Oracle Database 10*g* on 64-bit Linux: Ready for Enterprise-class Computing

An Oracle White Paper
January 2004

ORACLE

Oracle Database 10*g* on 64-bit Linux: Ready for Enterprise-class Computing

Oracle Database 10*g* on 64-bit Linux: Ready for Enterprise-class Computing

INTRODUCTION

The growing popularity of the Linux operating system now extends to enterprise-class systems and mission-critical business applications. The advent of the new IA-64 architecture and Itanium processors offers unprecedented levels of power and robustness for enterprise Linux. Mission critical systems have challenging requirements for excellent performance and scalability, which can only be met by operating systems providing robust, scalable, and high-quality memory, process and I/O management.

Earlier versions of the Linux operating system had limitations that made it difficult to use for the enterprise. Intel, RedHat and Oracle have worked closely together to enhance Linux to enable companies to deploy large-scale configurations using the latest version of Oracle Database, Oracle Database 10*g*, with RedHat Linux 3.0 running on 64-bit systems based on the Intel Itanium 2 processor.

This paper gives an overview of the key enhancements made in the virtual memory and I/O subsystems, as well as the changes made in the process scheduling policies and the optimizations of the Intel compiler for Itanium 2.

OPERATING SYSTEM ENHANCEMENTS

Linux I/O subsystem

Asynchronous I/O

Before the introduction of asynchronous I/O, processes submitted disk I/O requests sequentially and synchronously. With synchronous I/O, when a request is submitted to the operating system, the writing process blocks until the write is completed. It is only after the request completes that the calling process can continue processing.

Asynchronous I/O allows a process to submit an I/O request without waiting for it to complete. Because the system does not put the process to sleep while the

I/O request is submitted and processed, the calling process is able to perform other tasks while the I/O proceeds.

Most I/O cache schemes employ deferred I/O operations. With Oracle Database, the database writer process writes data buffers modified in memory to the data files of the database, using a least recently used (LRU) algorithm. The database writer process manages the buffer cache so that user processes that must read blocks from disk into the cache can always find free buffers. This algorithm requires high I/O throughput at high transaction rates to keep sufficient number of database buffers free for new operations. At high transaction rates, and especially with the large percentages of update queries in OLTP transactions, the number of modified database buffers in the buffer cache can be very high. It is essential that the database writer process write those buffers to disk as quickly as possible to minimize the amount of CPU time consumed. The operation actually has very little data dependency on when the blocks are written, as long as the database server gets notified that the writes are completed. This requirement fits perfectly with the non-blocking semantics of asynchronous I/O.

Another advantage of this implementation is that it also allows Oracle processes to issue multiple I/O requests with a single system call, rather than a large number of distinct I/O requests. The system can optimize disk activity by reordering requests or combining individual requests that are adjacent on disk into fewer, larger, and thus less expensive, requests.

Relieve contention for global lock

Another area of improvement in the Linux I/O subsystem is the elimination of global lock usage for maintaining the integrity of kernel data structures that are accessed concurrently by multiple processes. In earlier releases of Linux, I/O requests were queued one at a time while holding a global lock (io_request_lock), used for the entire device block subsystem. This forced all processes performing I/O to compete for this single resource, even when I/Os were performed to unrelated devices, creating an unnecessary bottleneck and reduced overall system I/O throughput.

With the optimization, I/O requests are now queued while holding a lock specific to the queue associated with the request. With this fine-grained locking scheme, a separate lock is held for each individual block device. The result is a more scalable concurrent I/O queuing scheme and significantly better I/O throughput for multi-processor systems with multiple I/O controllers under heavy database load.

Variable Block I/O Sizes

Earlier versions of Linux require I/Os to be broken up into several blocks with a maximum size of 4KB (RAWIO_BLOCKSIZE), and then merged again by the SCSI layer or driver.

Support for variable block I/O sizes removes this constraint. Raw I/O requests can now be submitted as one single request, with variable block sizes.

Raw I/O, which refers to unbuffered I/O done through the /dev/raw interface, breaks I/O requests into 512-byte units (even if the device hardware and associated driver is capable of handling larger requests). The 512-byte requests are recombined within the request queue before being processed by the device driver. The variable block I/O optimization changes the block size used for raw I/O from 512 bytes to a much 16KB. This improves I/O throughput and efficiency by avoiding unnecessary split and recombinations. Internal kernel memory requirement is also significantly reduced with variable block I/O.

This optimization improves the performance and the scalability of I/O operations.

Virtual Memory: Huge TLB Pages

Database server applications consume large amounts of memory. Because accessing data from memory is an order of magnitude faster than accessing data from disk, Oracle shares memory segments,(thebuffer cache) among multiple database processes to hold copies of data blocks read from data files.

When a user process requires a particular piece of data, Oracle searches through the buffer cache. If the data is already in the cache (a cache hit), it will read the data directly from memory. Otherwise the data block is read from the data file on disk into memory (a cache miss). Cache misses are much more expensive than cache hits. Therefore, in a production environment, a system administrator will typically allocate as much as memory as possible for shared memory segments in order to increase the likelihood of cache hits.

Like other modern operating systems, Linux uses very sophisticated virtual memory management algorithms to make the most efficient use of physical memory resources. Linux allocates virtual memory to meet the total memory requirements of each process, and then manages the available physical memory to meet the actual memory requirements. With this environment, each Linux process executes as if it had the entire address space of the CPU available to itself.

The mapping between virtual addresses and physical addresses is done using a data structure called the page table. Linux maintains the page table of each process in physical memory. On IA-64, it uses a 3-level page table tree (3 levels of directory entries), which means that each time a processor accesses virtual memory, the translation of the virtual address results in four physical memory accesses: three to go through the page table levels and one for the physical page frame that the virtual address maps to. To avoid this performance penalty, each processor uses a small amount of associative memory, called a Translation Look-aside Buffer (TLB), to cache the page table entries of recently accessed virtual pages and uses the page table entry stored in the TLB to calculate the

physical address, making it possible for a virtual memory access to be as fast as a direct physical memory access. The TLB makes virtual memory practical because it is small and therefore can be built directly in to the CPU and run at full CPU speed.

There are very few TLB entries on a processor (typically a few dozen), so applications like Oracle that access large amounts of memory can have a high TLB miss rate. This is particularly damaging because with the amount of physical memory on database servers if growing very rapidly.

With these vast amounts of memory to access, there is a need to make each TLB mapping as large as possible to reduce the pressure on each processor's TLB. Large continuous regions in a process address space, such as contiguous data, may be mapped by using a small number of large pages rather than large number of small pages. Thus a requirement for having a separate large page size facility from the operating system becomes very important for both functionality and performance.

A new feature called Huge TLB pages allows applications to benefit from using large pages without affecting many other aspects of the OS. The Huge TLB page support brings several benefits to applications:

1. It increases the TLB coverage per CPU and reduces TLB miss rates

With large page support, the processor deals with more memory for each page table entry. Consequently, the TLB miss rate decreases significantly.

2. It reduces the memory usage of page tables

Page table size is greatly reduced which leads to improved memory utilization. This is because a single page table entry can address vast amounts of memory.

3. It eliminates the risk of swapping physical pages and reduces the page cache complexity

Better performance is also achieved because the big pages are not swapped, so the entire buffer cache is locked in physical memory. System performance increases as a result of less page swapping activities and less page cache complexity.

OPTIMIZING COMPILER FOR ITANIUM 2

Several optimizations were made to the compiler used for Oracle on Itanium 2 systems. These enhancements have several objectives:

1. To improve instruction cache utilization by reducing code size, separating frequently executed code from less frequently executed code, rearranging instructions to match program flow, and using instruction prefetch hints provided by the Itanium architecture.

2. To rearrange data and make use of Itanium cache control hints to make sure that frequently accessed data is found closer to the processor in the cache hierarchy.
3. To reduce memory traffic through efficient use of the large register set available on the Itanium architecture.
4. To meet the synchronization and consistency requirements of database operations without unneeded system overhead.

The result is an overall significant improvement of the performance of runtime execution of Oracle applications.

PERFORMANCE PROOFPOINTS

On December 8, 2003, the combination of Oracle Database running on Linux and Intel Itanium 2-based processing systems set a world record with the most important industry-standard benchmark for on-line transaction processing (OLTP) systems, the TPC benchmark C or TPC-C.

Oracle Database 10*g* broke the overall world record, achieving 1,184,893.38 tpmC, with a price/performance ratio of $5.52/tpmC. The benchmark was completed on a 16-node cluster of four-way HP Integrity rx5670 servers, with Intel Itanium 2 Processor 6M 1.5GHz, running Red Hat Enterprise Linux AS 3.0 and Oracle Database 10g with Real Application Clusters.

CONCLUSION

With these enhancements, Intel, RedHat and Oracle are providing the infrastructure necessary for deploying business-critical applications on enterprise-class Linux servers.

Customers who wish to migrate their applications to Linux systems will find the combination of Oracle Database 10g with RedHat Linux 3.0 and Intel Itanium 2 to be the configuration of choice for building scalable and high-performance IT solutions.

Oracle Database 10*g* on 64-bit Linux: Ready for Enterprise-class Computing
January 2004
Author: Herve Lejeune
Contributing Authors:

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com

Oracle Corporation provides the software
that powers the internet.