Optimizing Databases for SMP

By Stephen Horn and D. Britton Johnston

This article highlights the advantages of SMP in a transaction processing, database environment. It also takes an in-depth look at techniques for optimizing database scalability and enhancing database performance in an SMP environment.

IBM's combination of the PowerPC Architecture™ and the AIX 4.1 Symmetric Multiprocessing (SMP) capability is an example of an industry-wide commitment to support this new style of computing with complementary open hardware and system software offerings. Unlike past technology trends, vendors are trying to exploit the advantages of SMP and limit the disruptions of technology change. User organizations and software vendors can also facilitate this transition. By becoming more knowledgeable about SMP, they can leverage the technology to achieve its maximum benefit. Nowhere is SMP's leverage potentially greater than in database management.

SMP Advantages

Although SMP originated in scientific computing, its true advantages are in transaction processing and databases. The greatest advantage is the ability of different CPUs to handle multiple tasks simultaneously. Opportunities for parallelism exist throughout database management. Multiple disks, users, records, and requests can often be accessed, serviced, updated, and satisfied at the same (or nearly the same) time. SMP can deliver the greatest performance boost for this type of application. The big win occurs with databases because transaction loads and data structures can usually be partitioned so that more processing can be handled simultaneously.

The challenge of SMP is to effectively deliver the power of multiple CPUs working in parallel. Increasing the total number of processors increases the total processing power of a system—but not necessarily its throughput. All SMP systems follow a curve of diminishing returns, in which the extra throughput achieved by adding more CPUs becomes smaller with the addition of each CPU, as shown in Figure 1. This fall-off is due to the extra overhead of coordinating tasks among CPUs.

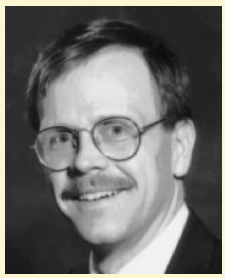
A system is considered scalable if its throughput increases in direct proportion to the number of CPUs added (as indicated by the 45° dotted line in Figure 1). The goal of optimization is to maintain the scalability curve near the 45° optimum (or 1:1 scalability) for as long as possible (Figure 2) using techniques at the system, database, and application levels.

Techniques for Optimizing SMP Performance

In an SMP machine, multiple CPUs execute multiple program flows called *threads* through a common memory state, as shown in Figure 3. Shared memory implies that more than one CPU can execute instructions from the same in-memory binary image and can access data from the same memory-resident structure.

The operating system controls the regions of memory that are accessed, when they are accessed, and which applications can access them. Most operating systems, however, grant limited memory allocation rights to applications. In fact, one benefit of an SMP-enabled Database Management System (DBMS) should be to hide the details of SMP so that applications can move unaltered from a uniprocessor machine to an SMP machine.

The key variables for optimizing database scalability in an SMP system are techniques for deferring or avoiding I/O, reducing memory contention, reducing locking overhead, and performing work in parallel.



Stephen Horn

D. Britton Johnston

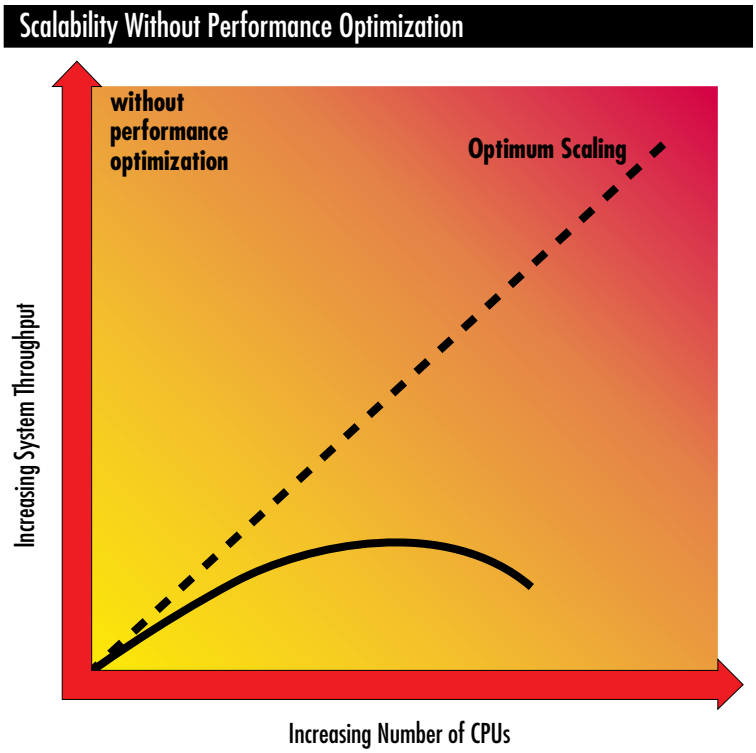


Figure 1. SMP scalability without performance optimization

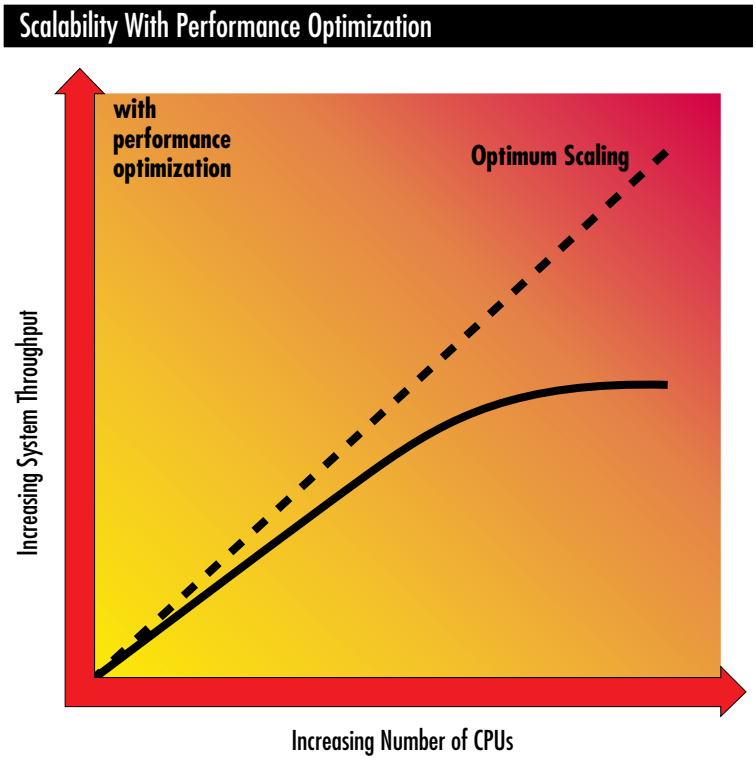


Figure 2. SMP scalability with performance optimization

Most DBMSs have features that manage (and allow users to manage) one or more of these performance parameters. The next section describes ways that applications can enhance database performance.

Enhancing Performance

The best way to optimize performance on a multi-CPU system is to start with a database that is already optimized for a single-CPU system. Many benefits automatically transfer to SMP by affecting one or more of the variables listed above.

Fine-Grain Locking

When one transaction locks a portion of memory, other transactions that depend on that portion of memory cannot complete. A fine-grain locking strategy limits the amount of memory and the length of time that memory is locked—thereby reducing its potential contention with other transactions. Three techniques to increase locking granularity are described below.

- **Multiple shared-memory locks**. Each data structure in shared memory can be controlled by its own lock rather than by a common lock shared with other data structures. Therefore, a client attempting to update the lock table to lock a record does not need to wait for another client process to finish updating the transaction table. Both processes can proceed in parallel. The resulting decrease in contention for shared memory can have a significant impact on throughput and response times, particularly on SMP machines with many users accessing the databases concurrently.
- **Granular transactions**. Only the parts of a transaction that write data require exclusive access to in-memory resources. By locking the data for only part of the transaction (that is, by making transactions more granular), the lock time can be dramatically reduced, and more transactions can use the same collection of in-memory structures at the same time.

  IBM's AIX 4.1 provides a special feature that supports granular transactions, an operating system feature that database system vendors should exploit. The AIX SMP architecture requires software-assisted cache coherency. By design, a database retains sufficient information to know when to resynchronize memory. By keeping this feature turned off until it is specifically required, the DBMS can further

reduce overhead in concurrent transaction processing on an SMP machine.

- **Record-level locking**. DBMSs that lock at the page or block level encounter serious contention problems when a transaction touches more than a few records. Since they must lock an entire block, all records in the block are locked for the duration of the transaction, possibly including tens or thousands of records not affected by the transactions. As more transactions begin, the frequency of contention increases exponentially, resulting in a massive gridlock of transactions waiting for needed records. This can devastate response time and throughput. If a transaction only locks a particular record, the frequency of contention is dramatically reduced.

Multiserver Operation

The ability of a DBMS to operate across multiple copies of itself against the same database is called *multiserver operation*. Organizations do not necessarily need an SMP machine to run multiple servers for the same database. In a distributed database, for example, separate physical databases stored at different locations in the network can be automatically linked together into a single logical database accessible to all tools and applications. However, the SMP machine with its multiple CPUs provides an ideal environment in which to run multiple servers. Users can achieve the parallelism of multiple servers without extra network overhead.

Multithreaded Operation

Multithreaded operation enables a server to accept multiple instruction streams from several clients simultaneously. Just as they have used multiple servers, organizations have run multithreaded DBMSs prior to SMP. The technique reduces CPU idle time (such as when a CPU waits for a disk I/O to complete) and serves multiple users concurrently. The architecture comes into its own, however, when threads have more than one CPU on which to execute, and can do so at the same time.

Spin Locks

Processes typically employ an operating system structure called a *semaphore* to request service from each other. Semaphores manage a queue of processes waiting for shared resources. When the resource is freed, the next process in the queue is granted access. In an SMP environment, semaphores can mean inefficient communication.

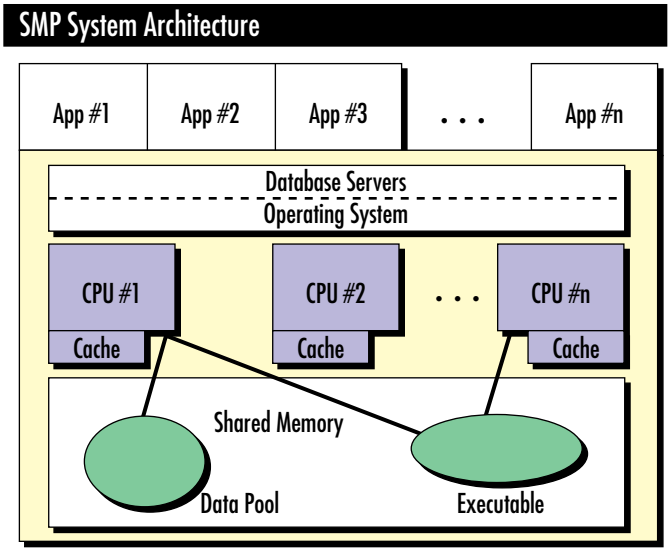


Figure 3. SMP system architecture

They generally involve the operating system and may require that one process sleep until the resource is ready for it (when it can then be reactivated by the operating system).

A spin lock differs from a semaphore. With *spin locks*, each process performs its own test for a requested resource. If the resource is being used, the process goes through a series of retries (the "spin"), checking for availability of the resource until it meets a preset (tunable) threshold, after which it resorts to a semaphore. The requesting process never has to relinquish its place in the run queue, go through the operating system, or change states (sleep/wake)—saving considerable time.

Asynchronous Page Writers (APWs)

This strategy also reduces the time that application processes spend waiting for resources such as disks. APWs are independent processes that periodically write updated database and log-file buffers to disk. If a server process needs to reuse a buffer, there is a high probability that a page writer has already written it to the disk. This means that the server process almost never has to perform its own I/O. APWs can be self-tuning to adjust the number of APWs that are active, the frequency in which they scan the buffer pool for modified pages, and the frequency in which they write dirty pages to disk.

Opportunistic Record Buffering

Another way to reduce disk I/O is to pack records from multiple transactions into a single network message to the client. This technique

also dramatically reduces the amount of network traffic, thereby speeding up disk accesses that are needed.

Larger Buffer Pool

The larger the memory buffer available to the DBMS, the more of the database that can be kept in memory. With more memory available, the probability is reduced that a particular part of memory will be required by two processes at the same time, thereby reducing the frequency of contention. A larger buffer pool also increases the likelihood that data will be held in memory, rather than on disk, so that the frequency of contentions caused by disk I/Os is also reduced.

What Programmers Can Do

With database-centered applications, application designers can always do things to maximize performance. One is to eliminate *hot spots*—those places within the database that the application constantly targets for retrieval or update purposes. If every CPU constantly searches for the same data, all but one CPU will be left waiting in line—regardless of the strategies employed by system designers to avoid or reduce those waits.

The most significant aspect under a programmer's control is the selection of the machine, operating system, and DBMSs themselves. Know the SMP scalability curve for the particular configuration of hardware and software you are considering. Ask the vendors if they provide special features to extend that curve and if those features play off each other. SMP is a natural for database applications. Be certain that your DBMS is a natural for SMP.

Stephen Horn, Progress Software Corporation, 14 Oak Park, Bedford, MA 01730. Internet: stephen.horn@progress.com. Mr. Horn is the database product manager for Progress® Software. He has over 20 years of experience in developing applications and systems software in databases and transaction processing. He has a BA in Operations Research from the University of Massachusetts.

D. Britton Johnston, Progress Software Corporation, 14 Oak Park, Bedford, MA 01730. Internet: britt.johnston@progress.com. Mr. Johnston is the database development manager for Progress Software. He has worked on design and implementation of various RDBMSs for ten years. Mr. Johnston has a BS in Computer Engineering from the University of Connecticut.