

Analyzing performance characteristics of OLTP cached workloads by linear interpolation

Trung Diep*, Murali Annavaram*, Hubert Nueckel[~], Brian Hirano[°], and John P. Shen*

*Microprocessor Research
Intel Labs

[~]Software Solutions Group
Intel Corporation

[°]Server Technologies
Oracle Corporation

Abstract— This is a report on understanding how varying the configuration parameters affects the behavior of an online transaction processing (OLTP) workload. In particular, this paper uses four important parameters, namely, number of processors (P), number of disks (D), number of warehouses (W), and number of concurrent client processes (C) to represent an OLTP configuration, and analyzes how these four parameters affect performance. We use the notation $\langle P, D, W, C \rangle$ to represent an OLTP configuration. Our goal is to study the effects of scaling $\langle P, D, W, C \rangle$ and formulate empirical relationships among the four parameters. We analyze both cached as well as partially cached setups and use linear interpolation to show how varying the four parameters changes critical workload behaviors. We use a small-scale shared memory multiprocessor (4-way) system with a small number of SCSI drives (4 drives) to build and explore a variety of OLTP configurations.

Index Terms—Online Transaction Processing, Performance Characterization, Workload Behavior.

I. INTRODUCTION

ONLINE transaction processing benchmarks are important workloads for the design and performance analysis of microprocessors and computer systems targeting the server market. Setting up and configuring an OLTP workload are nontrivial due to the complex interactions among the myriad of configuration parameters that need to be properly tuned for achieving good performance. Furthermore, the hardware costs of building a computer system with sufficient amount of system memory and disk I/O bandwidth can be substantial. Typically, researchers employ two types of setups for characterizing OLTP workloads: scaled or cached.

Scaled setups are large-scale OLTP systems typically set up by OEM system vendors and are intended to reflect the production environments of large enterprises. Scaled setups are used to showcase world-record TPM (transaction per minute) scores with a goal of pushing the limits of transaction processing throughput. These setups must comply with the Transaction Processing Council (TPC) specifications for TPC-C [11] to be certified. However, setting up an audit-sized TPC-C workload used for reporting TPM-C scores is both challenging and costly. OEM system vendors use expensive setups with large number of disks to achieve a high level of concurrency so that other independent transactions can be processed while those waiting for disk I/O are stalled. On a balanced system, these workloads with a database size of thousands of warehouses are dominated by disk I/O activities

and a working set size that far exceeds the capacity of the cache memory hierarchy and even system memory. Researchers use performance counters to monitor the behavior of these systems to identify performance bottlenecks [1][2][3][4][5].

Cached setups significantly scale down the database size (typically to 10 warehouses) so that the working set fits well in system memory. Once the most commonly accessed data blocks are brought into the system memory from disks, cached setups have negligible disk I/O reads. Studying cached-setup workloads is made more feasible because the bulk of the system behavior is on the interaction between CPUs and the memory subsystem with less dependence on the disk I/O subsystem. This reduction in scale permits the simulation of cached workloads to be more tractable with full-system simulators. Hence, cached setups are more widely used in research studies to assess architectural features [5][6][7][9][10]. These setups do not comply with the TPC-C specifications but allow researchers to explore new architectural features that are not currently implemented. Previous work [5] claimed that the behavior of a cached workload is a reasonable representation of a scaled workload.

This paper is an initial study on understanding how varying the configuration parameters affects the behavior of an OLTP workload. In particular, this paper uses four important parameters, namely, number of processors (P), number of disks (D), number of warehouses (W), and number of concurrent client processes (C) to represent an OLTP configuration, and analyzes how these four parameters affect performance. We use the notation $\langle P, D, W, C \rangle$ to represent an OLTP configuration. Even though W and C can be dependent on each other, we treat them as independent variables by making C less than W.

Our goal is to study the effects of scaling $\langle P, D, W, C \rangle$ and formulate empirical relationships among the four parameters. This paper is a first step towards achieving that goal by focusing on a subspace of this huge cross-product space. We analyze both cached as well as partially cached setups and use linear interpolation to show how varying the four parameters changes critical workload behaviors. We use a small-scale shared memory multiprocessor (4-way) system with a small number of SCSI drives (4 drives) to build and explore a variety of OLTP configurations.

While a large setup is needed to achieve record-breaking

transaction throughput and to comply with TPC-C requirements, our research is focused primarily on small-scale SMP systems. As processor performance continues to improve along with better memory and disk technologies, developing a better understanding of the multi-dimensional configuration space of OLTP workloads on small-scale SMP systems is important towards the exploration of future CMP (chip multiprocessor) designs for OLTP workloads, and this is the aim of our research.

In [12], they used a scaled OLTP workload and studied the workload behavior by varying the amount of memory and the number of warehouses. Using examples, they demonstrated that the configuration of the OLTP system could impact several key architectural and operating system characteristics, such as the breakdown of user and kernel time, the disk I/O rates, and the cycles per instruction (CPI) of the processors. They concluded that departures from a well-balanced scaled system can adversely affect the workload behavior and can mislead designers down the wrong path. In this paper, we analyze cached and partially cached setups and present detailed sensitivity analysis, supported by quantitative results, showing trends, similarities and differences between the two setups.

The key results of this paper are summarized below. (1) This paper presents a quantitative analysis of the OLTP configuration space of an Oracle-based OLTP workload, Oracle Database Benchmark (ODB), running on an SMP system with IA-32 processors. We use the terms scaled, cached, partially cached setups to describe the different types of workloads. (2) We show that the transaction throughput can be higher for a partially cached setup than that of a cached setup by increasing the number of warehouses from ten to twenty. Even though more disk I/O reads are made, the partially cached workload has sufficient work to tolerate the misses to the database buffer cache. (3) Using performance counters to monitor cached and partially cached OLTP workloads, we show how kernel and user execution times exhibit radically different trends relative to the transaction throughput. The kernel code has much worse instruction-level parallelism than user code; hence, the CPI (Cycles Per Instruction) of kernel code is higher than user code. We analyze the reasons for the CPI difference and show that the MPI (Cache Misses Per Instruction) is worse for kernel than the user code.

II. EXPERIMENTAL APPROACH

In our research, we use the Oracle database server as our experimental vehicle. We configure and run this workload on a small-scale multiprocessor system employing the Intel Xeon MP processors. The performance counters in the Xeon processors are used to collect our experimental data.

A. Oracle Database Benchmark

In this study, we use the Oracle Database Benchmark (ODB), which is derived from an internal OLTP benchmark using Oracle RDBMS 8.1.6. ODB simulates an order-entry

business where terminal operators (or clients) execute transactions against a database. The database is made up of a number of warehouses. Each warehouse supplies items to ten sales districts, and each district serves three thousand customers. Typical transactions include entering and delivering customer orders, recording payments received from customer, checking status of a previously placed order, and querying the system to check inventory levels at a warehouse.

When ODB starts execution, the underlying Oracle database spawns two types of processes: user processes and Oracle processes. A user process executes a client's application code. Oracle processes can be either server processes that perform the actual database work on behalf of the user or background processes that perform maintenance work. Two background processes of note are the database writer and the log writer. The database writer updates the modified database blocks to the disk in order to free the dirty buffers in memory, while the log writer records the log entries that describe the changes made to the database. As long as the log writer is proceeding faster than the rate at which the log buffers are being filled, the disk I/O impact from the log writer is usually small.

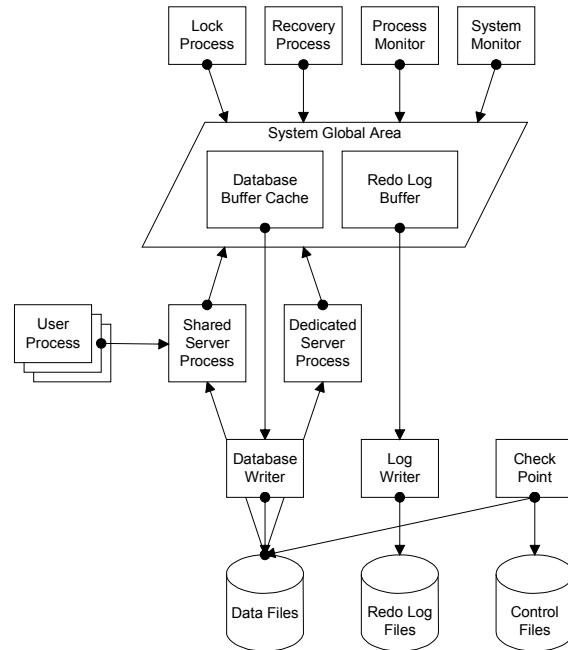


Figure 1: Overview of Oracle Database Server Processes.

As illustrated in Figure 1, all Oracle processes share a common large shared memory segment called the System Global Area (SGA). The SGA is a region of memory that is allocated when an Oracle database is started. The SGA is shared by the server processes as well as the background processes. For the purpose of this paper, there are four main areas within the SGA. The first area, usually the largest, is devoted to the database buffer cache, which holds parts of a database in memory. The database buffer cache tracks the usage of the database blocks so that it can keep the most recently and frequently used blocks in memory. The second area of interest is the redo log buffer. This is a circular buffer

that sequentially logs all relevant changes to the database. The third area is called the fixed SGA. It contains control block information that includes items such as the head of all lists, pointers to the heads of lists, and descriptors for the buffer cache. There are no user data in the fixed SGA. The last area, referred to as the SGA heap, has the remaining data structures in the SGA. The SGA heap contains shareable state of connected Oracle users as well as the execution state of SQL statements and bytecode for Oracle-implemented procedural languages (e.g., PL/SQL, Java).

Each server process has a private data segment called the Process Global Area (PGA). The PGA holds the data and control information and is allocated when a server process is spawned. The PGA is not shared with other processes and is divided into two parts: the fixed PGA and the PGA heap. The stack used by the process is part of the process' private memory.

B. Database Server Configurations

The Oracle RDBMS has numerous parameters in its configuration. For this study, the most important parameter is the size of the database buffer cache allocated in the SGA. The database buffer cache is intended to hold as much of the database working set as allowed in memory. 750K cache blocks, each with a size of 2KB, are allocated. The total amount of memory that is allocated to the buffer cache is 1.5 GB. Including other data structures, the total SGA size consumes about 1.8 GB of memory. Another 1 GB of memory is allocated to the OS kernel. This leaves about 1.2 GB of memory for the server processes and their corresponding PGA data structures. We do not use the large memory support in Oracle databases or the Intel physical addressing extensions to go beyond 4 GB of virtual memory space.

C. Linux OS and SMP Configurations

Measurement data are gathered on a 4-way SMP system running Red Hat Linux 7.2 using the kernel 2.4.9-34smp. The Intel Xeon MP processors operate at 1.6 GHz and have three levels of caches. The first level has an execution trace cache, while the second and third levels have unified instruction and data caches of 256 KB and 1 MB, respectively. The Xeon MP processors are based on the NetBurst microarchitecture and are capable of running with hyper-threading technology [13]. For the purpose of this study, we do not enable the hyper-threading technology. The system under study is populated with 4 GB of PC200 DDR memory using the ServerWorks Grand Champion HE chipset and has 4 Ultra160 SCSI drives, each with 73 GB of capacity.

D. EMON Performance Counters

The Xeon MP processor provides a comprehensive list of performance-monitoring events [14]. There are 18 performance counters grouped into 9 pairs, with each pair associated to a particular subset of events. The particular counters can be selected by specifying the counter configuration control registers. The main benefit to this type of

performance monitoring is that it is completely noninvasive and does not in anyway affect the actual execution of ODB.

III. CACHED SETUP MEASUREMENT DATA

We begin our exploration of the multi-dimensional configuration space of OLTP workloads by first looking at cached setups. We fix the number of warehouses at 10 ($W=10$) and vary the number of clients ($C=\{1, 5, 10\}$), processors ($P=\{1, 2, 4\}$) and disks ($D=\{1, 2\}$). Such cached setups are similar to those assumed in most published research papers. In Section 4, we expand this subspace to include more warehouses and clients when we explore partially cached setups.

A. Characteristics of a Cached Setup

A cached setup is defined as an OLTP setup that has a sufficiently large database buffer cache in the SGA to hold the working set of database blocks in memory. Because it has virtually no disk I/O reads, a cached setup is intended to achieve the highest transaction throughput by utilizing maximally the CPUs. The database server can execute and complete the transactions without relinquishing control and awaiting for data from disk I/O to finish. The only exception occurs when the data is first brought in from disk to memory, and this overhead is amortized over many transactions to be effectively insignificant. On average, the number of database blocks a transaction typically reads during the lifetime of a transaction is in the range of 50 to 60. Of those database blocks accessed, a transaction usually incurs about 3 to 4 disk I/O reads on average for a scaled setup. In a cached setup, the number of disk I/O reads per transaction drops to less than 0.1, or over an order of magnitude fewer disk I/O reads per transaction.

B. Impact of Processors and Disk I/O

Transaction processing throughput of a processor can be increased, by increasing the number of clients. A 10-warehouse database has been commonly used in the database research community as a cached setup. As shown in Figure 2, the transaction throughput increases with the number of concurrent clients accessing the 10-warehouse (10W) database on a single-processor (1P) system. More clients can be added to increase the amount of concurrency further; however, doing so would encounter two problems. The first is that once the maximum CPU utilization is reached, adding more clients does not increase the overall throughput, unless more CPUs are added. The second has to do with internal contention resulting from multiple clients accessing and updating the same database blocks. To resolve this contention, more warehouses will need to be added to support larger number of clients. Hence, adding more clients increases the amount of concurrent work to be performed and increases the overall transaction-processing throughput, unless the CPU resources or the database become a bottleneck.

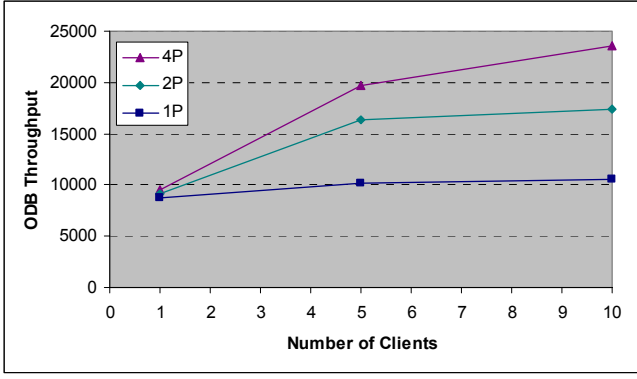


Figure 2: Transaction throughput for different number of clients and processors.

Employing more CPUs in a multiprocessor system can alleviate the CPU resource bottleneck. As can be seen with 1C in Figure 2, there is not sufficient amount of work to be performed, and there is little advantage for the 2P and 4P systems. As the number of clients is increased, the transaction throughput increases at rates proportional to the number of processors. At 10C, the throughput for a 4P system is about 2.4 times higher than that of a 1P system. The scaling is not linear due to the disk I/O writes made by the log writer. When the disk I/O is improved by striping the log files from 1 drive to 2 drives in a RAID0 configuration, the throughput is improved by another 16% on a 4P system as shown in Figure 3.

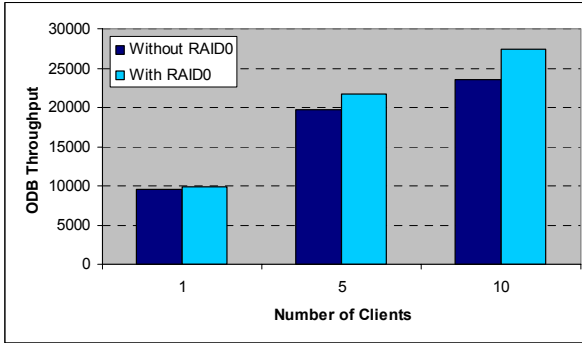


Figure 3: Transaction throughput for the log file with and without RAID0.

C. Impact of Database Buffer Cache Sizes

The achievable throughput is also influenced by the size of the database buffer cache. Figure 4 shows the transaction throughput for different database buffer cache sizes in blocks from 450K to 750K. The transaction throughput is lower by as much as 22% when the buffer cache size is configured to be 450K with one client active at a time. When the number of concurrent clients is increased to ten, the throughput difference narrows to about 6%. The improvement is due to the net benefits of constructive data sharing among multiple clients whereby the positive effects from data prefetching that is useful for another client outweigh the negative effects of cache pollution. Furthermore, at 600K, the database buffer cache is

large enough for this database that cache pollution is not a problem as seen by the non-increase in throughput for a 750K-block cache size. Since we are trying to maximize the database buffer cache size, the rest of the data presented in this paper are taken with a cache size of 750K blocks. On a system that supports 64 bits of address space, the buffer cache size can extend another dimension of the configuration space. For our case on a 32-bit system, varying the buffer cache size by making it smaller is not as interesting.

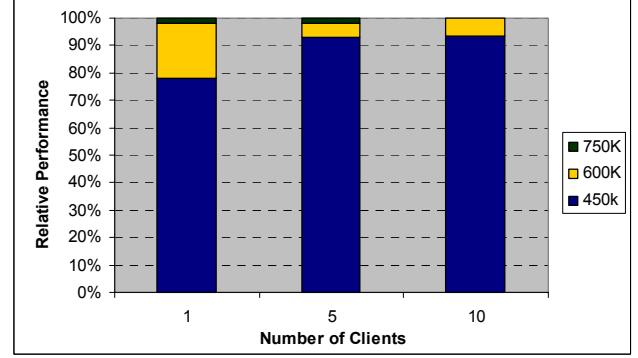


Figure 4: Transaction throughput for different database buffer cache sizes.

IV. PARTIALLY CACHED SETUP MEASUREMENT DATA

In Section III, the number of warehouses is restricted to ten. This section moves beyond cached setups to include partially cached setups that involve twenty to forty warehouses. The number of clients is expanded to range from one to forty. This expands our OLTP configuration subspace being explored to $\langle P=\{1, 2, 4\}, D=\{1, 2\}, W=\{10, 20, 40\}, \text{ and } C=\{1, 5, 10, 20, 40\} \rangle$.

A. Differences in a Partially Cached Setup

Whereas the database buffer cache in the SGA can hold an entire working set in a cached setup, a partially cached setup is defined as a setup in which the working set is larger than what the database buffer cache can store. The benefits of caching database blocks remain but are diminished by the eviction of potentially useful database blocks due to capacity constraints. We want to distinguish the partially cached setup from a scaled setup in which the working set is much larger than the database buffer cache by orders of magnitude. For our work, a partially cached setup involves tens of warehouses whereas a scaled setup can have up to thousands of warehouses. However, there are significant differences between cached and partially cached setups. Instead of averaging less than 0.1 disk I/O reads per transaction, partially cached setups average between 0.3 to 1.1 disk I/O reads per transaction. This is closer to the average of between 3 to 4 disk I/O reads per transaction for scaled setups. The resulting effects are that the transaction throughputs can be lower with more warehouses and that more concurrent clients are needed to mask the disk I/O overhead associated with database buffer cache misses.

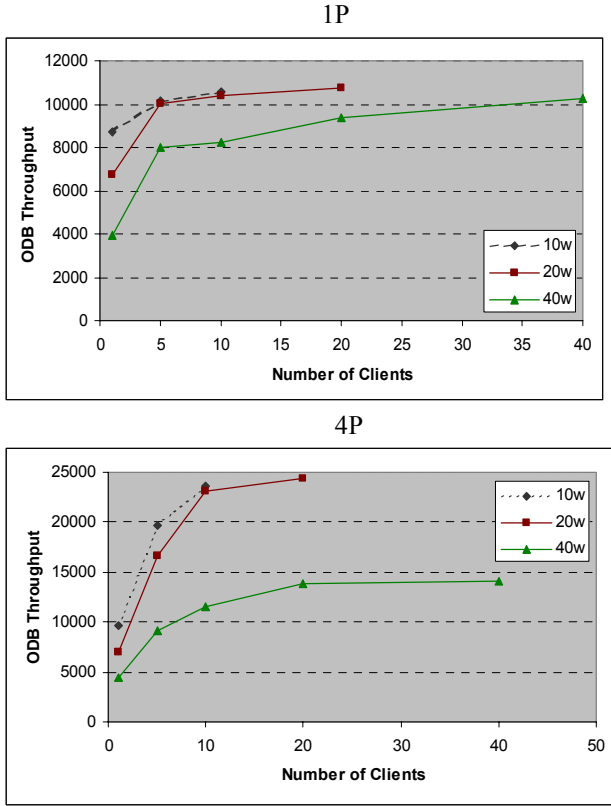


Figure 5: Transaction throughput with different warehouse sizes on 1P (top) and 4P (bottom) SMP systems.

Figure 5 illustrates how the transaction throughput scales relative to both the number of clients as well as the number of warehouses. The top figure is for the 1P system, and the bottom figure is for the 4P system. (In our scaling, we only scale the number of clients up to the number of warehouses.) We see that increasing the number of warehouses can lower the transaction throughput. While increasing from 10W to 20W, the drop is insignificant. Increasing from 20W to 40W produces a significant drop. For 10C, it is about 22% for the 1P system and 55% for the 4P system. Going from 10W to 20W, the database buffer cache is still effective at caching the database blocks. The number of misses to the database buffer cache is still tolerable because there is sufficient work available from the other clients to mask the disk I/O read latency. An interesting point to note is that the transaction throughput for 20W is higher than that of 10W on both 1P and 4P systems, even though the 20W is only partially cached. Because processor performance has improved dramatically, the commonly used 10W workload is not sufficient to saturate the processors. On the other hand, going to 40W significantly increases the working set and the database buffer cache size is not big enough to effectively cache the working set. This results in more disk I/O activities that are not masked and the I/O bandwidth starts to become a bottleneck. In going from 20W to 40W, we see the difference between cached and partially cached setups.

B. Impact of Processors and Disk I/O

The transaction throughput for a partially cached setup is much more sensitive to the disk I/O bandwidth. While adding more processors does improve performance, there is not sufficient amount of concurrency to hide latencies of the disk I/O reads completely. Figure 6 shows the transaction throughput for a 20W (top figure) and a 40W (bottom figure) database running on 1P, 2P, and 4P systems. As alluded to in the last section, the performance of 20W scales similarly to that of the 10W (see Figure 2). On the other hand for the 40W database, the performance improvement from increasing 1P to 2P is about 32% at the most, while the improvement from increasing 1 to 4 processors is about 48%. These improvements are much smaller than those seen for a cached setup. To approach comparable speedups, more disk I/O bandwidth is needed to improve partially cached setup. As shown in Figure 7 for a 4P, merely increasing the number of physical drives from 1 to 2 disks (1D to 2D) improves the performance by about 30% at 40C. When enough disks are added to completely mask the disk I/O read latency, then the partially cached setup becomes similar to a scaled setup. The key observation here is that increasing W increases the amount of I/O, which in turn puts more pressure on the I/O bandwidth. Hence, increasing D to match the increase of W is essential in alleviating the I/O bottleneck.

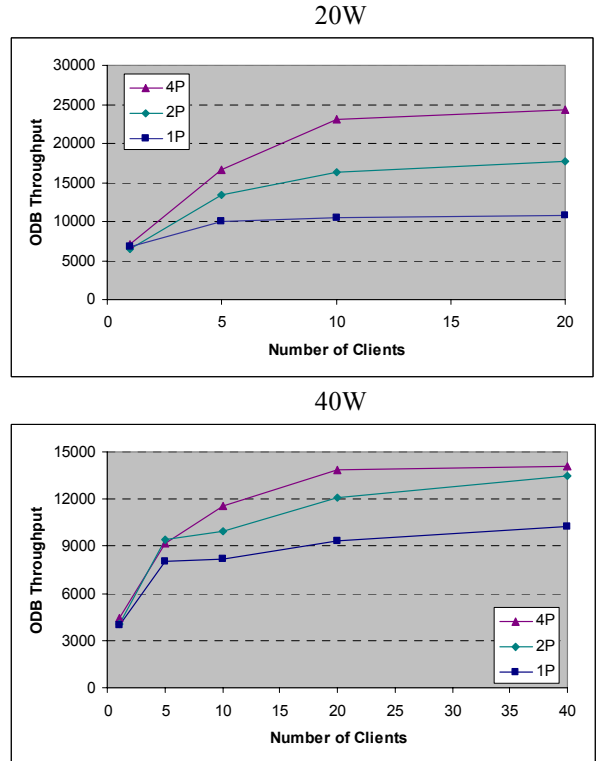


Figure 6: Transaction throughput for 20W (top) & 40W (bottom) databases on different number of processors.

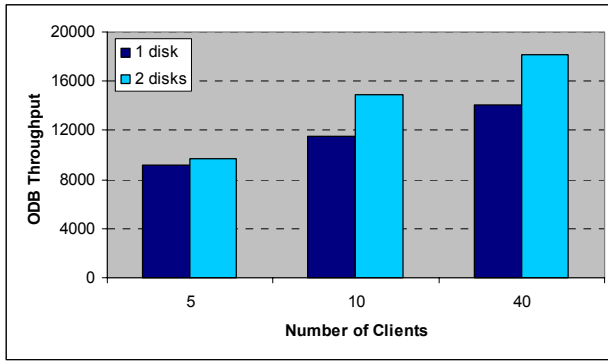


Figure 7: Transaction throughput for a 40W database using 1 or 2 disk drives.

V. THE COMMON DENOMINATOR: EXECUTION PROFILES

In the preceding sections, we covered a subspace of the OLTP performance space based on the four parameters: $\langle P, D, W, C \rangle$ where $P=\{1, 2, 4\}$, $D=\{1, 2\}$, $W=\{10, 20, 40\}$, and $C=\{1, 5, 10, 20, 40\}$. We plotted the ODB transaction throughputs for numerous points in this four dimensional space. Another parameter, the amount of memory allocated to the SGA, is fixed to be at its maximum value for a 32-bit virtual address space; i.e., 750K blocks. This section examines our experimental data from a slightly different perspective. In the previous sections, we vary the scaling parameters $\langle P, D, W, C \rangle$ and look at the resultant transaction throughputs. This section scans the range of transaction throughputs (x-axis) and analyzes the associated execution attributes (y-axis) of the workload.

One way of finding a relationship among the configuration parameters and the transaction throughput is to examine the percentage of the execution time spent on user code versus kernel code. Figure 8 shows two graphs, one for the user execution and the other for the kernel execution. A data point in each graph represents a particular $\langle P, D, W, C \rangle$ configuration and is plotted based on its transaction throughput (x-axis) and the corresponding user or kernel execution time (y-axis), measured in terms of percentage of total execution time. Each of the two figures contains many data points representing various combinations of P, D, W , and C , as well as three trend lines for 1P, 2P, and 4P. Each trend line represents the best-fit linear interpolation of the relevant data points. The R^2 values for the 1P, 2P, and 4P trend lines on the user execution graph are 0.95, 0.62, and 0.79, respectively. Some of the deviations can be attributed to the inherent sampling errors associated with the EMON data gathering.

We see in Figure 8 that kernel and user execution times exhibit radically different trends. The kernel execution time includes all CPU cycles that execute in the supervisory mode, and generally corresponds to the execution of OS kernel instructions and system calls. The rest of the other execution times are lumped together and are counted as the user execution time. The execution time does not include the actual DMA transfers made by the I/O devices. For MP systems, the

execution time is averaged across multiple processors. Across all configurations with different number of clients and warehouses, kernel execution times remain within a narrow band of 5% to 15%. Looking at the trend lines more carefully, kernel execution times do increase slightly with increasing throughput. This is due to more context-switching overhead from having more transactions completed. Nevertheless, based on this sensitivity analysis, the kernel execution time (in percentage of total execution time) grows rather slowly and that even for scaled setups, it is not unreasonable to expect it to remain within the 15% range.

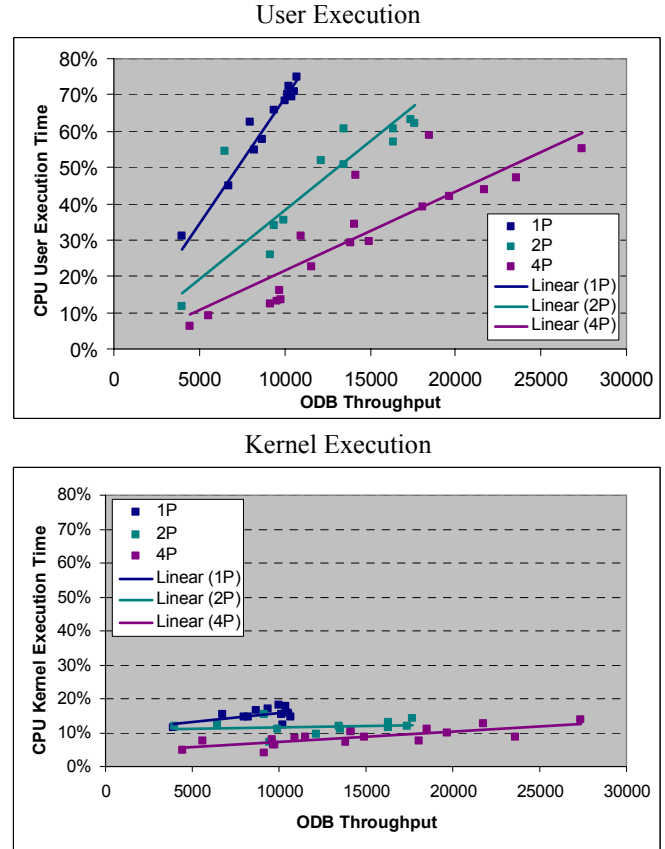


Figure 8: User execution (top) and kernel execution (bottom) profiles.

The user execution times exhibit rather different trends than the kernel execution times. We see in Figure 8 that as the transaction throughput increases, the user execution times increase proportionally. The user execution time is a good indicator of the overall utilization of the CPUs and of how balanced is the configuration. For example, the upper rightmost point of the 1P trend line represents the highest throughput measured on a 1P system. This configuration ($\langle 1P, 1D, 20W, 20C \rangle$) achieves a transaction throughput of about 10K and consumes about 75% of the available CPU utilization. With the kernel execution taking another 15% of the CPU utilization, the total CPU utilization is 90% indicating that the maximum possible throughput for an 1P system is at best about 10% higher.

As we move from 1P to 2P and 4P, the trend lines appear to be less steep. The reason is that the scaling is plotted on a per-processor basis. When factoring in the number of processors by multiplying the utilization with the number of processors, the linear scaling is similar to the 1P trend line but extends further out. We see that the CPU is less fully utilized for the 2P and 4P configurations. The highest user execution time is about 65% for 2P and 55% for 4P configurations, resulting in total CPU utilization of 80% and 70%, respectively. Extrapolating the 4P trend line to about 85% user execution time, for a total CPU utilization of 100%, would yield a transaction throughput of about 38K.

VI. CORRELATIONS BETWEEN CPI AND L3 MPI

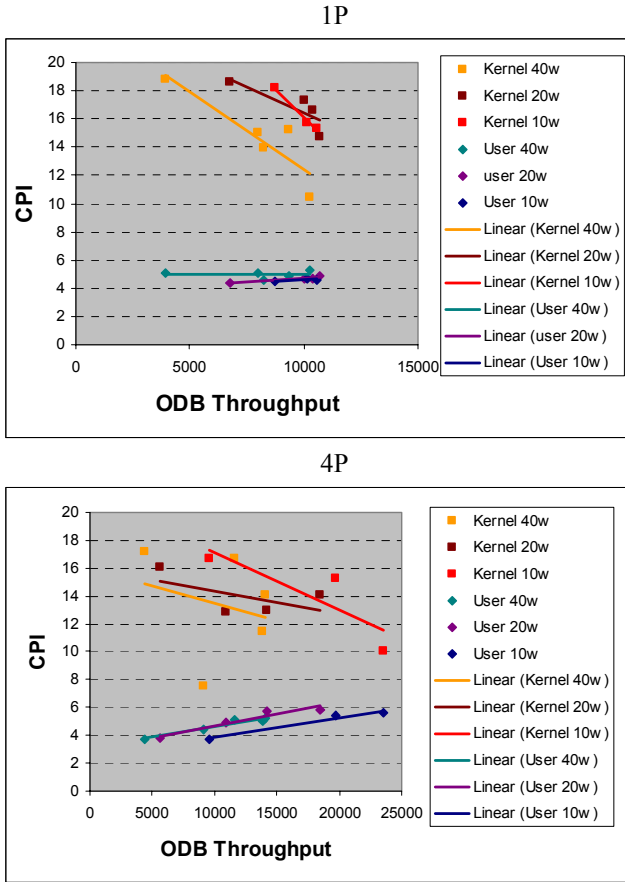


Figure 9: Correlation of CPI to throughput for 1P (top) and 4P (bottom) systems.

This section uses a similar approach of linear interpolation of trend lines to examine the average cycles per instruction (CPI) for the CPUs and the average misses per instruction (MPI) for the L3 cache. Figure 9 plots CPI versus transaction throughput for the 1P and the 4P systems in two separate figures. Each figure shows how the kernel CPI and the user CPI scale relative to transaction throughput. Separate trend lines are shown for 10W, 20W, and 40W configurations for both kernel and user CPI. The kernel CPIs, ranging from 10 to 20, are substantially higher than their corresponding user CPIs, which has a range of 4 to 8. There are several reasons to

account for this difference. OS code tends to be more branch intensive and have fewer instructions executed within a context. The combination of frequent context switches and the poor locality in OS code result in a relatively high cost of instruction execution for the OS kernel. The lower user CPIs indicate the CPUs are relatively more effective in executing user code.

An interesting observation of the trend lines indicates that as transaction throughput increases, the higher costs of kernel execution can be amortized to lead to lower kernel CPIs. For a specific warehouse configuration, the kernel CPIs decrease with higher transaction throughputs; i.e., each of the kernel CPI trend lines slopes downward. With more clients concurrently active, the likelihood of sharing resources for instructions and data between contexts increases and improves the spatial and temporal locality of the kernel code execution. In addition, the dynamic code paths in a higher transaction throughput become longer with more occurrences of spin locks as a result of more contention among the active processes. Therefore, this lowers the average cost of kernel execution as measured by CPI even though the total kernel execution time increases with more transaction throughput.

We see similar trending of the kernel CPI for both the 1P and 4P systems, but with some differences. First, because the transaction throughput levels for the 4P system are higher than those for the 1P system, the kernel CPIs are correspondingly lower due to the amortization and better locality. Second, the downward slope of the kernel CPI trend lines is less steep for the 4P system, and this is due to the inefficiency of having multiple dedicated caches. As shown in Figure 10, the L3 misses per instruction (MPI) trend lines correlate extremely well with the CPI trend lines. Comparing the results in Figure 9 and Figure 10, we can conclude that the kernel CPIs and the user CPIs are strongly dictated by their corresponding L3 MPIs. The steepness of the L3 MPI trend lines is greater for the 1P system than on the 4P system. This change in steepness is more apparent for the user MPI trend lines. On the 1P system, both the user CPI (Figure 9) and MPI (Figure 10) trend lines are relatively flat and are relatively independent of the number of warehouses and clients. For those same workload configurations running on a 4P system, both the user CPI (Figure 9) and MPI (Figure 10) trend lines actually increase with increasing transaction throughput. When there are more clients active at the same time, there are more opportunities for the processors to share and update the same cachelines. This behavior, which is common for semaphores and other synchronization variables, can result in many write invalidations as required to maintain coherence by the distributed caches. Another reason that the MPI increases in an 4P system is due to the data and process migration from one processor to another. Despite the increase in user CPIs, the net result of higher transaction throughput is made possible by the larger decrease in the corresponding kernel CPIs. We are conjecturing that the transaction throughput can be pushed higher by improving the user L3 cache misses on an MP

system. There appears to be an opportunity for CMP designers to improve on the cache organization on multiple processors.

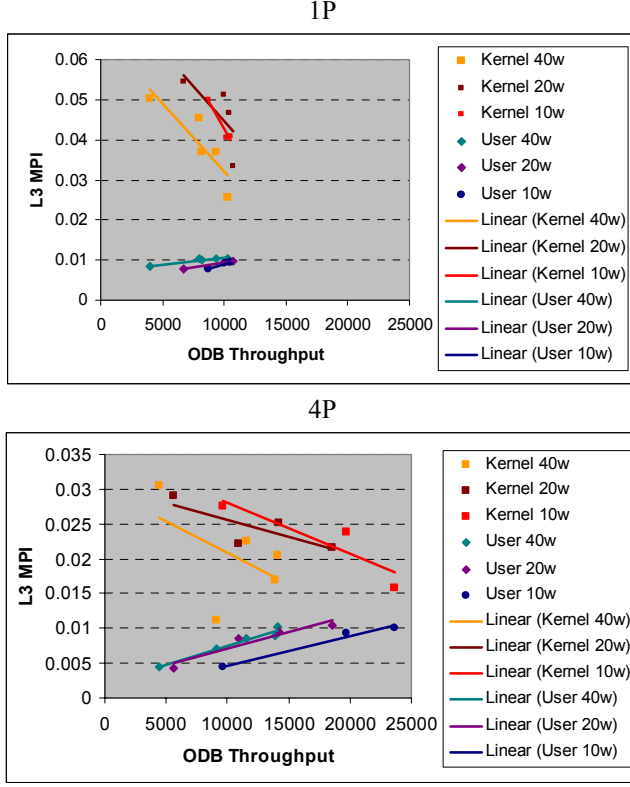


Figure 10: Correlation of L3 MPI to throughput for 1P (top) and 4P (bottom) systems.

VII. EMPIRICAL MODEL

In the preceding sections, linear interpolation is used to plot the trending seen for multiple $\langle P, D, W, C \rangle$ data points between their corresponding ODB throughputs and the underlying performance characteristics, be it the CPI or the MPI. Good correlation is observed by interpolating one variable at a time and by grouping them with respect to the number of processors and warehouses. Another use of linear interpolation is to apply across all $\langle P, D, W, C \rangle$ data points as a five-dimension space, where four axes correspond to the four input parameters and the fifth axis is the performance metric such as CPI, as shown in the following equation,

$$a \cdot \bar{P} + b \cdot \bar{D} + c \cdot \bar{W} + d \cdot \bar{C} = \bar{f}(),$$

where \bar{P} is a vector of processor parameters, \bar{D} is a vector of disk parameters, \bar{W} is a vector of warehouse parameters, \bar{C} is a vector of client parameters, and $\bar{f}()$ is a vector of CPIs. The size of each vector is equal to the total number of data points of the cross product $\langle P, D, W, C \rangle$ space, so that the matrix $[\bar{P}, \bar{D}, \bar{W}, \bar{C}]$ would represent the input parameters of the entire configuration space used. The resulting multivariate linear interpolation for CPI_{User} and CPI_{Kernel} has the following coefficients.

	$f=CPI_{User}$	$f=CPI_{Kernel}$
a	0.2165	0.0189
b	2.3340	8.9799
c	0.0431	0.1694
d	0.0212	-0.1157

The linear interpolation equation above is a very crude empirical model for characterizing ODB performance based on the four configuration parameters of P, D, W, and C. The coefficients, a, b, c, and d, indicate the relative sensitivity of the resultant CPI performance to the four configuration parameters of P, D, W and C, respectively. A number of speculative observations can be made based on the coefficients of this model. For both kernel CPI and user CPI, the coefficient b has the largest value among the four coefficients, indicating that the CPI is strongly influenced by the parameter D, the number of disks. Comparing the coefficients for kernel CPI vs. user CPI, we see that kernel coefficients tend to be higher due to the higher CPI of kernel execution. The one exception is that the coefficient a for the kernel CPI is significantly less than for the user CPI. This is an indication that user execution is more strongly correlated to the parameter P, the number of CPUs, because user execution constitute a larger percentage of the total execution as compared to kernel execution.

VIII. SUMMARY AND CONCLUSIONS

Cached setups, which scale down the database size to tens of warehouses, have working sets that fit well in system memory. The reduction in scale, which makes analyzing and simulating OLTP workloads more tractable, is one reason why cached setups are widely used in the research community. This paper is a quantitative analysis of the configuration space of the cached and partially cached OLTP workloads by exploring four parameters: number of processors (P), number of disks (D), number of warehouses (W), and number of concurrent client processes (C). Our goal is to study the effects of scaling $\langle P, D, W, C \rangle$ on performance and to formulate empirical relationship of workload behavior among the four parameters. In this paper, we show that to increase overall transaction throughput, there must be enough clients to generate the work to be done, enough CPU resources to process the transactions, and enough warehouses to avoid internal database contention. With current processor speeds, the 10-warehouse workload is found to be not sufficient to saturate current processors. However, adding more warehouses increases more disk I/O activities and requires more disks to alleviate the I/O bottleneck. When the balance between more concurrency with more warehouses and sufficient clients to tolerate the disk I/O read latency is achieved, we show that a partially cached setup of twenty warehouses can produce a higher transaction throughput than a cached setup of ten warehouses. By employing linear interpolation to produce trend lines across multiple $\langle P, D, W, C \rangle$ data points, we also show how kernel and user execution times exhibit radically different trends relative to the transaction throughput. The kernel code has much worse instruction-level parallelism, but kernel CPIs

decrease with higher transaction throughput. On the other hand, in an MP system, the user CPIs actually increase with higher transaction throughput and is directly correlated to the increasing user MPIs. This appears to be an opportunity for CMP designers to improve on the multiprocessor cache organization to push the transaction throughput higher.

REFERENCES

- [1] Z. Cvetanovic and D. Bhandarkar. Characterization of Alpha-Axp Performance using TP and SPEC Workloads. In Proceedings of the 21st Annual International Symposium on Computer Architecture, pages 60–70, April 1994.
- [2] M. Franklin, W.P. Alexander, R. Jauhari, A.M.G. Maynard, B.R. Olszewski. Commercial Workload Performance in the IBM Power2 Risc System/6000 Processor. IBM J. of Research and Development, 38(5): 555–561, 1994.
- [3] A. Ailamaki, D. DeWitt, M. Hill, and D. Wood. DBMSs on a Modern Processor: Where Does Time Go? In Proceedings of the 25th International Conference on Very Large Data Bases, pages 266–277, September 1999.
- [4] K. Keeton, D.A. Patterson, Y.Q. He, R.C. Raphael, and W.E. Baker. Performance Characterization of a Quad Pentium Pro SMP Using OLTP Workloads. In Proceedings of the 25th International Symposium on Computer Architecture, pages 15–26, June 1998.
- [5] L.A. Barroso, K. Gharachorloo, and E. Bugnion. Memory System Characterization of Commercial Workloads. In Proceedings of the 25th International Symposium on Computer Architecture, pages 3–14, June 1998.
- [6] P. Ranganathan and K. Gharachorloo and S.V. Adve and L.A. Barroso. Performance of Database Workloads on Shared-Memory Systems with Out-of-Order Processors. In Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems, pages 307–318, October 1998.
- [7] M. Rosenblum, E. Bugnion, S. Herrod, E. Witchel, A. Gupta. The Impact of Architectural Trends on Operating System Performance. In Proceedings of the 15th ACM Symposium on Operating Systems Principles, pages 285–298, December 1995.
- [8] P.S. Magnusson, F. Dahlgren, H. Grahm, M. Karlsson, F. Larsson, F. Lundholm, A. Moestedt, J. Nilsson, P. Stenström, B. Werner. SimICS/sun4m: A Virtual Workstation. In Proceedings of the Usenix Annual Technical Conference, June 15-18, 1998.
- [9] M. Annavaram, J.M. Patel and E.S. Davidson. Call Graph Prefetching for Database Application. In Proceedings of the 7th International Symposium on High-Performance Computer Architecture, pages 281–290, January 2001.
- [10] M. Annavaram, T. Diep and J.P. Shen. Branch Behavior of a Commercial OLTP Workload on Intel IA32 Processors. In Proceedings of the International Conference on Computer Design, pages 242–248, September 2002.
- [11] http://www.tpc.org/tpcc/results/tpcc_result_detail.asp?id=101091903
- [12] K. Keeton, D.A. Patterson. The impact of Hardware and Software Configuration on Computer Architecture Performance Evaluation. In Workshop on Computer Architecture Evaluation using Commercial Workloads, Feb 1998.
- [13] D. Marr. Intel Netburst Microarchitecture and Hyper-Threading Technology. In Proceedings of the Hot Chips 14 Symposium, Aug 2002.
- [14] The IA-32 Intel® Architecture Software Developer's Manual, Volume 3: System Programming Guide.