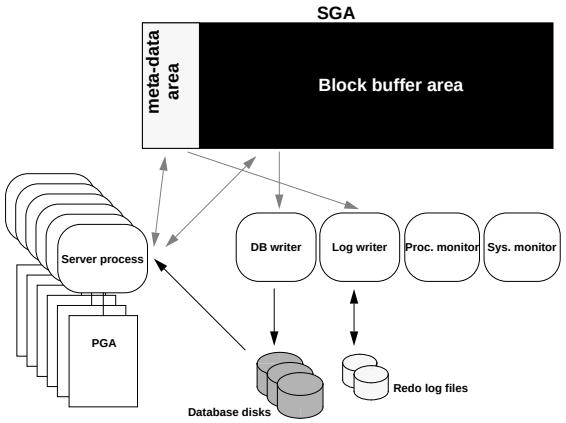

Figure 1: Major components of the Oracle 7.3.2 server.

transaction commits and for a more efficient use of disk bandwidth. Oracle 7.3.2 has a single daemon process, the *log writer* that groups commit logs from independent transactions into a single disk write for more efficient use of the disk bandwidth.

2.1.2 OLTP Workload

Our OLTP workload is modeled after the TPC-B benchmark [22]. This benchmark models a banking system, with each transaction simulating a balance update to a randomly chosen account. The account balance update involves updates to four tables: the branch table, the teller table, the account table itself, and the history table. Each transaction has fairly small computational needs, but the workload is I/O intensive because of the four table updates per transaction. Fortunately, even in a full-size TPC-B database (e.g., a few hundred GBs), the history, branch, and teller tables can be cached in physical memory, and frequent disk accesses are limited to the account table and the redo log.

We use Oracle in a dedicated mode for this workload, whereby each client process has a dedicated server process for serving its transactions. The communication between a client and its server process occurs through a Unix pipe. Each transaction consists of five queries followed by a commit request. To guarantee the durability of a committed transaction, commit requests block the server until the redo information has been written to the log. To hide I/O latencies, including the latency of log writes, OLTP runs are usually configured with multiple server processes (e.g. 7-8) per processor.

Even though TPC-C has superseded TPC-B as the official OLTP benchmark, we have chosen to model our OLTP workload on TPC-B because it is simpler to set up and run, and yet exhibits similar behavior to TPC-C as far as processor and memory system are concerned. We have verified this similarity by comparing our memory system performance results presented in Section 4 with results from audit-size TPC-C runs [4] on the same hardware platform (AlphaServer 4100). We have also confirmed this similarity on other Alpha multiprocessor platforms.

We provide a few general statistics, gathered with the DCPI [1] profiling tool, on the high-level behavior of the workload. On a four processor AlphaServer, the workload spends 71% in user-mode, 18% in the kernel and 11% in the idle loop. As expected from a tuned system, the workload hides most of the I/O latency. A more detailed breakdown shows that 85% of the user time is spent in the servers, 15% in the clients and that the computation requirement of the daemons is negligible. Another interesting behavior in OLTP is the high frequency of context switches. Server processes block between each query, on average every 25K instructions. Client processes run an even smaller number of instructions between blocking system calls. This leads to a fine interleaving between multiple server and client processes that share the same processor.

```
SELECT SUM(L_EXTENDEDPRICE * L_DISCOUNT) AS REVENUE
FROM LINEITEM
WHERE
L_SHIPDATE >= DATE '[date]' AND
L_SHIPDATE < DATE '[date]' + INTERVAL '1' YEAR
L_DISCOUNT BETWEEN [discount] - 0.01 AND [discount] + 0.01 AND
L_QUANTITY < [quantity];
```

Figure 2: SQL code for query Q6 in TPC-D.

2.1.3 DSS Workload

Our DSS workload is modeled after the TPC-D benchmark [23]. The benchmark simulates the decision support system for a wholesale supplier that distributes and sells products worldwide. The database size is specified by a scale factor (SF), with a scale factor of 1 corresponding to a raw data size of 1 GB. Typical audited TPC-D databases sizes are 100-1000 GBs. The following are the eight tables in the database (with the number of rows shown for SF=1): lineitem (6000K rows), order (1500K rows), partsupp (800K rows), part (200K rows), customer (150K rows), supplier (10K rows), nation (25 rows), and region (5 rows). These tables provide the information on all parts that are bought from suppliers and sold to customers. Each customer order is reflected in the order table, along with the ordered parts being reflected in the lineitem table. The lineitem table is by far the biggest table, and constitutes about 75% of the total raw data that is used to build the database. The benchmark also allows various indices to be created for fields in the data tables to help queries that can benefit from index scans.

The TPC-D benchmark consists of 17 read-only queries (Q1 to Q17) and two update queries (UF1 and UF2). The read-only queries vary in complexity and running time, with some queries accessing only one or two tables and other queries accessing all the main data tables. These queries are used to answer critical business questions; e.g., Q1 provides a summary pricing report of all lineitems shipped as of a given date. The two update queries model infrequent modifications to the database to keep it up-to-date.

The focus of this paper is on the read-only queries, since these queries are by far the most important component in the overall performance of TPC-D. The read-only queries are expressed in the Structured Query Language (SQL), a non-procedural language commonly used in relational databases. For illustration purposes, Figure 2 shows the SQL code for one of the simplest queries, “Q6”. This query selects (from the lineitem table) all goods ordered in a given period with a given discount and maximum quantity, and computes the total sum of discounts over these goods.

Before executing an SQL query, the database engine first formulates an optimized execution plan built out of a combination of basic table manipulation operations: selects, joins, sorts, and

	Tables Used	Selects	Joins	Sorts/ Aggr.	User	Kernel	Idle
Q1	1	1 FS	-	1	94%	2.5%	3.5%
Q4	2	1 FS, 1 I	-	1	86%	4.0%	10%
Q5	6	6 FS	5 HJ	2	84%	4.0%	12%
Q6	1	1 FS	-	1	89%	2.5%	8.5%
Q8	7	8 FS	7 HJ	1	82%	5.0%	13%
Q13	2	1 FS, 1 I	1 NL	1	87%	4.0%	9.0%

Table 1: Representative DSS queries. FS: full table scan, I: index scan, HJ: hash join, NL: nested loop join. Aggr.: Aggregates.

aggregates. *Selects* operate on a single table and may use an index or a hash table. A *join* takes two tables as input and produces one result table by merging rows that match a given criteria. Joins fall into three categories: merge (sort) which sorts both tables and does the join; nested loop which for every row in one table, finds rows to join from the second table (ideally using an index scan); and hash join which creates a hash on the larger table and scans the other table for the join. Finally, database engines support a range of *sort* and *aggregate* operations on rows of a given table.

The choice of the optimized execution plan is critical to the performance of the query. *Intra-query parallelism*, the parallelization of a query between a given number of server processes, is required to hide the latency of disk accesses and to take advantage of the additional processors present on a multiprocessor. With Oracle, the degree of parallelization is configurable and is typically set to 6-7 servers per processor. Referring back to the example in Figure 2, Q6 can be parallelized by parceling the full table scan among the server processes, and performing a final reduction to determine the aggregate sum.

We have chosen to study 6 queries, shown in Table 1, that represent different access patterns and exhibit different levels of complexity, parallelism, and parallel execution efficiency. The table also shows the number and type of basic operations chosen by the Oracle optimizer for the parallelized version of the queries in our study. In addition, we show the breakdown between user, kernel, and idle time on the four processor AlphaServer.

In contrast to OLTP, the kernel component of DSS workloads is negligible. Another key difference with OLTP workloads is that blocking system calls (typically disk read accesses in DSS) occur rather infrequently; e.g., in Q6, an average of 1.7M instructions are executed by a server between blocking calls.

2.2 Web Index Search Workload

The AltaVista index search is a state-of-the-art indexing program used to run the popular AltaVista Internet Search service. We chose to study AltaVista for two main reasons: (i) it represents an important use of high performance multiprocessors for Web applications, and (ii) it fits especially well in our study because it is similar to a DSS workload without depending on a commercial database engine.

AltaVista is a multi-threaded application that matches queries against a large search index database (currently over 200 GBs). The entire database is mapped into the address space of the threads. Unlike a database engine, which makes its own block replacement decisions, AltaVista relies on the operating system to implement an adequate page replacement policy. AltaVista runs multiple threads per processor to hide the latency of page faults. Since the main data structure, i.e., the index data, is read-only, there is virtually no synchronization necessary between the multiple threads while serving queries. Both the main search loop in each thread and the data layout have been carefully optimized to the point of scheduling individual instructions for maximum issue bandwidth and on-chip cache performance.

Before searching the actual database, the thread first checks the query against a cache of recently issued queries, allowing it to return the result of some queries immediately. During the search it is likely that the thread will give up its processor before reaching its quantum due to page faults in the memory mapped index files. Effective caching of index pages in main memory, high hit rates in the results cache, and a sufficient number of threads to serve queries lead to an extremely efficient system with virtually no idle time.

3 Methodology

This section describes the hardware platform and tools used for our monitoring studies, our simulation platform, and the workload parameters used in this study.

3.1 Hardware Platform

Our monitoring experiments were mostly performed on an AlphaServer 4100 with four 300 MHz 21164 processors and 2 GB of physical memory. Each processor has 8KB direct-mapped on-chip instruction and write-through data caches, a 96KB on-chip combined 3-way set associative second-level cache (Scache), and a 2MB direct-mapped board-level cache (Bcache). The block size is 32 bytes at the first level caches and 64 bytes at lower levels of the cache hierarchy. The individual processors are rated at 8.1 SPEC95int and 12.7 SPEC95fp respectively, and the system bus has a bandwidth of 1 GB/s. The dependent load latencies measured on the AlphaServer 4100 are 7 cycles for a Scache hit, 21 cycles for a Bcache hit, 80 cycles for a miss to memory, and 125 cycles for a dirty miss (cache-to-cache transfer). The latencies are quite good especially under a light load (approximately 260 nsec for a miss to memory). We have also used the 8 processor AlphaServer 8400 for a few of our database studies, and for doing the characterization of the AltaVista search engine. Both servers were configured with high-performance StorageWorks HSZ disk array subsystems.

3.2 Monitoring Tools

The complexity of commercial workloads, combined with the lack of access to source code (e.g., in the case of Oracle), make them particularly challenging to use as benchmarks in an architectural study. We find that monitoring the execution of such workloads on existing platforms is absolutely critical for developing a deeper understanding of the application behavior, and for subsequently sizing and tuning the workload parameters to achieve representative and efficient runs.

Although developing an understanding of the applications and tuning them was by far the most time consuming part of our study, we were helped greatly by the wide variety of monitoring and profiling tools that are available on the Alpha platform. To begin with, the Alpha 21164 processor provides a rich set of event counters that can be used to construct a detailed view of

processor behavior, including all activity within the three-level cache hierarchy [3]. We used the IPROBE monitoring tool to gain access to the processor event counters. A typical monitoring experiment involved multiple runs of the workload with IPROBE, measuring a single event in each run.[1] Event types available include counts of accesses and misses in the various caches, TLB misses, types of instructions, branch mispredicts, issue widths, memory barriers, replay traps, etc.

DCPI [1] (Digital Continuous Profiling Infrastructure) is an extremely low overhead sampling-based profiling system that is also based on the processor event counters, and is especially useful because of its ability to associate event frequencies with specific executable images or processes. ATOM [17] is a static binary translator that facilitates the instrumentation of an executable image. We used ATOM to instrument the Oracle binary to identify different memory regions accessed by the servers, to study the different phases of OLTP transactions, to characterize the system call and synchronization behavior of the workloads, and finally to generate user-level traces for various simple footprint and cache studies.

3.3 Simulation Environment

We augment our monitoring experiments by using simulations to primarily study the effect of architectural variations on our workloads. For this purpose, we developed an Alpha port of the SimOS simulation environment. SimOS [15] is a full system simulation environment originally developed at Stanford University to study MIPS-based multiprocessors. Our version of SimOS simulates the hardware components of Alpha-based multiprocessors (processors, MMU, caches, disks, console) in enough detail to run Digital's system software. Specifically, SimOS-Alpha models the micro-architecture of the Alpha 21164 processor [5] and runs essentially unmodified versions of Digital Unix 4.0 and the 21164 PALcode.

SimOS-Alpha supports multiple levels of simulation detail, enabling the user to choose the most appropriate trade-off between simulation detail and slowdown. Its fastest simulator uses an on-the-fly binary translation technique similar to Embra [25] to position the workload into a steady state. The ability to simulate both user and system code under SimOS is essential given the rich level of system interactions exhibited by commercial workloads. In addition, setting up the workload under SimOS is particularly simple since it uses the same disk partitions, databases, application binaries, and scripts that are used on our hardware platform.

3.4 Workload Parameters

One of the most challenging aspects of studying commercial applications is appropriately sizing the workloads to make them manageable for architectural studies without severely altering the underlying behaviors that are being studied. This section briefly describes the actual setup and parameters that we use for each of our workloads. Section 6 discusses the validation of our approach by analyzing the impact of our scaling assumptions.

[1]The Alpha 21164 can count at most three different events during any given run. We use only a single count per run to keep any perturbation from counter overflow interrupts to a minimum.

There are three key issues for monitoring experiments: (i) amount of disk and physical memory required, (ii) bandwidth requirement for the I/O subsystem, and (iii) total runtime. For database workloads, scaling down the size of the database and the block buffer cache size (in the SGA) directly addresses (i) above, and typically leads to more manageable runtimes as well.[2] The I/O subsystem requirements are more difficult to address, since hiding the I/O latency in such workloads typically requires very large disk arrays for parallel access. Given the limitations in the capacity of the I/O subsystem on our experimental platform, we chose to make the relevant portions of the database memory resident by appropriately sizing the block buffer cache relative to the database size. However, we still use the same number of processes per processor as an out-of-memory database (even though this may no longer be necessary for hiding I/O latencies). The above policy turns out to be important for capturing a representative memory behavior in such workloads (see Section 6).

The OLTP workload uses a TPC-B database with 40 branches, corresponding to a 900MB database. The Oracle SGA is set to 1 GB to make the database memory resident, and the block size in the block buffer cache is set to 2KB. It is impossible to eliminate all I/O in an OLTP workload since every transaction commit requires the redo log to be written to disk. Fortunately, the I/O bandwidth requirements on the log disk can be easily matched by standard SCSI disks. Our experiments consist of 28 server processes (7 per physical processor), each executing 1500 transactions. To allow for reproducible results, we shutdown and restart the OLTP database, and warm up the SGA, before each measurement run.

The DSS workload uses a 500MB TPC-D database, with the SGA set to 1 GB with 32KB blocks. We use a parallelism degree of 16 for table operations, leading to 4 server processes per processor; some of the queries exhibit pipelined table operations, leading to 8 server processes per processor. We warm up the SGA by running a given query at least once before starting the measurement experiments.

Our strategy for studying AltaVista is different because we were able to do our monitoring experiments on the actual hardware platform used to run this application. Therefore, we did not do any scaling of the database. However, we limited the number of queries (replayed from a saved log of queries) to achieve approximately a 10 minute run per experiment. We use a total of 30 server threads.

The issues in scaling the workload for simulation studies are slightly different. High simulation speed (about one order of magnitude slower than the host system) is essential for full system simulation to position the workload to the start of the query with a warmed up database. Unfortunately, the slow speed of detailed processor and memory system simulators constrains the length of the workload actually studied. The use of simulation does have a certain number of advantages, for example making it perfectly feasible to model an aggressive I/O subsystem.

Simulations of commercial workloads based on user-level activity alone require further careful simplifications that are briefly discussed in Section 6.

[2]For OLTP and AltaVista, the total runtime is a function of total number of transactions/queries that are executed, and not the size of the database.

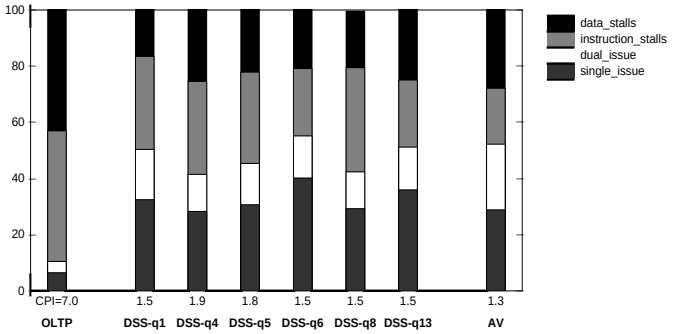

Figure 3: Basic cycle breakdown.

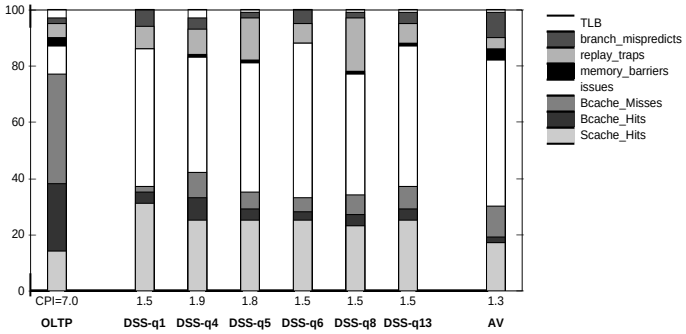

Figure 4: Detailed estimated cycle breakdown.

4 Monitoring Results

Our monitoring studies primarily use the hardware event counters of the Alpha with the IPROBE tool. Gathering a full set of event counts requires a large number of individual runs, and ensuring consistent behavior across multiple runs is an important issue. All event count measurements automatically factor out the execution of the idle thread.

Figure 3 presents the basic breakdown of execution time into four components: single- and double-issue cycles, and instruction- and data-related stall cycles.[3] We also show the cycles per instructions (CPI) directly calculated from the event counts. This figure shows a clear distinction between OLTP which has a CPI of 7.0 and DSS and AltaVista with CPIs in the 1.3-1.9 range. All workloads show a significant fraction of stall cycles, with instruction stalls being as important as data stalls.

By using a wider set of event counts, combined with latency estimates, we are able to calculate the approximate breakdown of cycles into more detailed categories, such as those shown in Figure 4. Three categories show the cycles spent at each level of the cache hierarchy. The issue category is the sum of single- and dual-issue cycles. The other categories are pipeline and address-translation related stalls. The replay trap is specific to the Alpha 21164 implementation and is used in lieu of stalling the pipeline due to resource hazards. We used a latency of 5 cycles for branch mispredicts and replay traps and 20 cycles for TLB misses. For memory stalls, we used the dependent load latencies of Section 3.1. The memory system breakdowns are only approximate since we can not account for possible overlap among accesses.

Table 2 presents the cache miss rate at all levels of the cache hierarchy for the three workloads. The local miss rate is the ratio of accesses that miss in the cache relative to those that reach that cache. The fraction of dirty misses is relative to Bcache misses.[4] The following sections cover the behavior of each of the workloads in greater detail. All three sections refer to Figures 3 and 4 and to Table 2.

[3] Although the 21164 is a quad-issue machine, two of the slots are used for floating-point instructions. As a result, our workloads practically never triple- or quad-issues.

[4] Dirty misses are Bcache misses that have to be fetched from another processor's cache.

4.1 OLTP

Figure 3 clearly shows that OLTP suffers from a very low issue rate, especially when compared to DSS or AltaVista. The cycle breakdown shows a CPI of 7.0 for our OLTP workload based on TPC-B, which is about twice the CPI of audit-size TPC-C runs on the same family of hardware servers [4]. This difference is primarily due to more complex and compute intensive queries in TPC-C and the better locality of that workload due to the higher number of small database tables. Nevertheless, the memory system trends of both workloads match quite well.

Figure 4 shows that over 75% of the time is spent stalling for memory accesses and Figure 3 indicates that those stall cycles are about evenly divided between instruction- and data-related stalls. Perhaps more surprising is that about half of the memory system stalls are to accesses that hit in the secondary and board-level caches (Scache and Bcache). This behavior underscores the importance of having very low-latency to non-primary caches, where the most significant working set resides.

The results in Table 2 show that the workload overwhelms all on-chip caches (Icache, Dcache and Scache). The performance of the Icache is particularly poor and explains the large component of instruction-related stalls in Figure 3. The workload does exhibit some locality in the Bcache (local miss ratio of 19%). Since the Bcache miss rate is still significant, both the latency component due to Bcache misses (memory and dirty miss stall cycles) and the bandwidth requirements on the interconnect are quite significant in multiprocessor configurations. Even though the AlphaServer 4100 has one of the fastest memory systems in its class, the intense traffic generated by OLTP leads to higher average latencies.

The latency of dirty misses, which account for over 15% of all Bcache misses in OLTP, is also a significant factor. The importance of dirty misses is heightened by two factors. First, current CPU designs tend to optimize accesses to the Bcache from the processor at the expense of interventions by external requests. For example, the base dependent load latency in our system for dirty misses is 125 cycles (417 ns) as opposed to 80 cycles (267 ns) for memory. This trend is prevalent on other current multiprocessors with latencies of 742 ns (dirty) vs. 560 ns (clean) on the Sun Enterprise 10000 and of 1036 ns (dirty) vs. 472 ns (clean) on the SGI Origin 2000 [8].

Second, the fraction of dirty misses increases with both the size of Bcaches and with the number of CPUs. The fraction of

	OLTP	DSS-Q1	DSS-Q4	DSS-Q5	DSS-Q6	DSS-Q8	DSS-Q13	AltaVista
Icache (global)	19.9%	9.7%	8.5%	4.6%	5.9%	3.7%	6.7%	1.8%
Dcache (global)	42.5%	6.9%	22.9%	11.9%	11.3%	11.0%	12.4%	7.6%
Scache (global)	13.9%	0.8%	2.3%	1.0%	0.6%	1.0%	1.0%	0.7%
Bcache (global)	2.7%	0.1%	0.5%	0.2%	0.2%	0.3%	0.3%	0.3%
Scache (local)	40.8%	3.6%	10.7%	5.7%	3.9%	6.0%	6.1%	7.6%
Bcache (local)	19.1%	13.0%	21.3%	23.9%	30.7%	27.9%	31.3%	41.2%
Dirty miss fraction	15.5%	2.3%	2.2%	10.6%	1.7%	8.4%	3.3%	15.8%

Table 2: Cache miss rate for OLTP, DSS, and AltaVista.

Type	Accesses	Bcache Misses	Dirty Misses
Instructions	72.5%	49.4%	0.0%
Private data	20.7%	15.0%	0.0%
Shared data (meta-data)	6.0%	32.0%	95.3%
Shared data (block buffer)	0.8%	3.6%	4.7%

Table 3: Breakdown of OTLP server process access patterns.

dirty misses was observed to increase to 20% for the same configuration running on a four CPU AlphaServer 8400 which has 4MB Bcaches, and to 24% when using eight processors. Our simulation results using SimOS show that this fraction can exceed 50% on 4MB two-way set-associative caches with only four processors. This indicates that dirty misses are a key factor for the performance of OLTP on larger machines.

Table 3 shows a breakdown of the types of server process accesses that cause Bcache and dirty misses. These results are based on cache simulations using ATOM-derived traces. Although the block buffer area is much larger than the meta-data area, it accounts for only 12% of the shared memory accesses, 10% of the Bcache misses to shared data, and under 5% of the dirty misses. In fact, accesses to the meta-data lead to a large fraction of Bcache misses, and account for virtually all dirty misses.

4.2 DSS

In contrast to OLTP, the DSS queries are much more efficient, with instructions taking 1.5 to 1.9 cycles on average. Table 2 shows that the first level Icache and Dcache are not completely successful in keeping the miss rates low. However, the 96K Scache is extremely effective in capturing the main footprint for this workload; the worst local miss rate for the Scache is about 10% for Q4. The approximate breakdown of time shown in Figure 4 clearly shows this effect where the memory system time is dominated by hits in the Scache. The Bcache is also reasonably effective in reducing misses to the memory system, given the relatively small size of the on-chip caches; most misses at this level are to the large database tables that are accessed through sequential table scans and exhibit good spatial locality. Finally, the fraction of Bcache misses that are dirty in another processor's cache are generally low, with the larger fractions occurring in the two more complex queries (Q5 and Q8). Based on the above observations, the biggest improvements in the memory hierarchy are likely to come from larger first level caches.

Figure 4 also shows that the replay trap component in DSS queries constitute a visible fraction of the stall time, especially in Q5 and Q8. Our detailed event counter experiments show that these replay traps are primarily due to the write buffer or miss address file being full.

4.3 AltaVista

The AltaVista experiments utilized one of the AlphaServer 8400's at the Palo Alto site during the summer of 1997. The AlphaServer 8400 had eight 300MHz 21164 processors, each with a 4MB Bcache, and 8GB of main memory. Although the AlphaServer 8400 differs from the AlphaServer 4100 in terms of Bcache and system bus, both platforms are based on the same processors.

Table 2 shows the cache performance of AltaVista. AltaVista has the lowest CPI of all the application that we studied. There are two main reasons for that: low first-level cache miss ratios and high fraction of dual-issue cycles. The instruction working set fits into the Icache, and the data working set fits into the Scache, although with relatively good Dcache performance. The effectiveness of the on-chip caches renders the Bcache fairly ineffective. In fact, over 40% of the accesses that reach the Bcache lead to a miss. AltaVista's next significant working set is the entire index, which does not even fit into main memory. Figure 3 also shows a high fraction of dual-issue cycles, due to the careful coding of AltaVista's main loop.

4.4 Summary of Monitoring

We found the monitoring experiments to be critical for understanding the behavior of commercial workloads, and for fine-tuning them to achieve representative performance. Good monitoring support, such as the rich set of event counters on Alpha 21164, are invaluable for such a study. These measurements indicate that both hits and misses to the secondary caches, as well as the latency of dirty misses, are important for OLTP. In contrast, the only significant memory system component of DSS and AltaVista is hits in the secondary on-chip cache.

5 Simulation Results

We complement our direct observations of workload execution with simulation experiments that go beyond the limitations of monitoring, specifically to study the sensitivity of various architectural parameters and to classify the memory sharing behavior

Event	OLTP			DSS Q6		
	SimOS	HW	error	SimOS	HW	error
Instructions	99.5	100.0	0.5%	101.1	100.0	1.1%
Bcache misses	2.25	2.39	5.7%	0.15	0.14	9.1%
User cycles	66.5%	71%		88.5%	89%	
Kernel cycles	16.9%	18%		2.2%	2.5%	
PAL cycles	6.0%			1.0%		
Idle cycles	10.7%	11%		8.4%	8.5%	

Table 4: Comparison of events measured on hardware platform and simulator.

of the workloads. We used SimOS on the OLTP workload and on two representative queries of DSS. Q5 is a complex hash join and Q6 is a simple scan. For OLTP, we start all clients and servers and run 4000 transactions to position the workload. We then study in detail the execution of 1000 transactions. For Q5 and Q6, we run the query once to position the workload and then study the second execution in detail.

We first configure SimOS to resemble the four processor AlphaServer 4100 hardware used in the monitoring experiments. This allows us to calibrate and validate the simulator against the monitoring results. These results are presented in Section 5.1. We then evaluate the sensitivity of architectural parameters on the workload execution and memory system performance. We first discuss the memory system and sharing behavior of OLTP in Section 5.2. We then discuss the sensitivity of both workloads to on-chip cache sizes in Section 5.3 and their sharing patterns as a function of the coherence granularity in Section 5.4.

5.1 Simulation Methodology and Validation

Although we configured SimOS to resemble the performance characteristics of the hardware, there are some differences between the platforms. SimOS models a single-issue pipelined processor with a two-level cache hierarchy, including separate 32KB two-way set associative instruction and data caches and a unified 2MB off-chip cache, all with a 64-byte line size. We do not model the second-level on-chip cache of the 21164. In addition, the simulation model uses sequential consistency, as opposed to the weaker Alpha memory model of our hardware platform. The differences in processor pipeline, cache hierarchy, consistency model, and I/O subsystem preclude us from performing timing comparisons with the hardware platform. However, we can compare the behavior of a large number of key execution components.

Table 4 compares a few execution statistics of the hardware and simulation platforms. The table shows the number of instructions executed (including non-idle operating system time) and the off-chip cache miss rate. In addition, we show the breakdown of cycles into four categories. The PAL time corresponds to the execution of privileged instructions sequences (called PALcode in Alpha) such as reloading the TLB. The instruction counts match almost perfectly. Bcache miss rates also match rather closely, especially considering the minor difference in the operating systems versions, which includes different versions of the page coloring algorithm. The cycle breakdowns also show a very close match. These times are measured using DCPI on the hardware platform, which does not separate PAL from

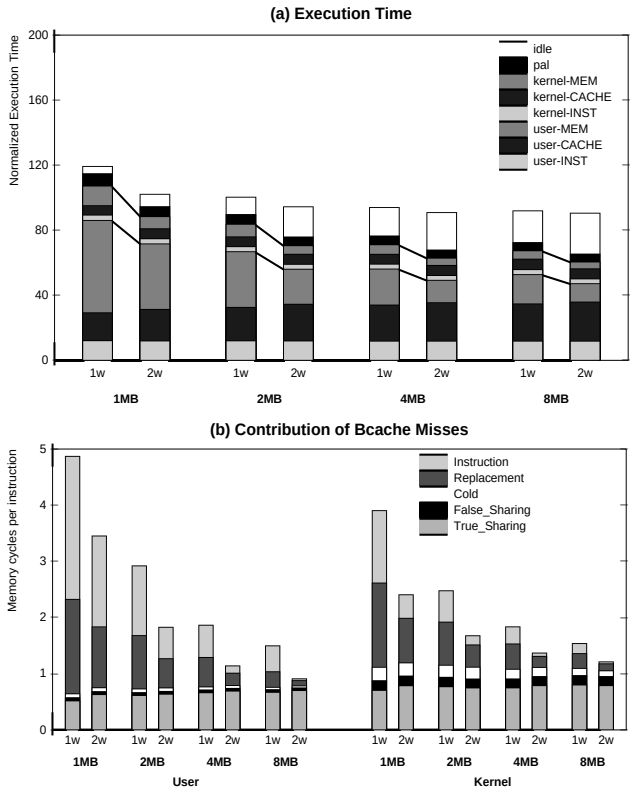

Figure 5: OLTP execution time (a) and Bcache behavior (b) for different cache sizes and associativity (P=4).

user or kernel time. Overall, this data shows that SimOS provides an accurate simulation of a real system.

5.2 Sharing patterns of OLTP

We first study the impact of Bcache sizes and associativity on the performance of OLTP workloads. Figure 5(a) shows the normalized execution time for various cache sizes and 1- and 2-way associativity (1w,2w) on a four processor configuration. All times are normalized to the execution of the 2MB direct-mapped cache. The figure also shows the breakdown of execution time into user, kernel, PAL and idle components. User and kernel are further categorized by instruction execution (INST), stalls within the cache hierarchy (CACHE), and memory system stalls (MEM). The first observation is that the kernel component is much smaller than the user component for all data points. Both user and kernel benefit from increases in cache sizes and higher associativity, with 2-way associativity typically providing the performance of the next bigger direct-mapped cache. Nonetheless, cache and memory system stalls remain a big component of the execution time even at the largest configuration. The second observation is that the idle time increases with bigger caches. This is a consequence of keeping the same number of server processes, which can no longer hide the I/O latency for configurations with increased processing rates. However, a realistic configuration with larger cache sizes would simply exploit more servers.

Intuitively, higher processing rates are expected to increase the

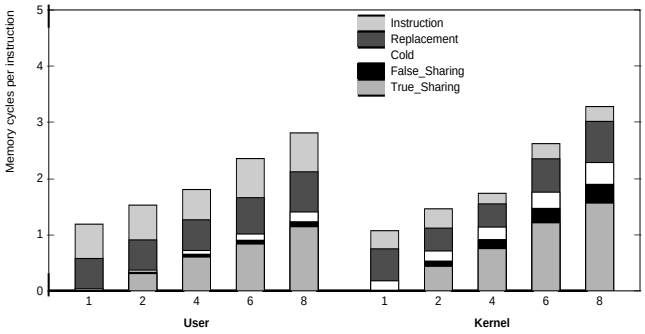

Figure 6: Contribution of Bcache stalls at different number of processors (1-8P, 2MB 2-way Bcaches).

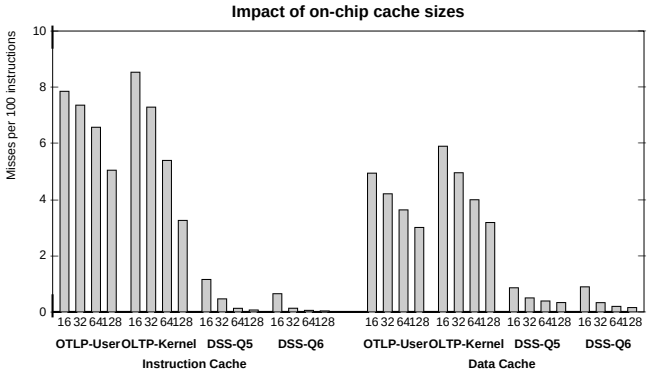

Figure 7: On-chip cache miss-rates for OLTP and DSS for split I/D caches (16KB-128KB).

demands on the I/O subsystem for OLTP. However, our simulations show an interesting interaction that results in the opposite effect. A more efficient memory system results in more transactions being ready to commit each time the log writer is activated. The batching of commits from independent transactions into a single log file operation leads to a more efficient use of the log disk. Going from the 1MB direct-mapped configuration to the 8MB 2-way configuration, we see a 33% reduction in the number of writes to the database log and a 20% reduction in the total size of these writes.

Figure 5(b) shows the contribution of Bcache misses to the execution time in terms of cycles per instruction. We have separated the effect of user and kernel misses. For each configuration, we also show the breakdown into five categories. Communication misses are separated into true and false sharing using the methodology of Dubois et al. [6].[5] Data capacity and conflict misses are merged into a single replacement category. Data cold misses and instruction misses complete the breakdown.

Let us focus on user misses. There are large gains from set-associativity, even at large cache sizes. This can be partly attributed to the page mapping decisions made by the operating system, which may be limiting the performance of direct-mapped caches.[6] Therefore, we will focus on results for 2-way caches in order to offset the conflicts caused by the page mapping policy. Instruction misses and data replacement play an important factor up to 4MB, at which point the useful working set is captured by the cache. As expected, communication misses dominate at the larger cache sizes. Interestingly, only a surprisingly small fraction of the communication in Oracle is due to false sharing. The trends are similar for kernel misses, although false sharing is a more important factor.

Our more detailed statistics show that the fraction of cache-to-cache transfers (dirty misses) increases with improvements of the memory system, as observed in the monitoring results. With 4MB and 8MB 2-way set associative caches, the fraction of dirty misses is 55% and 62% respectively. Clearly, trade-offs that penalize dirty misses will have a severe impact on OLTP performance.

We also looked at the impact of the number of processors on the sharing patterns. Again, we chose to model 2MB 2-way set associative caches to compensate for the excessive conflicts in direct-mapped caches. Figure 6 shows that communication stalls increase linearly with the number of processors for both user and kernel and that the operating system communicates slightly more per instruction that the application. The ratio of false sharing to true sharing is independent of the number of processors. Instruction and data replacement misses are essentially unaffected by the size of the system. Even at eight processors, the contribution of kernel misses to the overall execution time remains low. The above results emphasize the relative importance of communication misses for larger processor configurations.

5.3 Cache Hierarchy Performance

Primary cache misses that hit in the cache hierarchy play a significant role for OLTP and a dominant role for DSS. Figure 7 shows the miss-rates of the workloads as a function of the on-chip cache size for four processor configurations. In all cases, we assume split instruction and data caches that are 2-way set associative. User and Kernel misses are separated only for OLTP. OLTP and DSS have very different cache performance, with DSS having significantly lower cache miss rates. Larger on-chip caches are effective for both OLTP and DSS. While large on-chip caches can capture most of the misses in DSS, it is unlikely that the working sets of OLTP can fit into current or next generation on-chip caches.

5.4 Sensitivity to Cache Line Size

Figure 8 shows the impact of line size on the Bcache miss rate for a four processor configuration. Again, we model a 2MB 2-way set associative cache. The data communicated between server processes (true sharing) exhibits good spatial locality in both OLTP and DSS Q5, probably due to the use of producer-consumer communication through buffers. As expected, there is an increase in the amount of false sharing, although this remains surprisingly low for the database and moderate for the kernel. Streaming data which is classified as cold misses by our simulator also exhibits great spatial locality, especially in DSS Q6. At the same time, the replacement and instruction miss rate are not visibly affected by changes in the line size. Therefore, bigger cache lines may be more effective at larger cache sizes where

[5] Communication misses do not imply a dirty cache miss, since the data may have been written back to memory.

[6] The page mapping policies of Digital UNIX have improved since the version we used in these experiments.

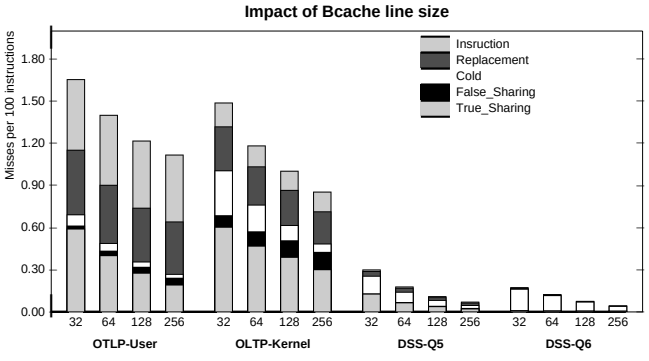

Figure 8: Bcache miss rate for different line sizes (32-256 bytes, 2MB 2-way Bcaches).

most replacement and instruction misses have already been eliminated.

The miss rate metric shown in Figure 8 obviously does not account for the increased latency and bandwidth requirements of larger cache lines. In fact, with simulation parameters based on an Alpha 21164 processor, the performance peaks at 128B for OLTP. For DSS, line sizes of 128B and 256B have nearly identical performance.

5.5 Summary of Simulation Results

Our simulation studies complement our monitoring experiments. The ability to look at more detailed information such as the classification of cache misses is invaluable in developing a deeper understanding of these workloads. The simulation results suggest that OLTP workloads continue to benefit from larger Bcaches up to the 4-8MB range. Since most of the misses at that point are true-sharing misses, there appears to be no benefit from going beyond such sizes. However, primary working sets of DSS appear to fit comfortably into next generation on-chip cache sizes, making DSS somewhat oblivious to the Bcache design parameters.

6 Impact of Scaling

This section briefly describes the experiments we performed to confirm that the way we scale our workloads does not affect their memory system characteristics. As discussed in Section 3.4, there are three key issues to consider: (i) amount of disk and physical memory required, (ii) bandwidth requirement for the I/O subsystem, and (iii) total runtime.

In the case of AltaVista, we did not have to do any scaling to the application aside from limiting the time of each run as discussed in Section 3.4. Furthermore, given the extremely high memory system efficiency that AltaVista achieves, we did not try to further study a scaled down version through simulation.

For our database workloads, addressing the above issues involved scaling down the database size, sizing the SGA appropriately to make the database memory resident, and limiting the number of transactions executed (in the case of OLTP). For both the OLTP and DSS workloads, we sized the database so that it just fit in the main memory of the server used for the monitoring experiments. Although this makes the database much smaller than typical sizes, all relevant data structures are still orders of magnitude larger than the size of the hardware caches in our systems. Therefore, given the access patterns of OLTP and DSS, our workloads are big enough (by a wide margin) for memory system studies.

Having an in-memory experiment drastically reduces the disk capacity and bandwidth requirements of the system. Since I/O is no longer an issue, we can store the database and logs on regular file systems rather than raw devices. In this situation, it is important to limit the size of the file system buffer cache to avoid double buffering problems.

In our memory-resident studies we used as many server processes as would have been required to hide the I/O latencies in an out-of-memory run. We found this critical as the number of servers per processor has a significant impact in the cache behavior of the application. For example, in OLTP, going from 8 to 28 server processes in our hardware platform decreased the fraction of dirty misses by a factor of two (due to an increase in capacity/conflict misses). Similarly, the DSS Q5 query sees an increase of 20% in the Bcache miss rate when going from 4 to 16 server processes.

Another indication of the validity of our scaling methodology for DSS is that the actual run times of those queries that scale with the problem size [7] correlate extremely well (i.e., by the scaling factor) with the run times for an audited (100GB) database on the same platform.

Scaling issues for full system simulation environments such as SimOS are very similar to those described above since one can simulate reasonably large machines. For user-level simulations however, other simplifications beyond scaling have to be addressed. The following are observations from experiments we have done based on tracing user-level activity with ATOM [17] to determine the viability of using such traces. It is important to consider the effect of not modeling operating system instructions. For OLTP, operating system activity is non-negligible but it also does not dominate the memory system behavior in a well tuned system. Moreover, the trace generator must compensate for the effects of time dilation resulting from the binary instrumentation by adjusting all real-time and timeout values. More importantly, the trace simulator must appropriately emulate the scheduling and page mapping policies of a multiprocessor operating system. The scheduling decisions take place at each blocking system call (I/O requests, process synchronization) and additionally at regular intervals that correspond to the operating system scheduling quantum. Finally, the memory behavior of the database engine is dominated by the server processes, allowing the memory activity of daemon processes to be ignored as a simplification. This methodology has been used to study the effectiveness of simultaneous multithreading on database workloads [10]. Overall, while user-level simulation studies are feasible, they require a deep understanding of the system implications of the workload to ensure representative results.

[7] In our study those queries are Q1, Q5 and Q6.

7 Related Work

Our work extends previous studies by combining careful monitoring of real systems and detailed full system simulation in the context of commercial grade OLTP, DSS and Web index server applications. This combination allows us to identify key memory system performance issues and analyze their trends. One such trend is the increasing sensitivity of OLTP applications to dirty miss latency.

Some previous studies have identified different bottlenecks for the performance of OLTP applications than the ones we underscore. Thakkar and Sweiger [20] have measured the performance of an early OLTP benchmark (TP1) in the Sequent Symmetry multiprocessor, using both memory-resident and out-of-memory databases. Rosenblum et al. [16] focused on the impact of architectural trends on the operating system performance using three applications, including TPC-B. Both studies identify I/O performance as the critical factor for OLTP applications, although Rosenblum et al. point out that memory system performance will be the key bottleneck once the I/O problem has been addressed. We have observed that the I/O problem has indeed been addressed in modern database engines, and that memory system performance is already the main bottleneck. Perl and Sites [14] study Microsoft SQL Server performance on an Alpha-based Windows NT server through trace-driven simulation. They claim that OLTP performance is mostly limited by chip pin bandwidth, since the bandwidth that would be required to maintain a CPI of 1 in the Alpha 21164 would be much greater than what is available in that processor. We believe that their conclusion is influenced by the fact that they ignored the latency-dependent nature of the workload. An OLTP application has a large fraction of dependent loads and branches, which slows down the issue rate well before bandwidth becomes a limiting factor.

A few studies have raised the level of awareness in the architecture community to the fact that OLTP workloads have a very different behavior when compared with scientific applications. Cvetanovic and Donaldson [4], in their characterization of the AlphaServer 4100, measure the performance of SPEC95 programs, the TPC-C benchmark and other industry benchmarks. Our paper differs from this study in that it focuses on the memory system, uses other commercial applications (in addition to OLTP), and extends system monitoring with detailed simulations. Maynard et al. [12] also compare commercial and scientific applications using trace-driven simulation, but use a uniprocessor system instead of a multiprocessor. As in our study, they recognize the importance of Icache misses in OLTP applications. However, the fraction of operating system activity in their studies of OLTP was more than twice what we have obtained after tuning our OLTP application.

Some researchers have used database workloads to study various aspects of system design. Lovett and Clapp [11] describe a CC-NUMA multiprocessor system for the commercial marketplace, and evaluate it using abstract models that are based on the the behavior of TPC-B and TPC-D/Q6. Although they agree with our observation that TPC-B memory behavior is representative of TPC-C, the limitations of their methodology do not allow them to study memory system performance in the level of detail done here. Eickemeyer et al. [7] take a uniprocessor IBM AS/400 trace of TPC-C and transform it to drive both a simulator and analytic models of coarse-grain multithreaded uniprocessors. They conclude that multithreading is effective in hiding latency of OLTP. Verghese et al. [24] use a DSS workload to evaluate operating system support for page migration and replication.

Trancoso et al. [21] use a public domain database engine to perform a detailed study of the memory performance of some TPC-D queries. However their engine could not automatically parallelize the queries, nor had the efficiency of the Oracle engined used here. Therefore their execution patterns differ substantially from ours. The use of a user-level simulator also prevents the study of system interactions.

8 Concluding Remarks

This paper presents a detailed characterization of three important workloads: online transaction processing (OLTP), decision support systems (DSS), and Web index search. We use a combination of hardware monitoring experiments augmented with full system simulations to identify the critical memory system design issues for each of the three workloads. A careful scaling of these workloads allows for manageable architectural studies without compromising representative memory system behavior.

Contrary to some previous studies, we find that operating system activity and I/O latencies do not dominate the behavior of well-tuned database workloads running on modern commercial database engines. Our three workloads stress the memory system in different ways. OLTP has significant instruction and data locality that can only be effectively captured by large off-chip caches. OLTP also exhibits a high communication miss rate which makes the workload sensitive to both memory and dirty miss latencies. Increases in cache sizes and number of processors will make dirty misses play an even more critical role in overall system performance. In contrast, DSS and AltaVista are primarily sensitive to the size and latency of on-chip caches. As such, these applications are somewhat insensitive to the off-chip cache size and to memory and dirty miss latencies.

As a consequence of these workload differences, systems optimized for OLTP versus DSS and Web index search workloads may lead to diverging designs. A system for DSS and Web index searching may opt to trade large board-level caches for either lower memory latencies or lower cost. Such a design point will clearly lead to poor OLTP performance.

We find the combination of monitoring and full system simulation techniques to be extremely valuable for the comprehensive study of complex applications, and plan to push this technique for studying the scalability issues of larger systems.

Acknowledgments

We would like to thank Robert Stets and Arun Sharma for their contributions to the early part of this work, Jef Kennedy from Oracle for reviewing the manuscript, Marco Annaratone for supporting this research, John Shakshober, Kalyan K. Das, and Charles Wang for their assistance with the database workloads, Mike Burrows for his help in setting up the AltaVista experiments, and Drew Kramer for his technical assistance in setting up our hardware platform. We also thank Jennifer Anderson, Jeff Dean, Alan Eustace, and Jeff Mogul from WRL for their many suggestions and careful reviewing of the final manuscript.

References

[1] J. M. Anderson, L. M. Berc, J. Dean, S. Ghemawat, M. R. Henzinger, S.-T. Leung, R. L. Sites, M. T. Vandervoorde, C. A. Waldspurger, and W. E. Weihl. Continuous profiling: Where have all the cycles gone? In *Proceedings of the 16th International Symposium on Operating Systems Principles*, pages 1–14, Oct 1997.

[2] M. Burrows. Private communication.

[3] Z. Cventanovic and D. Bhandarkar. Performance characterization of the Alpha 21164 microprocessor using TP and SPEC-workloads. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 60–70, Apr 1994.

[4] Z. Cvetanovic and D. D. Donaldson. AlphaServer 4100 performance characterization. *Digital Technical Journal*, 8(4):3–20, 1996.

[5] Digital Equipment Corporation. *Digital Semiconductor 21164 Alpha microprocessor hardware reference manual*, March 1996.

[6] M. Dubois, J. Skeppstedt, L. Ricciulli, K. Ramamurthy, and P. Stenstrom. The detection and elimination of useless misses in multiprocessors. In *Proceedings of the 20th International Symposium on Computer Architecture*, pages 88–97, May 1993.

[7] R. J. Eickemeyer, R. E. Johnson, S. R. Kunkel, M. S. Squillante, and S. Liu. Evaluation of multithreaded uniprocessors for commercial application environments. In *Proceedings of the 21th Annual International Symposium on Computer Architecture*, pages 203–212, June 1996.

[8] C. Hristea, D. Lenoski, and J. Keen. Measuring memory hierarchy performance of cache-coherent multiprocessors using micro benchmarks. In *Proceedings of Supercomputing '97*, November 1997.

[9] T. Kawaf, D. J. Shakshober, and D. C. Stanley. Performance analysis using very large memory on the 64-bit AlphaServer system. *Digital Technical Journal*, 8(3):58–65, 1996.

[10] J. L. Lo, L. A. Barroso, S. J. Eggers, K. Gharachorloo, H. M. Levy, and S. S. Parekh. An analysis of database workload performance on simultaneous multithreaded processors. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, June 1998.

[11] T. Lovett and R. Clapp. STiNG: A CC-NUMA computer system for the commercial marketplace. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 308–317, May 1996.

[12] A. M. G. Maynard, C. M. Donnelly, and B. R. Olszewski. Contrasting characteristics and cache performance of technical and multi-user commercial workloads. In *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 145–156, Oct 1994.

[13] J. D. McCalpin. Memory bandwidth and machine balance in current high performance computers. In *IEEE Technical Committee on Computer Architecture Newsletter*, Dec 1995.

[14] S. E. Perl and R. L. Sites. Studies of windows NT performance using dynamic execution traces. In *Proceedings of the Second Symposium on Operating System Design and Implementation*, pages 169–184, Oct. 1996.

[15] M. Rosenblum, E. Bugnion, S. A. Herrod, and S. Devine. Using the SimOS machine simulator to study complex computer systems. *ACM Transactions on Modeling and Computer Simulation*, 7(1):78–103, Jan. 1997.

[16] M. Rosenblum, E. Bugnion, S. A. Herrod, E. Witchel, and A. Gupta. The impact of architectural trends on operating system performance. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 285–298, 1995.

[17] A. Srivastava and A. Eustace. ATOM: A system for building customized program analysis tools. In *Proceedings of the SIGPLAN '94 Conference on Programming Language Design and Implementation*, pages 196–205, June 1994.

[18] Standard Performance Council. *The SPEC95 CPU Benchmark Suite*. http://www.specbench.org, 1995.

[19] G. Sturner. *Oracle7. A User's and developer's Guide*. Thomson Computer Press, 1995.

[20] S. S. Thakkar and M. Sweiger. Performance of an OLTP application on Symmetry multiprocessor system. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 228–238, June 1990.

[21] P. Trancoso, J.-L. Larriba-Pey, Z. Zhang, and J. Torrellas. The memory performance of DSS commercial workloads in shared-memory multiprocessors. In *Third International Symposium on High-Performance Computer Architecture*, Jan 1997.

[22] Transaction Processing Performance Council. *TPC Benchmark B (Online Transaction Processing) Standard Specification*, 1990.

[23] Transaction Processing Performance Council. *TPC Benchmark D (Decision Support) Standard Specification*, Dec 1995.

[24] B. Verghese, S. Devine, A. Gupta, and M. Rosenblum. Operating system support for improving data locality on CC-NUMA computer servers. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 279–289, October 1996.

[25] E. Witchel and M. Rosenblum. Embra: Fast and flexible machine simulation. In *Proceedings of the 1996 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 68–79, May 1996.

[26] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the 22nd International Symposium on Computer Architecture*, pages 24–36, June 1995.