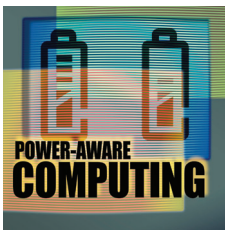
Energy Management for Commercial Servers



As power increasingly shapes commercial systems design, commercial servers can conserve energy by leveraging their unique architecture and workload characteristics.

Charles Lefurgy
Karthick Rajamani
Freeman Rawson
Wes Felter
Michael Kistler
Tom W. Keller
IBM Austin Research Lab

In the past, energy-aware computing was primarily associated with mobile and embedded computing platforms. *Servers*—high-end, multiprocessor systems running commercial workloads—typically included extensive cooling systems and resided in custom-built rooms for high-power delivery. In recent years, however, as transistor density and demand for computing resources have rapidly increased, even high-end systems face energy-use constraints. Moreover, conventional computers are currently air cooled, and systems are approaching the limits of what manufacturers can build without introducing additional techniques such as liquid cooling. Clearly, good energy management is becoming important for all servers.

Power management challenges for commercial servers differ from those for mobile systems. Techniques for saving power and energy at the circuit and microarchitecture levels are well known,[1] and other low-power options are specialized to a server's particular structure and the nature of its workload. Although there has been some progress, a gap still exists between the known solutions and the energy-management needs of servers.

In light of the trend toward isolating disk resources in separate cabinets and accessing them through some form of storage networking, the main focus of energy management for commercial servers is conserving power in the memory and microprocessor subsystems. Because their workloads are typically structured as multiple-application programs, system-wide approaches are more applicable to multiprocessor environments in commercial servers than techniques that are primarily applicable to single-application environments, such as those based on compiler optimizations.

COMMERCIAL SERVERS

Commercial servers comprise one or more high-performance processors and their associated caches; large amounts of dynamic random-access memory (DRAM) with multiple memory controllers; and high-speed interface chips for high-memory bandwidth, I/O controllers, and high-speed network interfaces.

Servers with multiple processors typically are designed as symmetric multiprocessors (SMPs), which means that the processors share the main memory and any processor can access any memory location. This organization has several advantages:

- Multiprocessor systems can scale to much larger workloads than single-processor systems.
- Shared memory simplifies workload balancing across servers.
- The machine naturally supports the shared-memory programming paradigm that most developers prefer.
- Because it has a large capacity and high-bandwidth memory, a multiprocessor system can efficiently execute memory-intensive workloads.

In commercial servers, memory is hierarchical. These servers usually have two or three levels of

0018-9162/03/$17.00 © 2003 IEEE

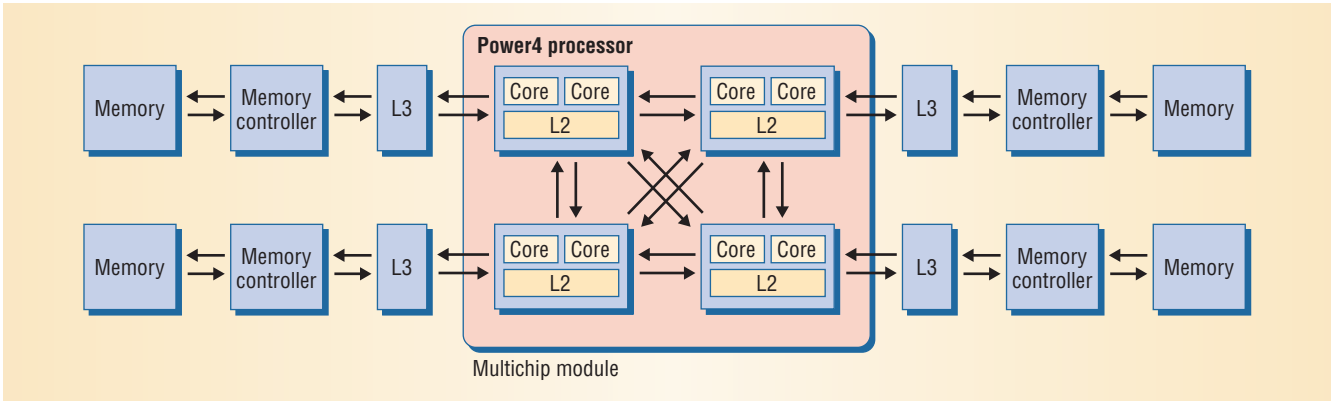


Figure 1. A single multichip module in a Power4 system. Each processor has two processor cores, each executing a single program context. Each core contains L1 caches, shares an L2 cache, and connects to an off-chip L3 cache, a memory controller, and main memory.

Table 1. Power consumption breakdown for an IBM p670.

IBM p670 server	Processors	Memory	I/O and other	Processor and memory fans	I/O component fans	Total watts
Small configuration (watts)	384	318	90	676	144	1,614
Large configuration (watts)	840	1,223	90	676	144	2,972

cache between the processor and the main memory. Typical high-end commercial servers include IBM's p690, HP's 9000 Superdome, and Sun Microsystems' Sun Fire 15K.

Figure 1 shows a high-level organization of processors and memory in a single multichip module (MCM) in an IBM Power4 system. Each Power4 processor contains two processor cores; each core executes a single program context.

The processor's two cores contain L1 caches (not shown) and share an L2 cache, which is the coherence point for the memory hierarchy. Each processor connects to an L3 cache (off-chip), a memory controller, and main memory. In some configurations, processors share the L3 caches. The four processors reside on an MCM and communicate through dedicated point-to-point links. Larger systems such as the IBM p690 consist of multiple connected MCMs.

Table 1 shows the power consumption of two configurations of an IBM p670 server, which is a midrange version of the p690. The top row gives the power breakdown for a small four-way server (single MCM with four single-core chips) with a 128-Mbyte L3 cache and a 16-Gbyte memory. The bottom row gives the breakdown for a larger 16-way server (dual MCM with four dual-core chips) with a 256-Mbyte L3 cache and a 128-Gbyte memory.

The power consumption breakdowns include

- the processors, including the MCMs with processor cores and L1 and L2 caches, cache controllers, and directories;
- the memory, consisting of the off-chip L3 caches, DRAM, memory controllers, and high-bandwidth interface chips between the controllers and DRAM;
- I/O and other nonfan components;
- fans for cooling processors and memory; and
- fans for cooling the I/O components.

We measured the power consumption at idle. A high-end commercial server typically focuses primarily on performance, and the designs incorporate few system-level power-management techniques. Consequently, idle and active power consumption are similar.

We estimated fan power consumption from product specifications. For the other components of the small configuration, we measured DC power. We estimated the power in the larger configuration by scaling the measurements of the smaller configuration based on relative increases in the component quantities. Separate measurements were made to obtain dual-core processor power consumption.

In the small configuration, processor power is greater than memory power: Processor power accounts for 24 percent of system power, memory power for 19 percent. In the larger configuration, the processors use 28 percent of the power, and memory uses 41 percent. This suggests the need to supplement the conventional, processor-centric approach to energy management with techniques for managing memory energy.

The high power consumption of the computing components generates large amounts of heat, requiring significant cooling capabilities. The fans driving the cooling system consume additional power. Fan power consumption, which is relatively fixed for the system cabinet, dominates the small configuration at 51 percent, and it is a big component of the large configuration at 28 percent. Reducing the power of computing components

would allow a commensurate reduction in cooling capacity, therefore reducing fan power consumption.

We did not separate disk power because the measured system chiefly used remote storage and because the number of disks in any configuration varies dramatically. Current high-performance SCSI disks typically consume 11 to 18 watts each when active.

Fortunately, this machine organization suggests several natural options for power management. For example, using multiple, discrete processors allows for mechanisms to turn a subset of the processors off and on as needed. Similarly, multiple cache banks, memory controllers, and DRAM modules provide natural demarcations of power-manageable entities in the memory subsystem. In addition, the processing capabilities of memory controllers, although limited, can accommodate new power-management mechanisms.

Energy-management goals

Energy management primarily aims to limit maximum power consumption and improve energy efficiency. Although generally consistent with each other, the two goals are not identical. Some energy-management techniques address both goals, but most implementations focus on only one or the other.

Addressing the power consumption problem is critical to maintaining reliability and reducing cooling requirements. Traditionally, server designs countered increased power consumption by improving the cooling and packaging technology. More recently, designers have used circuit and microarchitectural approaches to reduce thermal stress.

Improving energy efficiency requires either increasing the number of operations per unit of energy consumed or decreasing the amount of energy consumed per operation. Increased energy efficiency reduces the operational costs for the system's power and cooling needs. Energy efficiency is particularly important in large installations such as data centers, where power and cooling costs can be sizable.

A recent energy management challenge is leakage current in semiconductor circuits, which causes transistors designed for high frequencies to consume power even when they don't switch. Although we discuss some technologies that turn off idle components, and reduce leakage, the primary approaches to tackling this problem center on improvements to circuit technology and microarchitecture design.

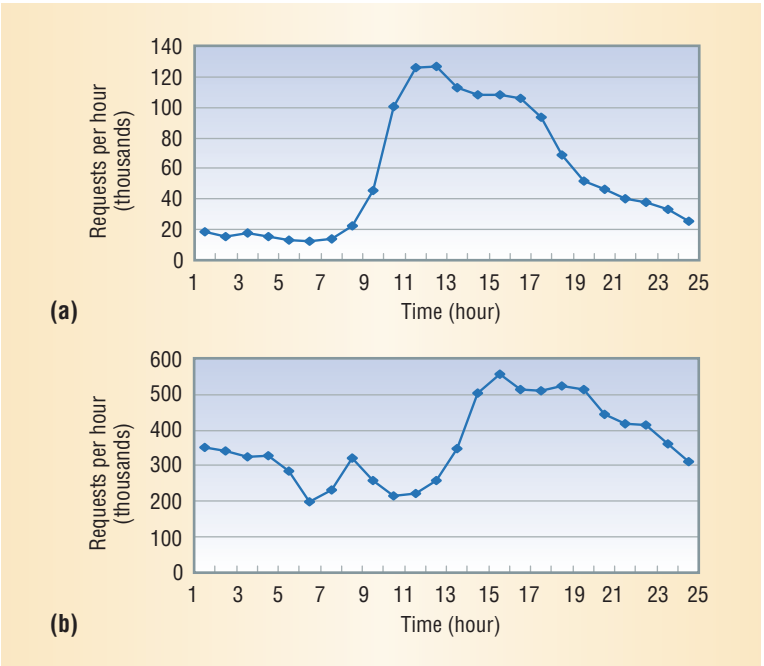


Figure 2. Load variation by hour at (a) a financial Web site and (b) the 1998 Olympics Web site in a one-day period. The number of requests received varies widely depending on time of day and other factors.

Server workloads

Commercial server workloads include transaction processing—for Web servers or databases, for example—and batch processing—for noninteractive, long-running programs. Transaction and batch processing offer somewhat different power management opportunities.

As Figure 2 illustrates, transaction-oriented servers do not always run at peak capacity because their workloads often vary significantly depending on the time of day, day of the week, or other external factors. Such servers have significant buffer capacity to maintain performance goals in the event of unexpected workload increases. Thus, much of the server capacity remains unutilized during normal operation.

Transaction servers that run at peak throughput can impact the latency of individual requests and, consequently, fail to meet response time goals. Batch servers, on the other hand, often have less stringent latency requirements, and they might run at peak throughput in bursts. However, their more relaxed latency requirements mean that sometimes running a large job overnight is sufficient. As a result, both transaction and batch servers have idleness—or *slack*—that designers can exploit to reduce the energy used.

Server workloads can comprise multiple applications with varying computational and performance requirements. The server systems' organization often matches this variety. For example, a typical e-commerce Web site consists of a first tier of simple page servers organized in a cluster, a second tier of higher performance servers running Web applications, and a third tier of high-performance database servers.

Although such heterogeneous configurations pri-

Realizing the potential of processor power and energy management often requires software support.

marily offer cost and performance benefits, they also present a power advantage: Machines optimized for particular tasks and amenable to workload-specific tuning for higher energy efficiencies serve each distinct part of the workload.

PROCESSORS

Processor power and energy management relies primarily on microarchitectural enhancements, but realizing their potential often requires some form of systems software support.

Frequency and voltage scaling

CMOS circuits in modern microprocessors consume power in proportion to their frequency and to the square of their operating voltage. However, voltage and frequency are not independent of each other: A CPU can safely operate at a lower voltage only when it runs at a low enough frequency. Thus, reducing frequency and voltage together reduces energy per operation quadratically, but only decreases performance linearly.

Early work on *dynamic frequency and voltage scaling* (DVS)—the ability to dynamically adjust processor frequency and voltage—proposed that instead of running at full speed and sitting idle part of the time, the CPU should dynamically change its frequency to accommodate the current load and eliminate slack.[2] To select the proper CPU frequency and voltage, the system must predict CPU use over a future time interval.

Much of the recent work in DVS has sought to develop prediction heuristics,[3,4] but further research on predicting processor use for server workloads is needed. Other barriers to implementing DVS on SMP servers remain. For example, many cache-coherence protocols assume that all processors run at the same frequency. Modifying these protocols to support DVS is nontrivial.

Standard DVS techniques attempt to minimize the time the processor spends running the operating system idle loop. Other researchers have explored the use of DVS to reduce or eliminate the stall cycles that poor memory access latency causes. Because memory access time often limits the performance of data-intensive applications, running the applications at reduced CPU frequency has a limited impact on performance.

Offline profiling or compiler analysis can determine the optimal CPU frequency for an application or application phase. A programmer can insert operations to change frequency directly into the application code, or the operating system can perform the operations during process scheduling.

Simultaneous multithreading

Unlike DVS, which reduces the number of idle processor cycles by lowering frequency, *simultaneous multithreading* (SMT) increases the processor use at a fixed frequency. Single programs exhibit idle cycles because of stalls on memory accesses or an application's inherent lack of instruction-level parallelism. Therefore, SMT maps multiple program contexts (*threads*) onto the processor concurrently to provide instructions that use the otherwise idle resources. This increases performance by increasing processor use. Because the threads share resources, single threads are less likely to issue highly speculative instructions.

Both effects improve energy efficiency because functional units stay busy executing useful, nonspeculative instructions.[5] Support for SMT chiefly involves duplicating the registers describing the thread state, which requires much less power overhead than adding a processor.

Processor packing

Whereas DVS and SMT effectively reduce idle cycles on single processors, *processor packing* operates across multiple processors. Research at IBM shows that average processor use of real Web servers is 11 to 50 percent of their peak capacity.[4] In SMP servers, this presents an opportunity to reduce power consumption.

Rather than balancing the load across all processors, leaving them all lightly loaded, processor packing concentrates the load onto the smallest number of processors possible and turns the remaining processors off. The major challenge to effectively implementing processor packing is the current lack of hardware support for turning processors in SMP servers on and off.

Throughput computing

Certain workload types might offer additional opportunities to reduce processor power. Servers typically process many requests, transactions, or jobs concurrently. If these tasks have internal parallelism, developers can replace high-performance processors with a larger number of slower but more energy-efficient processors providing the same throughput.

Piranha[6] and our Super-Dense Server[7] prototype are two such systems. The SDS prototype, operating as a cluster, used half of the energy that a conventional server used on a transaction-processing workload. However, throughput computing is not

suitable for all applications, so server makers must develop separate systems to provide improved single-thread performance.

MEMORY POWER

Server memory is typically double-data rate synchronous dynamic random access memory (DDR SDRAM), which has two low-power modes: *power-down* and *self-refresh*. In a large server, the number of memory modules typically surpasses the number of outstanding memory requests, so memory modules are often idle. The memory controllers can put this idle memory into low-power mode until a processor accesses it.

Switching to power-down mode or back to active mode takes only one memory cycle and can reduce idle DRAM power consumption by more than 80 percent. This savings may increase with newer technology.

Self-refresh mode might achieve even greater power savings, but it currently requires several hundred cycles to return to active mode. Thus, realizing this mode's potential benefits requires more sophisticated techniques.

Data placement

Data distribution in physical memory determines which memory devices a particular memory access uses and consequently the devices' active and idle periods. Memory devices with no active data can operate in a low-power mode.

When a processor accesses memory, memory controllers can bring powered-down memory into an active state before proceeding with the access. To optimize performance, the operating system can activate the memory that a newly scheduled process uses during the context switch period, thus largely hiding the latency of exiting the low-power mode.[8,9]

Better page allocation policies can also save energy. Allocating new pages to memory devices already in use helps reduce the number of active memory devices.[9,10] Intelligent page migration—moving data from one memory device to another to reduce the number of active memory devices—can further reduce energy consumption.[9,11]

However, minimizing the number of active memory devices also reduces the memory bandwidth available to the application. Some accesses previously made in parallel to several memory devices must be made serially to the same memory device.

Data placement strategies originally targeted low-end systems; hence, developers must carefully evaluate them in the context of commercial server environments, where reduced memory bandwidth can substantially impact performance.

Minimizing the number of active memory devices also reduces the memory bandwidth available to the application.

Memory address mapping

Server memory systems provide configurable address mappings that can be used for energy efficiency. These systems partition memory among memory controllers, with each controller responsible for multiple banks of physical memory.

Configurable parameters in the system memory organization determine the *interleaving*—the mapping from a memory address to its location in a physical memory device. This mapping often occurs at the granularity of the higher-level cache line size.

One possible interleaving allocates consecutive cache lines to DRAMs that different memory controllers manage, striping the physical memory across all controllers. Another approach involves mapping consecutive cache lines to the same controller, using all the physical memory under one controller before moving to the next. Under each controller, memory address interleaving can similarly occur across the multiple physical memory banks.

Spreading consecutive accesses—or even a single access—across multiple devices and controllers can improve memory bandwidth, but it may consume more power because the server must activate more memory devices and controllers.

The interactions of interleaving schemes with configurable DRAM parameters such as page-mode policies and burst length have an additional impact on memory latencies and power consumption. Contrary to current practice in which systems come with a set interleaving scheme, a developer can tune an interleaving scheme to obtain the desired power and performance tradeoff for each workload.

Memory compression

An orthogonal approach to reducing the active physical memory is to use a system with less memory. IBM researchers have demonstrated that they can compress the data in a server's memory to half its size.[12] A modified memory controller compresses and decompresses data when it accesses the memory. However, compression adds an extra step to the memory access, which can be time-consuming.

Adding the 32-Mbyte L3 cache used in the IBM compression implementation significantly reduces the traffic to memory, thereby reducing the performance loss that compression causes. In fact, a compressed memory server can perform at nearly the same level as a server with no compression and double the memory.

Energy Conservation in Clustered Servers

Ricardo Bianchini, Rutgers University
Ram Rajamony, IBM Austin Research Lab

Power and energy conservation have recently become key concerns for high-performance servers, especially when deployed in large cluster configurations as in data centers and Web-hosting facilities.[1]

Research teams from Rutgers University[2] and Duke University[3] have proposed similar strategies for managing energy in Web-server clusters. The idea is to dynamically distribute the load offered to a server cluster so that, under light load, the system can idle some hardware resources and put them in low-power modes. Under heavy load, the system should reactivate the resources and redistribute the load to eliminate performance degradation. Because Web-server clusters replicate file data at all nodes, and traditional server hardware has very high base power—the power consumed when the system is on but idle—these systems dynamically turn entire nodes on and off, effectively reconfiguring the cluster.

Figure A illustrates the behavior of the Rutgers system for a seven-node cluster running a real, but accelerated, Web trace. The figure shows the evolution of the cluster configuration and offered loads on each resource as a function of time. It plots the load on each resource as a percentage of the nominal throughput of the same resource in one node. The figure shows that the network interface is the bottleneck resource throughout the entire experiment.

Figure B shows the cluster power consumption for two versions of the same experiment, again as a function of time. The lower curve (dynamic configuration) represents the reconfig-

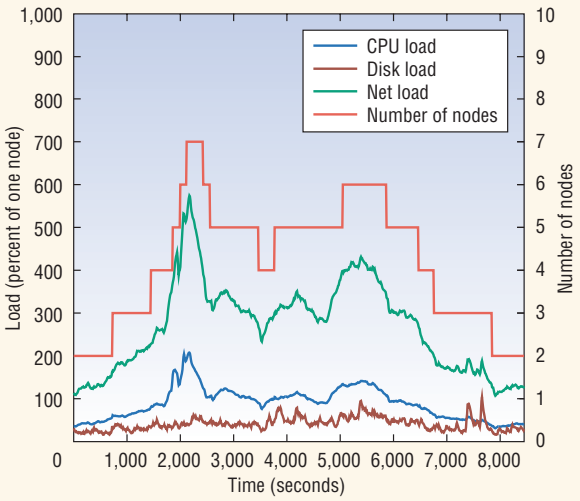


Figure A. Cluster evolution and per-resource offered loads. The network interface is the bottleneck resource throughout the entire experiment.

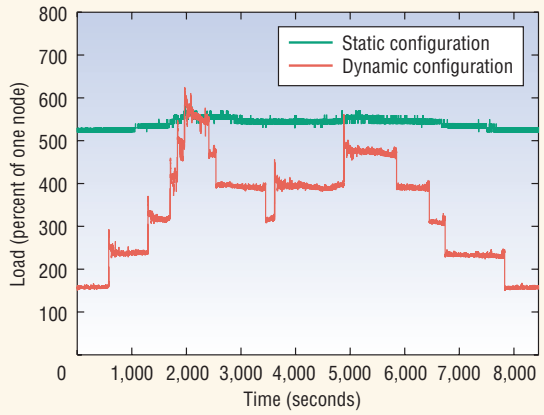


Figure B. Power under static and dynamic configurations. Reconfiguration reduces power consumption significantly for most of the experiment, saving 38 percent in energy.

Although the modified memory controller and larger caches need additional power, high memory capacity servers should expect a net savings. However, for workloads with the highest memory bandwidth requirements and working sets exceeding the compression buffer size, this solution may not be energy efficient. Quantifying compression's power and performance benefit requires further analysis.

Cache coherence

Reducing overhead due to cache coherence traffic can also improve energy efficiency. Shared-memory servers often use invalidation protocols to maintain cache coherency. One way to implement these protocols is to have one cache level (typically L2 cache) track—or *snoop*—memory requests from all processors.

The cache controllers of all processors in the server share a bus to memory and the other caches. Each cache controller arrives at the appropriate coherence action to maintain consistency between the caches based on its ability to observe memory traffic to and from all caches, and to snoop remote caches. Snoop accesses from other processors greatly increase L2 cache power consumption.

Andreas Moshovos and colleagues introduced a small cache-like structure they term a *Jetty* at each L2 cache to filter incoming snoop requests.[13] The Jetty predicts whether the local cache contains a requested line with no false negatives. Querying the Jetty first and forwarding the snoop request to the L2 cache only when it predicts the line is in the local cache reduces L2 cache energy for all accesses by about 30 percent.

Craig Saldanha and Mikko Lipasti propose replacing parallel snoop accesses with serial accesses.[14] Many snoop cache implementations broadcast the snoop request in parallel to all caches on the SMP bus. Snooping other processors' caches

urable version of the system, whereas the higher curve (static configuration) represents a configuration fixed at seven nodes. The figure shows that reconfiguration reduces power consumption significantly for most of the experiment, saving 38 percent in energy.

Karthick Rajamani and Charles Lefurgy[4] studied how to improve the cluster reconfiguration technique's energy saving potential by using spare servers and history information about peak server loads. They also modeled the key system and workload parameters that influence this technique.

Mootaz Elnozahy and colleagues[5] evaluated different combinations of cluster reconfiguration and two types of dynamic voltage scaling—*independent* and *coordinated*—for clusters in which the base power is relatively low. In independent voltage scaling, each server node makes its own independent decision about what voltage and frequency to use, depending on the load it is receiving. In coordinated voltage scaling, nodes coordinate their voltage and frequency settings. Their simulation results showed that, while either coordinated voltage scaling or reconfiguration is the best technique depending on the workload, combining them is always the best approach.

In other research, Mootaz Elnozahy and colleagues[6] proposed using *request batching* to conserve processor energy under a light load. In this technique, the network interface processor accumulates incoming requests in memory, while the server's host processor is in a low-power state. The system awakens the host processor when an accumulated request has been pending for longer than a threshold.

Taliver Heath and colleagues[7] considered reconfiguration in the context of heterogeneous server clusters. Their experimental results for a cluster of traditional and blade nodes show that their heterogeneity-conscious server can conserve more than twice as much energy as a heterogeneity-oblivious reconfigurable server.

References

1. R. Bianchini and R. Rajamony, *Power and Energy Management for Server Systems*, tech. report DCS-TR-528, Dept. Computer Science, Rutgers Univ., 2003.
2. E. Pinheiro et al., "Dynamic Cluster Reconfiguration for Power and Performance," L. Benini, M. Kandemir, and J. Ramanujam, eds., *Compilers and Operating Systems for Low Power*, Kluwer Academic, 2003. Earlier version published in *Proc. Workshop Compilers and Operating Systems for Low Power*, 2001.
3. J.S. Chase et al., "Managing Energy and Server Resources in Hosting Centers," *Proc. 18th ACM Symp. Operating Systems Principles*, ACM Press, 2001, pp. 103-116.
4. K. Rajamani and C. Lefurgy, "On Evaluating Request-Distribution Schemes for Saving Energy in Server Clusters," *Proc. 2003 IEEE Int'l Symp. Performance Analysis of Systems and Software*, IEEE Press, 2003, pp. 111-122.
5. E.N. Elnozahy, M. Kistler, and R. Rajamony, "Energy-Efficient Server Clusters," *Proc. 2nd Workshop Power-Aware Computing Systems*, LNCS 2325, Springer, 2003, pp. 179-196.
6. E.N. Elnozahy, M. Kistler, and R. Rajamony, "Energy Conservation Policies for Web Servers," *Proc. 4th Usenix Symp. Internet Technologies and Systems*, Usenix Assoc., 2003, pp. 99-112.
7. T. Heath et al., "Self-Configuring Heterogeneous Server Clusters," *Proc. Workshop Compilers and Operating Systems for Low Power*, 2003, 75-84.

Ricardo Bianchini *is an assistant professor in the Department of Computer Science, Rutgers University. Contact him at ricardob@cs.rutgers.edu.*

Ram Rajamony *is a research staff member at IBM Austin Research Lab. Contact him at rajamony@us.ibm.com.*

one at a time reduces the total number of snoop accesses. Although this method can reduce the energy and maximum power used, it increases the average cache miss access latency. Therefore, developers must balance the energy savings in the memory system against the energy used to keep other system components active longer.

ENERGY MANAGEMENT MECHANISMS

Most energy management techniques consist of one or more mechanisms that determine the power consumption of system hardware components and a policy that determines the best use of these mechanisms.

Although developers can implement some energy-management techniques completely in hardware, combining hardware mechanisms with software policy is beneficial. Placing the policy in software allows for easier modification and adaptation, as well as more natural interfaces for users and administrators. Software policies can also use higher-level information such as application priorities, workload characteristics, and performance goals.

The most commonly used power-management mechanisms support several operating states with different levels of power consumption. We broadly categorize these operating states as

- *active*, in which the processor or device continues to operate, but possibly with reduced performance and power consumption. Processors might have a range of active states with different frequency and power characteristics.
- *idle*, in which the processor or device is not operating. Idle states vary in power consumption and the latency for returning the component to an active state.

The Advanced Configuration and Power Interface specification (www.acpi.info) provides a

Heterogeneous systems have significant potential as a power-efficient architecture.

common terminology and a standard set of interfaces for software power and energy management mechanisms. ACPI defines up to 16 active—or *performance*—states, P0 through P15, and three idle—or *power*—states, C1 through C3 (or D1 through D3 for devices). The specification defines a set of tables that define the power/performance/latency characteristics of these states, which are both processor- and device-dependent.

A simple but powerful approach to defining power-management policies is to specify a set of *system operating states*—compositions of system component states and modes—and then write a policy as a mapping from current system state and application activity to a system operating state in accordance with desired power and performance characteristics. Defining policies based on existing system states such as idle, processing-application, and processing-interrupt is an efficient and transparent way to introduce power-management policies without overhauling the entire operating system.

IBM and MontaVista Software took this approach in the design of Dynamic Power Management, a software architecture that supports dynamic voltage and frequency scaling for system-on-chip environments.[15] Developers can use DPM to manipulate processor cores and related bus frequencies according to high-level policies they specify with optional policy managers.

Although the initial work focuses on dynamic voltage and frequency scaling in system-on-chip designs, developing a similar architecture for managing other techniques in high-end server systems is quite possible.

Early work on software-managed power consumption focused on embedded and laptop systems, trading performance for energy conservation and thus extending battery life. Recently, researchers have argued that operating systems should implement power-conservation policies and manage power like other system resources such as CPU time, memory allocation, and disk access.[16]

Subsequently, researchers developed prototype systems that treat energy as a first-class resource that the operating system manages.[17] These prototypes use an energy abstraction to unify management of the power states and system components including the CPU, disk, and network interface.

Extending such operating systems to manage significantly more complex high-end machines and developing policies suitable for server system requirements are nontrivial tasks. Key questions include the frequency, overhead, and complexity of power measurement and accounting and the latency and overhead associated with managing power in large-scale systems.

Although it increases overall complexity, a hypervisor—a software layer that lets a single server run multiple operating systems concurrently—may facilitate solving the energy-management problems of high-end systems. The hypervisor, by necessity, encapsulates the policies for sharing system resources among execution images. Incorporating energy-efficiency rules into these policies gives the hypervisor control of system-wide energy management. It also allows developers to introduce energy-management mechanisms and policies that are specific to the server without requiring changes in the operating systems running on it.

Because many servers are deployed in clusters, and because it's relatively easy to turn entire systems on and off, several researchers have considered energy management at the cluster level. For example, the Muse prototype uses an economic model that measures the performance value of adding a server versus its energy cost to determine the number of active servers the system needs.[18]

Power-aware request distribution (PARD) attempts to minimize the number of servers needed by concentrating load on a few servers and turning the rest off.[19] As server load changes, PARD turns machines on and off as needed.

Ricardo Bianchini and Ram Rajamony discuss energy management for clusters further in the "Energy Conservation in Clustered Servers" sidebar.

FUTURE DIRECTIONS

Three research areas will profoundly impact server system energy management:

- power-efficient architectures,
- system-wide management of power-management techniques and policies, and
- evaluation of power and performance tradeoffs.

Heterogeneous systems have significant potential as a power-efficient architecture. These systems consist of multiple processing elements, each designed for power- and performance-efficient processing of particular workload types. For example, a heterogeneous system for Internet applications combines network processors with a set of energy-efficient processors to provide both efficient network-protocol processing and application-level computation.

With the increase in transistor densities, heterogeneity also extends into the microarchitectural arena, where a single chip combines several cores, each suited for efficient processing of a specific workload. Early work in this area combines multiple generations of the Alpha processor on a single chip, using only the core requiring the lowest power while still offering sufficient performance for the current workload.[20] As the workload changes, system software shifts the executing program from core to core, trying to match the application's required performance while minimizing power consumption.

Server systems can employ several energy-management techniques concurrently. A coordinated high-level management approach is critical to handling the complexity of multiple techniques and applications and achieving the desired power and performance. Specifying high-level power-management policies and mapping them to available mechanisms is a key issue that researchers must address.

Another question concerns where to implement the various policies and mechanisms: in dedicated power-management applications, other applications, middleware, operating systems, hypervisors, or individual hardware components. Clearly, none of these alternatives can achieve comprehensive energy management in isolation. At the very least, arriving at the right decision requires efficiently communicating information between the system layers.

The real challenge is to make such directed autonomy work correctly and at reasonable overhead. Heng Zeng and colleagues suggest using models based on economic principles.[17] Another option is to apply formal feedback-control and prediction techniques to energy management. For example, Sivakumar Velusamy and colleagues used control theory to formalize cache power management.[21] The Clockwork project[22] applies predictive methods to systems-level performance management, but less work has been done on using these methods.

Designers and implementers need techniques to correctly and conveniently evaluate energy management solutions. This requires developing benchmarks that focus not just on peak system performance but also on delivered performance with associated power and energy costs. Closely tied to this is the use of metrics that incorporate power considerations along with performance. Another key component is the ability to closely monitor system activity and correlate it to energy consumption.

Performance monitoring has come a long way in the past few years, with high-end processors supporting an array of programmable event-monitoring counters. But system-level power management requires information about bus-level transactions and activity in the memory hierarchy. Much of this information is currently missing. Equally important is the ability to relate the values of these counters to both energy consumption and the actual application-level performance. The ultimate goal, after all, is to achieve a good balance between application performance and system energy consumption. ■

Acknowledgment

This work was supported, in part, by the US Defense Advanced Research Projects Agency (DARPA) under contract F33615-01-C-1892.

References

1. T. Mudge, "Power: A First-Class Architectural Design Constraint," *Computer*, Apr. 2001, pp. 52-57.
2. M. Weiser et al., "Scheduling for Reduced CPU Energy," *Proc. 1st Symp. Operating Systems Design and Implementation*, Usenix Assoc., 1994, pp. 13-23.
3. K. Flautner and T. Mudge, "Vertigo: Automatic Performance-Setting for Linux," *Proc. 5th Symp. Operating Systems Design and Implementation* (OSDI), Usenix Assoc., 2002, pp. 105-116.
4. P. Bohrer et al., "The Case for Power Management in Web Servers," *Power-Aware Computing*, R. Graybill and R. Melhem, eds., Series in Computer Science, Kluwer/Plenum, 2002.
5. J. Seng, D. Tullsen, and G. Cai, "Power-Sensitive Multithreaded Architecture," *Proc. 2000 Int'l Conf. Computer Design*, IEEE CS Press, 2000, pp. 199-208.
6. L.A. Barroso et al., "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing," *Proc. 27th ACM Int'l Symp. Computer Architecture*, ACM Press, 2000, pp. 282-293.
7. W. Felter et al., "On the Performance and Use of Dense Servers," *IBM J. Research and Development*, vol. 47, no. 5/6, 2003, pp. 671-688.
8. V. Delaluz et al., "Scheduler-Based DRAM Energy Management," *Proc. 39th Design Automation Conf.*, ACM Press, 2002, pp. 697-702.
9. H. Huang, P. Pillai, and K.G. Shin, "Design and Implementation of Power-Aware Virtual Memory," *Proc. Usenix 2003 Ann. Technical Conf.*, Usenix Assoc., 2003, pp. 57-70.

10. A.R. Lebeck et al., "Power-Aware Page Allocation," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS), ACM Press, 2000, pp. 105-116.
11. V. Delaluz, M. Kandemir, and I. Kolcu, "Automatic Data Migration for Reducing Energy Consumption in Multi-Bank Memory Systems," *Proc. 39th Design Automation Conf.*, ACM Press, 2002, pp. 213-218.
12. R.B. Tremaine et al., "IBM Memory Expansion Technology (MXT)," *IBM J. Research and Development*, vol. 45, no. 2, 2001, pp. 271-286.
13. A. Moshovos et al., "Jetty: Filtering Snoops for Reduced Energy Consumption in SMP Servers," *Proc. 7th Int'l Symp. High-Performance Computer Architecture*, IEEE CS Press, 2001, pp. 85-96.
14. C. Saldanha and M. Lipasti, "Power Efficient Cache Coherence," *High-Performance Memory Systems*, H. Hadimiouglu et al., eds., Springer-Verlag, 2003.
15. B. Brock and K. Rajamani, "Dynamic Power Management for Embedded Systems," *Proc. IEEE Int'l SOC Conf.*, IEEE Press, 2003, pp. 416-419.
16. A. Vahdat, A. Lebeck, and C. Ellis, "Every Joule Is Precious: The Case for Revisiting Operating System Design for Energy Efficiency," *Proc. 9th ACM SIGOPS European Workshop*, ACM Press, 2000, pp. 31-36.
17. H. Zeng et al., "Currentcy: A Unifying Abstraction for Expressing Energy Management Policies," *Proc. General Track: 2003 Usenix Ann. Technical Conf.*, Usenix Assoc., 2003, pp. 43-56.
18. J. Chase et al., "Managing Energy and Server Resources in Hosting Centers," *Proc. 18th Symp. Operating Systems Principles* (SOSP), ACM Press, 2001, pp. 103-116.
19. K. Rajamani and C. Lefurgy, "On Evaluating Request-Distribution Schemes for Saving Energy in Server Clusters," *Proc. IEEE Int'l Symp. Performance Analysis of Systems and Software*, IEEE Press, 2003, pp. 111-122.
20. R. Kumar et al., "A Multi-Core Approach to Addressing the Energy-Complexity Problem in Microprocessors," *Proc. Workshop on Complexity-Effective Design* (WCED 2003), 2003; www.ece.rochester.edu/~albonesi/wced03.
21. S. Velusamy et al., "Adaptive Cache Decay Using Formal Feedback Control," *Proc. Workshop on Memory Performance Issues* (held in conjunction with the *29th Int'l Symp. Computer Architecture*), ACM Press, 2002; www.cs.virginia.edu/~skadron/Papers/wmpi_decay.pdf.
22. L. Russell, S. Morgan, and E. Chron, "Clockwork: A New Movement in Autonomic Systems," *IBM Systems J.*, vol. 42, no. 1, 2003, pp. 77-84.

Charles Lefurgy *is a research staff member at the IBM Austin Research Lab. His research interests include computer architecture and operating systems. Lefurgy received a PhD in computer science and engineering from the University of Michigan. He is a member of the ACM and the IEEE. Contact him at lefurgy@us.ibm.com.*

Karthick Rajamani *is a research staff member in the Power-Aware Systems Department at the IBM Austin Research Lab. His research interests include the design of computer systems and applications with the focus on power and performance optimizations. Rajamani received a PhD in electrical and computer engineering from Rice University. Contact him at karthick@us.ibm.com.*

Freeman Rawson *is a senior technical staff member at the IBM Austin Research Lab. His research interests include operating systems, middleware, and systems management. Rawson received a PhD in philosophy from Stanford University. He is a member of the IEEE Computer Society, the ACM, and AAAI. Contact him at frawson@us.ibm.com.*

Wes Felter *is a researcher at the IBM Austin Research Lab. His interests include operating systems, networking, peer-to-peer, and the sociopolitical aspects of computing. Felter received a BS in computer sciences from the University of Texas at Austin. He is a member of the ACM. Contact him at wmf@us.ibm.com.*

Michael Kistler *is a senior software engineer at the IBM Austin Research Lab and a PhD candidate in computer science at the University of Texas at Austin. His research interests include operating systems and fault-tolerant computing. Contact him at mkistler@us.ibm.com.*

Tom W. Keller *manages the Power-Aware Systems Department at the IBM Austin Research Lab. Keller received a PhD in computer sciences from the University of Texas at Austin. Contact him at tkeller@us.ibm.com.*