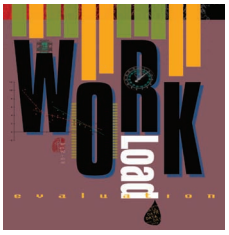
Simulating a $2M Commercial Server on a $2K PC



The Wisconsin Commercial Workload Suite contains scaled and tuned benchmarks for multiprocessor servers, enabling full-system simulations to run on the PCs that are routinely available to researchers.

Alaa R. Alameldeen
Milo M.K. Martin
Carl J. Mauer
Kevin E. Moore
Min Xu
Mark D. Hill
David A. Wood
University of Wisconsin-Madison

Daniel J. Sorin
Duke University

The Internet has made database management systems and Web servers integral parts of today's business and communications infrastructure. These and other commercial transaction-processing applications work with critical personal and business data—storing it, providing access to it, and manipulating it. As dependence on these applications increases, so does the need for them to run reliably and efficiently. Our group at the University of Wisconsin (www.cs.wisc.edu/multifacet/) researches innovative ways to improve the performance of the multiprocessor servers that run these important commercial applications.

Execution-driven simulation is a design evaluation tool that models system hardware. These simulations capture actual program behavior and detailed system interactions. They are more flexible and less expensive than hardware prototypes, and they model important system details more accurately than analytic modeling does. However, the combination of large systems and demanding workloads is difficult to simulate, especially on the inexpensive machines available to most researchers. Commercial workloads, unlike simpler workloads, rely heavily on operating system services such as input/output, process scheduling, and interprocess communication. To run commercial workloads correctly, simulators must model these services. In addition, multiprocessor servers introduce the challenges of interactions among processors, large main memories, and many disks.

To make effective use of limited simulation resources, researchers must balance three goals:

- developing a representative approximation of large workloads,
- achieving tractable simulation times, and
- simulating a sufficient level of timing detail.

We developed a simulation methodology to achieve these goals. Our methodology uses multiple simulations, pays careful attention to scaling effects on workload behavior, and extends VirtutechAB's Simics full-system functional simulator[1] with detailed timing models.

WORKLOAD SCALING AND TUNING

Our workloads currently consist of four benchmarks, described in the sidebar, "Wisconsin Commercial Workload Suite." These benchmarks approximate four important commercial application classes: online transaction processing (OLTP), Java middleware, static Web serving, and dynamic Web serving.

We scaled the application workloads down in both size and runtime, allowing our host machines to simulate the much more powerful servers that run commercial workloads. In our case, the hosts were PCs, each having 1 Gbyte of RAM, a single disk, and a 32-bit virtual address space.

To discover and remove performance bottlenecks, we tuned all of our workload setups on a real multiprocessor server. We found commercial workloads to be very sensitive to tuning. For example, tuning our OLTP workload improved its performance by a factor of 12.

Published by the IEEE Computer Society

Wisconsin Commercial Workload Suite

We developed the Wisconsin Commercial Workload Suite to support full-system simulation of multimillion-dollar servers on PCs. The current suite includes four benchmark applications.

OLTP: DB2 with a TPC-C workload

The TPC-C is a benchmark widely used to evaluate performance for the online transaction-processing market. It specifies the schema, scaling rules, transaction types, and mix—but not the implementation—of an order-processing database. It measures performance by the number of "new order" transactions performed per minute, subject to certain constraints.

Our OLTP workload is based on the TPC-C v3.0 benchmark. We used IBM's DB2 V7.2 EEE database management system with an IBM benchmark kit to build the database and emulate users.

The database size is 800 Mbytes, partitioned over five raw disks, with an additional dedicated database log disk. It represents data for 4,000 warehouses but scales down the warehouse sizes specified in TPC-C from 10 sales districts per warehouse to three, from 30,000 customers per district to 30, and from 100,000 items per warehouse to 100.

Each user randomly executes transactions according to the TPC-C transaction mix specifications. User think times and keying times are set to zero. Each user starts a different database thread. The benchmark measures all completed transactions, even those that do not satisfy timing constraints of the TPC-C specification.

Java server workload: SPECjbb

SPECjbb is a Java benchmark for emulating a three-tier system with emphasis on the middle-tier server business logic. SPECjbb does no disk or network I/O. It runs in a single Java virtual machine in which threads represent warehouse terminals. Each thread independently generates random input (tier-1 emulation) before calling transaction-specific business logic. The business logic operates on the data held in binary trees of Java objects (tier-3 emulation).

We used Sun's HotSpot 1.4.0 Server JVM with Solaris's native thread implementation. The benchmark includes driver threads to generate transactions. We set the system heap size to 1.8 Gbytes and the new object heap size to 256 Mbytes to reduce the frequency of garbage collection. Our experiments used 24 warehouses, with a data size of approximately 500 Mbytes.

Static Web content serving: Apache

Apache is a popular open source Web server employed in many enterprise server applications. In a benchmark focused on static Web content serving, we used Apache 2.0.39 for Sparc/Solaris 8 and configured to use pthread locks and minimal logging at the Web server.

For the client, we used the Scalable URL Request Generator.[1] Surge generates a sequence of static URL requests that exhibit representative distributions for document popularity, document sizes, request sizes, temporal and spatial locality, and embedded document count. We have a repository of 20,000 files, totaling about 500 Mbytes, and use clients with zero think time.

We compiled both Apache and Surge using Sun's WorkShop C 6.1 with aggressive optimization.

Dynamic Web content serving: Slashcode

Dynamic content serving has become increasingly important for Web sites that serve large amounts of information. For example, online stores, instant news feeds, and community message boards all serve dynamic content.

Slashcode is an open-source dynamic Web message-posting system used by the popular slashdot.org message board. We implemented our benchmark using Slashcode 2.0, Apache 1.3.20, and Apache's mod_perl module 1.25 (with Perl 5.6) on the server side. MySQL 3.23.39 is the database engine. The server content is a snapshot from the slashcode.com site, containing approximately 3,000 messages with a total size of 5 Mbytes.

The benchmark application spends most of its runtime on dynamic Web page generation. A multithreaded user-emulation program models browsing and posting behavior. Each user independently and randomly generates browsing and posting requests to the server according to a transaction-mix specification. We compiled both server and client programs using Sun's WorkShop C 6.1 with aggressive optimization.

Reference

1. P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," *Proc. 1998 ACM Sigmetrics Conf. Measurement and Modeling of Computer Systems*, ACM Press, 1998, pp. 151-160.

OLTP case study

Online transaction-processing systems form the core of the business computing infrastructure in industries such as banking, airline reservations, and online stores. In large businesses, OLTP systems often process hundreds of thousands of transactions per minute. The multiprocessor systems or system clusters that support this high throughput cost millions of dollars.

The primary benchmark to compare OLTP system performance is the Transaction Processing Performance Council's TPC-C benchmark (www.tpc.org/tpcc/). Published TPC-C results reveal the large scale of many commercial workloads. For example, a current noncluster TPC-C performance leader is a database server with 128 processors (with 8 Mbytes of cache each), 256 Gbytes of RAM, and 29 Tbytes of disk storage on 1,627 disks. The clients emulate nearly 400,000 users placing orders at about 40,000 warehouses. The system completed over 100 million transactions during the 25-minute warm-up and two-hour measurement periods. Its total hardware and software cost was more than $13 million.

Millions of dollars may be a reasonable investment for a computer system that runs a major company's core business application, but it is unrealistic for a single research group. Our objective was to

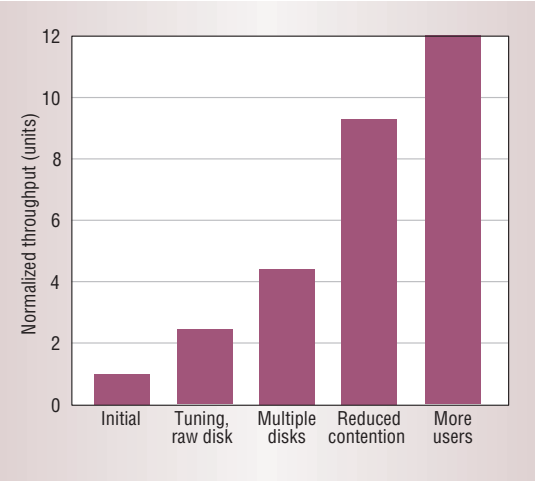


Figure 1. Transaction throughput of our OLTP workload after each tuning step, normalized to our initial setup. The final configuration has a throughput 12 times that of our original setup.

develop a workload that captures the important characteristics of real-world OLTP systems, while remaining small enough to use in our simulations.

We started with the TPC-C benchmark specification, IBM's DB2 database management system, and an IBM TPC-C benchmark kit. We set up, scaled down, and tuned the workload on an actual multiprocessor—a Sun E5000 with 12 167-MHz processors and 2 Gbytes of memory. Then we moved exact disk images of the workload into our simulation environment. Using a real machine allows long measurement intervals, makes benchmark setup and tuning much faster compared with simulation, and provides data that we can use to validate simulation results.

Initial scaling. Ideally, we would like to evaluate multiprocessor systems with large numbers of disks and large database sizes. However, we cannot simulate such systems within our current infrastructure. Therefore, we reduced the database size to 1 Gbyte to reduce the amount of disk read traffic. This reduction also allowed the database to fit in the memory of our real system and the simulation target machine. Although this scaling could affect our results, it was necessary to enable simulation on inexpensive PCs.

TPC-C models the database activity of a wholesale supplier with several geographically distributed sales districts and associated warehouses. The TPC-C specifications state that database size should be set by the number of warehouses, keeping the relative sizes of the other tables the same. Since the approximate size of all data associated with one warehouse is 100 Mbytes, we created a 10-warehouse database on a single disk (plus an additional log disk). However, this setup measured a much lower throughput in terms of transactions per minute than expected from similar systems.

Raw device access and other parameter tuning. Next, we tuned several kernel and database configuration parameters, such as kernel limits on the number of shared-memory segments and semaphores and database limits on the threads and locks. We also reconstructed the database on a raw database-managed disk, since using normal operating system files for database tables increases database overhead and results in data buffering in both the operating system file cache and the database buffer pool (effectively doubling memory usage).

Taken together, these changes improved performance by 144 percent.

Multiple disks. Subsequent analysis using operating system profiling tools showed that the database disk was now the likely bottleneck. Although the data was sized to fit in the system's main memory, the frequent updates in TPC-C caused substantial disk write traffic. To alleviate this problem, we decided to partition the 1-Gbyte database across five raw disks.

This change removed the I/O bottleneck, further improving performance by 81 percent.

Table contention reduction. Although we had dramatically increased performance, the operating system profiling tools still showed a large amount of system idle time. We discovered that, due to the small number of warehouses in the database, the system was serializing transactions that read and write the same entries in the "warehouse" table, limiting system throughput.

To eliminate this bottleneck, we deviated from the standard TPC-C scaling requirements by increasing the number of warehouses without increasing the total database size.

This change resulted in an 800-Mbyte database with 4,000 warehouses and improved performance by another 111 percent.

Additional concurrency. Our initial setup used 24 client emulators, which meant that we had, on average, two database threads running on each of our 12 processors. Although we eliminated think time and keying time for the emulated clients to reduce client overheads, there was not enough concurrency in the system to hide the I/O latency.

Increasing the number of emulated user threads to eight per processor (96 total) provided an additional 29 percent improvement.

OLTP performance improvement

Figure 1 plots the normalized throughput of our OLTP workload at each tuning stage. The remarkable performance improvement shows that tuning commercial workloads is essential for obtaining representative workloads.

The throughput of our tuned OLTP workload is close to published TPC-C results for similar hardware. More importantly, the tuning makes our workload far more representative of a real OLTP system.

WORKLOAD RUNTIME AND VARIABILITY

Simulation is orders of magnitude slower than real system execution. However, benchmarking commercial workloads often involves running long warm-up and measurement intervals to avoid cold-start and transient effects. The combination of these factors presents a challenge for commercial workload simulation. For example, we observed a slowdown factor of approximately 24,000 when simulating a 16-processor system with our detailed timing model. At this rate, simulating the two-hour minimum measurement interval required for the TPC-C benchmark would take more than five years, and simulating even one minute would take weeks.

To make evaluating commercial workloads practical, we scaled down these long intervals and developed an economical methodology for dealing with cold-start and transient effects. Our methodology consists of three parts. First, we avoid the warm-up overhead by starting from an already warm system state. Second, we sample a small portion of the workload by counting transactions. Third, we handle the variability due to measuring short intervals by averaging measurements from multiple runs.

Starting with warm workloads

To avoid the need to simulate the startup phases of our commercial workloads, such as the creation of database processes, we use Simics's ability to save a checkpoint (snapshot) of the simulated system's architected state. Checkpoints include the state of all processors, memory, devices, and disks. Using this technique, we can simulate a reasonable warm-up period and create a checkpoint that reflects a warm system's state. Starting our timing simulations with a warm system reduces the simulation time required and also mitigates cold-start effects.

Fixed-transaction-count simulation methodology

Since we cannot simulate benchmarks from start to finish, we must limit the length of our measurement interval to keep simulation time reasonable. A standard approach to measure performance on partial benchmark runs for simpler workloads (for example, the SPECcpu2000 benchmarks) is to record the number of cycles required to execute a fixed number of instructions. The resulting metric, *instructions per cycle*, corresponds exactly to performance for user-mode, single-threaded, uniprocessor simulations.

Unfortunately, IPC does not correspond to throughput on multiprocessors. For example, if an application spends more time waiting in the operating system's idle loop or in a loop to acquire a lock, the IPC can actually improve while throughput decreases. Therefore, applying this approach to multiprocessor commercial workloads is inappropriate and can lead to incorrect conclusions about workload performance.

Instead, we measure the time required to finish a certain number of a benchmark's transactions.[2] We use the number of cycles per transaction as an inverse-throughput metric to compare the throughput of different configurations.

Variability in many commercial workloads is large enough to affect research conclusions.

Since our commercial workloads are all throughput-oriented, they included the transaction (or request) concept. We modified the transaction generators for each workload, using a special instruction with no side effects, to alert the simulator whenever a transaction completes. During simulation measurement experiments, the simulator counts the number of transactions completed and stops when it reaches the desired count.

Variability of short simulations

Because we are practically limited to short simulation runs, the measured workload throughput becomes more dependent on the workload's execution path—the exact instruction sequence executed during the simulation. Execution paths can differ due to different orders of thread interleaving caused by operating system scheduling decisions or different orders of lock acquisition.

This dependency increases the effect of an important phenomenon for short simulation runs, namely *variability in workload timing results*. Variability refers to the differences between multiple estimates of a workload's performance, and it exists in both real and simulated systems.[3] Researchers must account for variability when they evaluate architectural innovations by comparing the performance of enhanced designs relative to a base configuration. Otherwise, they might attribute a performance difference caused by workload variability to a real difference in the relative performance of the enhanced and base systems.

Variability in many commercial workloads is large enough to affect research conclusions. Figure 2 demonstrates this variability's magnitude on our real system. When the observation interval is short (one second, or the equivalent of seven hours of simulation), we observe significantly different throughputs for different OLTP execution instances. Although simulation experiments are deterministic and will always follow the same execution path for the same

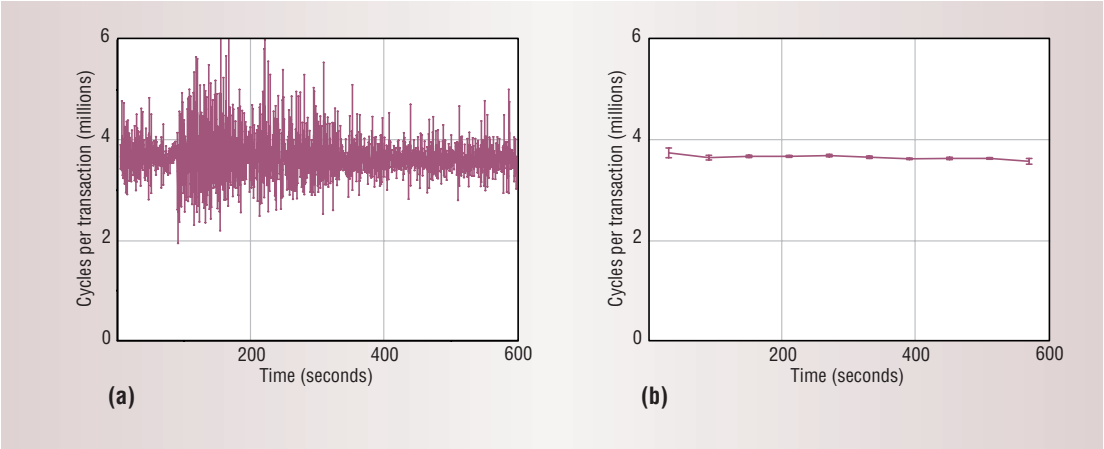


Figure 2. Cycles per transaction of five OLTP runs for different observation intervals over 10 minutes total. (a) An observation interval of 1 second shows large fluctuations in throughput, even though the benchmark completed more than 350 transactions per second on average. (b) An observation interval of 1 minute greatly diminishes the variability.

workload and system configuration, we do not know whether that deterministic path would provide an advantage or disadvantage for a particular configuration. This effect makes multiple configurations difficult to compare in simulation, since a single execution path might not represent all possible executions.

To solve this problem, we obtain multiple performance estimates by introducing an artificial source of variability, adding a small random delay to each memory access. The average memory latency is the same for all simulations, but each simulation will follow a different execution path by using different random seeds.

To illustrate the risk of using single simulation runs, Figure 3 shows simulation results for two different system configurations: two-way versus four-way set-associative L2 caches. Each run corresponds to approximately 0.2 seconds on the target machine, which is faster than the real system used in Figure 2. Each data point represents the number of cycles per transaction of one execution path.

Figure 3 demonstrates that different execution paths happen even for the same workload and system configuration. The average performance for all 20 runs confirms the intuitive conclusion that OLTP performs better on the four-way set-associative L2 cache configuration. However, if we performed a single experiment for each configuration, we might conclude that the two-way set-associative configuration performs better—for example, when comparing the minimum runtime of the two-way configuration with the maximum of the four-way configuration. If we randomly select one run from each configuration, there is a 31 percent chance of drawing the wrong conclusion.

This experiment shows that we cannot rely upon a single short simulation run to obtain correct conclusions in comparison experiments. We handle variability by using multiple simulations for each configuration. We use the average simulated *cycles per transaction* to represent the workload's performance, and we use the standard deviation to establish confidence intervals.

This approach greatly reduces the probability of reaching a wrong conclusion compared with single-run experiments, at the expense of increased total simulation runtime. However, if multiple simulation hosts are available, as at Wisconsin, running multiple short simulations in parallel is greatly preferable to running one long simulation.

We also developed a more sophisticated statistical methodology that helps achieve reasonable simulation time limits while, at the same time, reducing the probability of reaching a wrong research conclusion.[3]

TIMING SIMULATION OF COMMERCIAL WORKLOADS

Simics is a functional simulator that can execute unmodified operating systems, such as Solaris, but it does not accurately model timing of any particular system. To evaluate the performance of systems that run commercial workloads, we extended Simics with two timing models:

- a memory simulator that implements a two-level cache hierarchy, cache controllers, an interconnection network, and optional directory controllers; and
- a detailed processor timing simulator that implements an out-of-order processor executing the Sparc V9 instruction set.

To reduce complexity, our timing models approximate some details of target systems while still trying to capture those aspects that significantly affect system timing. For example, the memory system simulator models the states and transitions for different cache coherence protocols as well as the various latencies and bandwidth limitations of caches, memories, and interconnection network links. However, it uses approximate models for DRAM, disks, I/O timing, and references to memory-mapped I/O registers.

The detailed processor timing simulator uses *timing-first simulation*,[4] a decoupled technique implemented in two simulators: a timing simulator

augmented to functionally execute the instructions most important to performance, followed by a full-system functional simulator to ensure correctness. This approach allows the timing simulator to skip instructions that are unimportant to timing fidelity without introducing functional errors in the system's simulation. The timing simulator controls when each processor in the functional simulator can advance. When an instruction retires, the timing simulator steps the appropriate processor, then verifies the results of its functional execution. By advancing one processor before another, the timing simulator can determine the winner in a race to memory, but it does not modify the functional simulator's actual state.

While this approximation technique introduces a timing error, it does not significantly affect overall system timing. The error is proportional to the instructions that do not match between the timing and functional simulators, which are only 0.003 percent of all instructions on average for our commercial workloads.

These approximations increase the importance of validating our simulation results by comparing them with system measurements. Validation remains an important but difficult component of our simulation efforts—one that is far from complete at present. However, even though a validated cycle-accurate simulator is necessary for an absolute performance prediction, it is not necessary for most architecture studies. Validation efforts should focus on the intended use of the workload and simulator, and we think that our current models are sufficient for our applications.

CASE STUDY: CACHE COHERENCE PROTOCOL

Computer architecture research frequently compares performance between a base system and an enhancement of it. However, a design decision that increases performance for one benchmark may have the opposite effect on another. This is why computer architects should evaluate their ideas with the most relevant workloads. Our work to evaluate cache coherence protocols for commercial workloads illustrates this point.

Multiprocessor server architects must choose a cache coherence protocol to coordinate reads and writes to a memory location. A tradeoff between cache-miss latency and system-interconnect bandwidth is at the heart of this decision.

There are two major categories of cache coherence protocols: directory-based and snooping. Systems that use *directory-based protocols* suffer from long latencies for cache-to-cache transfers between processors' caches, since each data request goes first to the directory, which then forwards it to a processor that can provide data. Systems based on *snooping protocols* reduce the latency of cache-to-cache transfers, since each processor broadcasts all its requests to all processors, which allows requests to find the data provider directly. Unfortunately, broadcasting generates significant traffic on the system interconnect, especially for systems with a large number of processors.

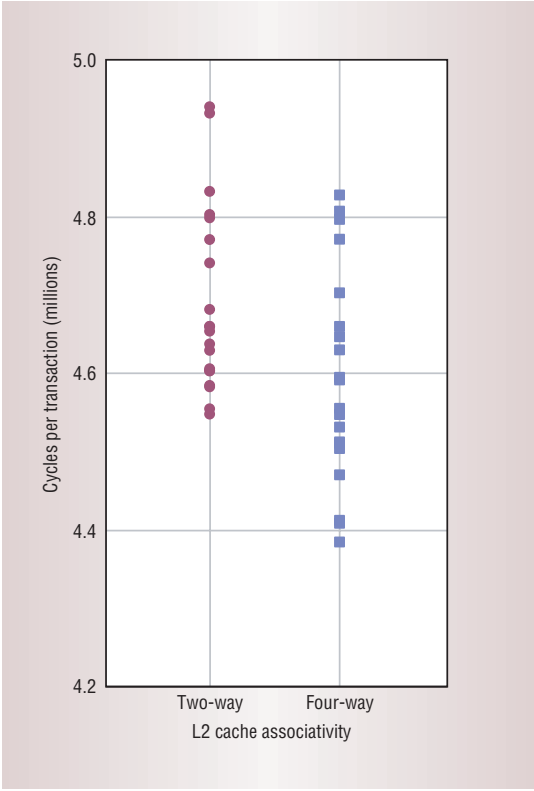


Figure 3. Cycles per transaction for 20 OLTP simulations of 200 transactions on two 16-processor systems that differ only in L2 cache associativity (two-way versus four-way). Each run corresponds to approximately 0.2 seconds on the target machine. Each data point represents the number of cycles per transaction of one execution path.

Our workload characterizations, as well as others,[5] show that cache-to-cache transfers are prominent in commercial workloads and have a significant adverse effect on performance. We explored the performance of both protocol categories on our four commercial workloads and on a scientific benchmark. Experiments with 16 processors and a moderate amount of system interconnection bandwidth showed that a directory protocol outperforms a snooping protocol for some workloads, but the converse is true for others.

Motivated by this result, we developed a hybrid protocol, called Bandwidth Adaptive Snooping Hybrid.[6] BASH acts like a snooping protocol if sufficient bandwidth is available, but gracefully degrades to act like a bandwidth-efficient directory protocol when bandwidth is scarce. The system monitors the interconnect utilization and adjusts the rate of broadcast requests accordingly. It decreases the broadcast rate if the interconnect utilization is too high (to avoid congestion delays) and increases it if utilization is too low (to reduce latency by broadcasting).

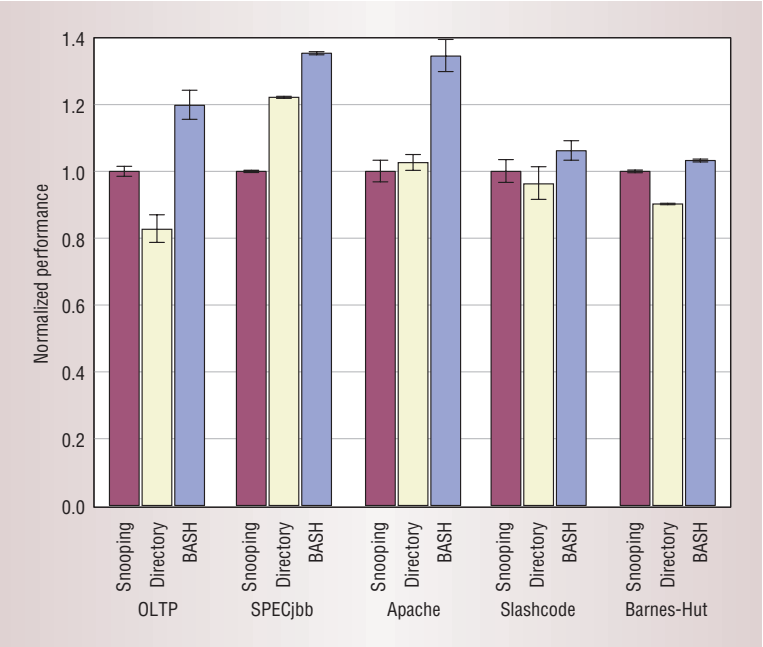


Figure 4. System performance with snooping, directory, and the Bandwidth Adaptive Snooping Hybrid protocols for four commercial workloads and one scientific application. The height of a bar represents the average normalized performance: Bigger is better. The error bars show the standard deviation.

Figure 4 compares results from using the snooping, directory, and hybrid protocols. It shows that BASH performs equally well or outperforms the better of snooping or directory systems for all our workloads. The benefit is significantly greater for our commercial workloads, compared with the Barnes-Hut scientific benchmark from the Splash-2 benchmark suite[7] because of their higher frequency of cache-to-cache transfers. Although BASH performs well for Barnes-Hut, the difference is not compelling. On the other hand, the substantial per- formance improvements for commercial workloads make BASH an attractive alternative for future multiprocessor server designs.

As the BASH case study shows, the outcome of computer architecture experiments depends greatly on the workloads used for evaluation. The Wisconsin Commercial Workload Suite successfully approximates the behavior of commercial server workloads in a PC environment, thus supporting further research in multiprocessor servers in a university research setting. We plan to continue expanding the workload suite by developing additional middle-tier benchmarks. We are working with Virtutech AB to make simulation checkpoints of our workloads available to the research community. ■

Acknowledgments

This work is supported in part by the US National Science Foundation (EIA-9971256, EIA-0205286, CDA-9623632, and CCR-0105721), an Intel Graduate Fellowship (Sorin), an IBM Graduate Fellowship (Martin), a Norm Koo/Sun Microsystems Fellowship (Martin), two Wisconsin Romnes Fellowships (Hill and Wood), Universitat Politècnia de Catalunya y Secretaría Estado de Educación y Universidades de España (Hill sabbatical), and donations from Intel Corp., IBM, and Sun Microsystems.

References

1. P.S. Magnusson et al., "Simics: A Full System Simulation Platform," *Computer*, Feb. 2002, pp. 50-58.
2. A.R. Alameldeen et al., "Evaluating Nondeterministic Multithreaded Commercial Workloads, *Proc. 5th Workshop Computer Architecture Evaluation Using Commercial Workloads*, Int'l Symp. High-Performance Computer Architecture, 2002; www.hpcaconf.org/hpca8/caecw02.pdf.
3. A.R. Alameldeen and D.A. Wood, "Variability in Architectural Simulations of Multithreaded Workloads," to appear in *Proc. 9th IEEE Symp. High-Performance Computer Architecture*, IEEE CS Press, 2003.
4. C.J. Mauer, M.D. Hill, and D.A. Wood, "Full-System Timing-First Simulation," *Proc. 2002 ACM Sigmetrics Conf. Measurement and Modeling of Computer Systems*, ACM Press, 2002, pp. 108-116.
5. L.A. Barroso, K. Gharachorloo, and E. Bugnion, "Memory System Characterization of Commercial Workloads," *Proc. 25th Ann. Int'l Symp. Computer Architecture*, IEEE CS Press, 1998, pp. 3-14.
6. M.M.K. Martin et al., "Bandwidth Adaptive Snooping," *Proc. 8th IEEE Symp. High-Performance Computer Architecture*, IEEE CS Press, 2002, pp. 251-262.
7. S.C. Woo et al., "The SPLASH-2 Programs: Characterization and Methodological Considerations," *Proc. 22nd Ann. Int'l Symp. Computer Architecture*, ACM Press, 1995, pp. 24-37.

Alaa R. Alameldeen *is a graduate student in the Computer Sciences Department at the University of Wisconsin-Madison. His research interests include multiprocessor system and memory design and performance evaluation of multithreaded workloads. Alameldeen received an MS from Alexandria University, Egypt, and an MS from the University of Wisconsin-Madison, both in computer science. He is a student member of the IEEE and the ACM. Contact him at alaa@cs.wisc.edu.*

Milo M.K. Martin *is a PhD candidate in the Computer Sciences Department at the University of Wisconsin-Madison. His research interests include memory system performance of commercial workloads and the use of dynamic feedback to build adaptive and robust systems. Martin received an MS in computer science from the University of Wisconsin-Madison. He is a student member of the*

IEEE and the ACM. Contact him at milo@cs.wisc.edu.

***Carl J. Mauer** is a graduate student at the University of Wisconsin-Madison. His research interests include multiprocessor architectural simulation and memory system design. Mauer received an MS in computer sciences from the University of Wisconsin-Madison. Contact him at cmauer@cs.wisc.edu.*

***Kevin E. Moore** is a graduate student in the Computer Sciences Department at the University of Wisconsin-Madison. His research interests include multiprocessor memory system design and performance evaluation of Java workloads. Moore received an MS in computer science from the University of Wisconsin-Madison. Contact him at kmoore@cs.wisc.edu.*

***Min Xu** is a PhD student at the University of Wisconsin-Madison. His research focuses on multiprocessor memory system performance and multiprocessor programmability. Xu received an MSEE in electrical and computer engineering from the University of Wisconsin-Madison. Contact him at mxu@cae. wisc.edu.*

***Mark D. Hill** is a professor and Romnes Fellow in both the Computer Sciences Department and the Electrical and Computer Engineering Department at the University of Wisconsin-Madison. He also co-directs the Wisconsin Multifacet project to improve commercial servers. Hill received a PhD from the University of California, Berkeley. He is an IEEE Fellow. Contact him at markhill@cs.wisc.edu.*

***David A. Wood** is a professor and Romnes Fellow in both the Computer Sciences Department and the Electrical and Computer Engineering Department at the University of Wisconsin-Madison. He also codirects the Wisconsin Multifacet project to improve commercial servers. Wood received a PhD in computer science from the University of California, Berkeley. He is a member of the IEEE Computer Society, the IEEE, and the ACM. Contact him at david@cs.wisc.edu.*

***Daniel J. Sorin** is an assistant professor of electrical and computer engineering and of computer science at Duke University. His research interests are in multiprocessor memory systems, with emphasis on availability, verification, and analytical performance evaluation. Sorin received a PhD in electrical and computer engineering from the University of Wisconsin-Madison. Contact him at sorin@ee.duke.edu.*