OpenVMS Technical Journal V7

Preliminary OLTP Performance Comparisons of Oracle Rdb V7.2 on OpenVMS I64 and Alpha

Author: Norman Lastovica, Oracle Rdb Engineering

Overview

Moving from the familiar environment of OpenVMS on Alpha and VAX systems to the world of OpenVMS running on Integrity Servers allows us to evaluate the performance and capabilities of another computer architecture and the systems built on it. Oracle is at the initial stages of optimizations of Oracle Rdb on the OpenVMS I64 platform, and we have performed preliminary performance tests comparing Alpha and Integrity servers.

This article provides some background information about the Oracle Rdb port to the OpenVMS I64 platform, and presents some observations based on the early performance tests performed with the HP OpenVMS engineering team during April of 2005.

To date, the results of the OLTP-oriented commercial workload tests indicate that system performance of current Integrity servers is at least as good as performance on the corresponding Alpha servers.

Run-time generated query-specific executable code

Rdb has always generated executable code at run time. This code is comprised of subroutines that are specific to the queries, fields, tables, data conversions that are necessary for database access. The performance of this code is a significant factor in the general database system performance characteristics of Oracle Rdb.

On the VAX architecture, VAX instructions are built into subroutines. Once the subroutine is complete, an REI instruction is executed, to make sure that:

- The CPU instruction caches are prepared for a change in the instruction stream
- The subroutine is available to be called

To move from the VAX architecture to the Alpha architecture, a new code generator in Rdb produces Alpha instructions, which are created into subroutines. Once this task is complete, an IMB (instruction memory barrier) causes the instruction cache to be invalidated before a new subroutine is called.

When porting Oracle Rdb to run on the Windows/NT operating system on the Intel I86 architecture, we took a slightly different approach. Rather than generating the I86 instruction set, a “pseudo code”

instruction set was created. These instructions are executed with a run-time interpretation software engine rather than the processor chip itself. This permitted rapid development and specialized high performance interpretation of potentially complex instructions. For example, rather than having to generate code "long hand" to perform a sorting operation of a vector of longwords, a single pseudo instruction indicates that a sort is required. The run-time interpreter calls a general sort subroutine. This results in improved performance and significantly reduced maintainability requirements.

Performance of the instruction interpreter is always a concern, but general product performance is at least as good as it is on the Alpha with Rdb V7.1. Therefore, minor performance optimizations can be performed as specific issues are identified. The high level of performance of Rdb over a wide variety of applications continues to be a primary goal.

I/O performance

For any database engine, the performance of disk I/O operations (both reading and writing) is of special concern. The entire I/O path (from $QIO or $IO_PERFORM though the operating system, drivers, I/O adaptors and controllers and so on) must be fast, reliable, and efficient.

Impact of Alignment Faults

Alignment faults can impact performance. For operations that occur infrequently, an occasional alignment fault does not cause any noticeable impact to performance. However, alignment faults that occur during an inner loop (starting or stopping a transaction, or fetching rows from the database) may cause measurable performance impacts. Furthermore, an alignment fault on the I64 system can be up to an order of magnitude slower than on the Alpha, due to the complexities of the I64 architecture. We minimized alignment faults on I64 (and, obviously, on Alpha as well).

Initial test system configuration

In order to make an equitable comparison, we configured a two-node clustered rx4640 I64 system, and an ES45 Alpha system, sharing a common SAN and EVA5000 storage controller. OpenVMS V8.2 ran on both machines, along with internal Oracle Rdb T7.2 field test build (approximately equal to the T7.2-030 field test kit). A simple application simulator executed simple transactions by updating a random record from a random database table. All indexes were cached in physical memory using the Oracle Rdb Row Cache feature.

The database consisted of 500,000,000 rows, and each row consisted of 32 quadword (BIGINT) columns of random data. Including after-image journals, the database was approximately 50,000,000 disk blocks.

Database population

The first test measured the relative performance of populating the database itself. Each table is loaded by a program that generates random row content and stores a row. 50,000 rows per transaction were stored in the database. Four copies of the loader program were run simultaneously.

Numerous tests were run during the initial configuration stage to make sure that the system was correctly set up and that all components were working as expected. Back-to-back performance tests showed that the ES45 consistently loads about 49,500 records per second, while the rx4640 loads about 71,000 records per second — a very positive sign that things are looking great

After the unexpectedly good performance of the database load process, we did not perform further analysis. Later analysis and testing revealed a bug in the Alpha code and a bug in the I64 code, which explains the difference in performance during the data loading stage. Both problems have been corrected.

Initial OLTP workload runs

Several preliminary test runs on both architectures revealed the typical application performance.

Comparisons between the two systems showed that the Alpha system consistently provides about 20% better performance than the IA64 system, which is surprising because our previous tests led us to expect some parity between the two configurations.

The difference in performance was due to a large difference in CPU consumption between the two machines. The I64 system had 100% CPU utilization while the Alpha system was running at less than

25% utilization. The database AIJ Log Server was the trigger to this problem. When the AIJ Log Server was disabled, the systems returned to roughly the behavior that we expected (the I64 system was no longer CPU-bound).

We suspect a synchronization problem between database users and the AIJ Log Server specific to internal differences in internal threading package within Rdb. We chose to disable the AIJ Log Server on both platforms for the remainder of the performance test and to analyze and correct the problem in the Rdb engineering labs at Oracle later; an I64-specific bug was found and corrected in the Rdb code.

Tools used to find performance problems

We used several of the tools available as SDA extensions to examination the CPU usage on the two systems more closely, including FLT and PRF.

FLT

The FLT tool showed that a number of alignment faults were generated within the Rdb database engine. Several BLISS macros at the root of the problem were modified to explicitly identify data that was not naturally aligned in memory, allowing the compiler to produce code that deals with the unaligned data better and avoids the alignment faults. Because our time was limited, we were not able to correct all of the faulting locations in the source code, but we addressed many of them during and after the testing.

PRF

The PRF tool analyzes the behavior of the running program. Because the Rdb database engine runs primarily in executive mode, we specified PC sampling of only those instructions that were executed in executive mode. At first we were astonished to discover that 20% of the CPU samples was used by three lines of source code in one module of Rdb. We repeatedly collected samples because it was so surprising, but the answer remained the same.

We found that, as each transaction was committed, a particular internal list was being scanned to clean up the transaction. For some classes of applications (including this one), the list itself became very large and scanning it was expensive. We replaced this list scanning method a different technique: a single-linked list was used to indicate only those blocks that were accessed during each particular transaction.

Our time on the systems was limited, so rather than attempting to make algorithmic changes in unfamiliar code in the Rdb database engine, we modified the test workload program to use dynamic SQL rather than pre-compiled statements. Dynamic SQL compiles a single outstanding request at a time, from Rdb's point of view. Contrast this with the hundreds of pre-compiled requests used by the original method. The goal was to reduce the size of the internal list of requests that was being scanned at each transaction commit.

Changing the program to use just a single dynamic request at a time changes the test profile considerably. Perprocess memory usage is reduced and user-mode CPU use is increased. (Both of these results were predicted and expected. Less memory is required because a large number of the request data structures need not be stored in memory, and additional CPU time is used to compile the dynamic request for each transaction.)

We ran the FLT tool again and the system executed many thousands of alignment faults per second. The faults occurred in the code that parsed the query BLR as database update statements were compiled in the database engine. These faults as well were corrected by modifying a few BLISS macros. We rebuilt and reinstalled an Rdb kit and most of the run-time alignment faults were eliminated on both platforms.

In equal circumstances, we anticipate that reducing the alignment faults benefits the I64 system somewhat more than the Alpha system, because alignment faults on the I64 platform are more expensive than on the Alpha. (Obviously they are expensive on Alpha as well.)

We then used the PRF tool to identify other hot spots in the OpenVMS executive code, as well as in SQL and the Rdb executive. With the workload running in a steady state, we collected CPU cycle

samples in Kernel, Executive, and User processor modes. User-mode samples represented code executing in the simulator program, and in the SQL-generated code and sharable images. Executive-mode CPU samples represent RMS and Rdb; in our test case, all samples were from within the RDMSHRP72 image. Finally, Kernel-mode execution is from within the VMS executive itself.

The resulting CPU samples for each of the processor modes surprised us. A significant percentage of the CPU tables fell into a very few modules and, in several cases, individual lines of code.

Improvements Based on Analysis

The collected data and analysis indicated additional attention for several areas. Rdb engineering made some algorithmic changes in transaction commit processing and SQL statement processing, as well as reducing alignment faults. Simultaneously, OpenVMS engineering rapidly prototyped several enhancements to avoid CPU consumption in several key areas. The prototype changes in Rdb and OpenVMS were carried forward into production release of both products.

The changes made by both HP and Oracle resulted improved Rdb performance improvement and OpenVMS performance. Application performance improved dramatically (on the order of 20%) on both the Alpha and the IA64 system. Most significantly, however, the transaction rate on the rx4640 and ES45 test systems was effectively identical (1,701 as opposed to 1,726 transactions per second).

The performance improvements made in Rdb V7.2 increase throughput and improve response time on Alpha systems as compared to Rdb V7.1. These increases are due to algorithmic improvements, code optimizations, and alignment fault avoidance.

Comparing Rdb V7.2 with Rdb V7.1 on Alpha

A requirement of Oracle Rdb V7.2 is that it must be at least as fast as the prior release. Oracle fully expects that the performance improvements made in Rdb V7.2 for both Alpha and I64 yield a positive result.

We ran many tests back to back on Alpha systems in Oracle's development lab. Comparisons of Rdb V7.2 with Rdb V7.1 running both OLTP and reporting workloads show that Rdb V7.2 is generally faster (over 10% faster for some isolated test cases) or, at worst, equally as fast as Rdb V7.1. In addition, database creation operations with Rdb V7.2 issues only about half of the total number of I/O operations that are issued by Rdb V7.1, while using less CPU time and completing in less elapsed time.

Future I64 Platforms

OpenVMS running on I64 systems performs comparably with Alpha systems, in the size range that we tested. We fully expect that future I64-based processors and systems will provide even higher levels of performance.

Oracle Rdb is still in the early stages of performance analysis and development on OpenVMS I64 systems. We expect that analysis of production applications will yield more information that Oracle will be able to use to improve the performance and reliability of Oracle Rdb.

OpenVMS and Oracle Engineering Credits

The performance tests and improvements described in this paper were carried out in a very short time. Craig Showers and Bill Gabor at the HP OpenVMS benchmark center in Nashua, New Hampshire, arranged and configured the systems used. Christian Moser and Greg Jordan of OpenVMS engineering provided invaluable performance analysis techniques and tools. Without their help, it would have been much more difficult to identify hot spots in the Rdb database engine. They also prototyped significant improvements for several OpenVMS performance-related areas.

All of the Oracle Rdb engineers deserve credit for porting the Rdb product family to OpenVMS on I64 and for providing the highest levels of performance in OLTP workloads.

For more information

For more information, please visit us on the Internet at www.oracle.com/rdb.