Performance of DEC Rdb Version 6.0 on AXP Systems

by

Lucien A. Dimino, Rabah Mediouni, T.K. Rengarajan,
Michael S. Rubino, and Peter M. Spiro

ABSTRACT

The Alpha AXP family of processors provided a dramatic increase in CPU speed. Even with slower processors, many database applications were dominated by relatively slow I/O rates. To maintain a balanced system, database software must incorporate techniques that specifically address the disparity between CPU speed and I/O performance. The DEC Rdb version 6.0 database management system contains shorter code paths, fewer I/O operations, and reduced stall times. These enhancements minimize the effect of the I/O bottleneck and allow the AXP processor to run at its intended higher speeds. Empirical performance results show a marked improvement in I/O rates.

INTRODUCTION

The DEC Rdb for OpenVMS AXP product (hereafter in this paper designated as DEC Rdb) is Digital's flagship database management system.[1] The DEC Rdb relational database software competes effectively in multiple data processing domains such as stock exchanges, image-processing applications, telemedicine, and large databases used for decision support or scientific applications. Virtually all these application frameworks are demanding increased processing power and increased I/O capabilities.

The Alpha AXP processor family represents a quantum jump in the processing power of CPUs. It is designed to scale up to 1,000 times the current processing power in a decade.[2] On the other hand, disk I/O latency is improving at a much slower rate than CPU power. As a result, especially on an AXP platform, the total time to execute a query is dominated by the time to perform the disk I/O operations. This disparity between processor speed and I/O latency is commonly called the I/O bottleneck or the I/O gap.[3]

In this paper, we describe our efforts to improve the performance of DEC Rdb on AXP systems. First we explain general porting steps that ensure a foundation of good performance on Alpha AXP systems. Then we describe our efforts to reduce the I/O bottleneck. We present the performance enhancements to various components of earlier versions of DEC Rdb and compare the enhancements and new features of version 6.0. Finally, we discuss the TPC-A transaction processing benchmark and present empirical results that quantify the benefits of the optimizations.

PERFORMANCE GAINS DURING THE PORT TO OpenVMS

In this section, we recount some of the initial, general performance modifications to the DEC Rdb software in the port from the VAX VMS system to the OpenVMS AXP system. We briefly discuss data alignment and reduction of the software subroutines or PALcode calls.

The DEC Rdb engineering developers saw the opportunity for increased performance through careful alignment and sizing of data structures. We needed to develop a solution that would exploit the performance gain of data alignment, yet still maintain an easy migration path for our large base of customers on VAX systems.

For the first release of DEC Rdb, all in-memory data structures were naturally aligned. In addition, many in-memory byte and word fields in these data structures were expanded to 32 bits of data (longwords). Once the in-memory data structures were aligned, we turned our attention to the on-disk data structures. The database root file, which is also frequently accessed, was completely aligned. New databases can be created with these aligned data structures, and existing databases can be aligned during a database convert operation on both VAX and AXP systems. This operation takes only seconds.

We did not align the data on the database pages in the storage area. Database pages contain the actual user data records. By leaving this data unaligned, we did not force a database unload/reload requirement on our customers. This factor and our support of clusters for both the VAX and the Alpha AXP architectures simplify migration from the VAX system to the AXP system.

After we completed the port of DEC Rdb, we ran performance benchmarks to determine which areas of the system could be enhanced. By using an internal tool called IPROBE, we learned that we could improve the performance of DEC Rdb by rewriting or eliminating code paths in PALcode subroutines. From January 1993 through July 1993, we reduced PALcode cycles from 143k to 62.6k per TPC-A transaction. Details of earlier performance modifications have been discussed in this Journal.[4]

PERFORMANCE ENHANCEMENTS

In this section, we describe how we improved the performance of DEC Rdb by addressing I/O bottleneck and problems in the code path.

The general strategy to combat the I/O gap in DEC Rdb was twofold. The first step was to minimize I/O operations. We used the "global buffers" of DEC Rdb to avoid read I/O requests. We

took advantage of the large physical memory available in modern computers to cache interesting parts of the database in memory and hence reduce the disk read I/O requests. The 64-bit Alpha AXP architecture allows computers to easily use more than 4 gigabytes (GB) of directly addressable memory. Write I/O requests are avoided by using the fast commit feature of DEC Rdb. The minimization of I/O activity is detailed in a technical report by Lomet et al.[5]

The second step was to reduce the stall time for the I/O operations that must be done. A reduced stall time allows DEC Rdb software to continue processing the queries even when disk I/O operations are in progress on its behalf. Two features of DEC Rdb version 6.0, the asynchronous prefetch and the asynchronous batch writes, reduce the stall time for the read and write I/O requests, respectively.

Although this strategy handles the I/O gap, the write to the after image journal (AIJ) becomes a limiting factor in high-performance transaction processing (TP) systems. The stall time for the AIJ write is reduced through the AIJ log server and AIJ cache on electronic disk features of DEC Rdb version 6.0.

In the following sections, we discuss these features in detail.

Write I/O Requests

To read a new set of pages from the database on disk, the DEC Rdb software selects a buffer in the buffer pool to be replaced. We refer to this as the victim buffer.

Prior to DEC Rdb version 6.0, writes of updated database pages to disk happened in synchronous batches. If the victim buffer is marked, a synchronous batch write is launched. In addition to the victim buffer, a number of least recently used (LRU) marked buffers are collected. The number of buffers in the batch write is specified as the BATCH_MAX parameter by users.

The list is then sorted by page numbers in order to perform global disk head optimization. After this, asynchronous disk write I/O requests are issued for all the marked buffers. In these earlier versions, the DEC Rdb product then waits until all the write I/O requests are completed. Although the individual writes are issued asynchronously and may complete in parallel, DEC Rdb waits synchronously for the entire batch write to complete. No query processing happens during the batch write. Figure 1 shows alternating synchronous batch writes and other work. Writes to the same disk are executed one after another, and writes to different disks are executed in parallel.

[Figure 1 (Alternating Synchronous Batch Writes and Other Work) is not available in ASCII format.]

The synchronous batch leverages two important optimizations: (1) parallelism between various disks and (2) disk head optimizations implemented at various levels of the storage hierarchy. The synchronous batch write feature reduced the average stall time per disk write I/O. The extent of reduction depends upon the degree to which the above two optimizations happen, which in turn depends upon the physical database design and application query behavior. In the TPC-A benchmark, the synchronous batch writes reduced the average stall time for the account write by 50 percent compared to synchronous individual writes.

To further reduce stall time, we implemented asynchronous batch writes (ABWs) in DEC Rdb version 6.0. With ABW, DEC Rdb now maintains the last few buffers unmarked. The size of this clean region of the buffer pool is specified by the user. As new pages are read from the database, database buffers migrate toward the end of the LRU chain of buffers. If a marked buffer were found in the clean region, an ABW would be invoked.

Figure 2 shows asynchronous batch writes invoked periodically, while other work continues. Processing does not explicitly wait for any of the disk write I/O requests to complete. (There may be implicit waits due to disk queuing effects.) Instead, processing continues. If a buffer with a pending write is chosen as the victim or if one of the buffers with pending writes is required for further processing, DEC Rdb then waits for completion of the pending writes. For applications with good temporal locality, it is likely that the buffers with pending writes will not be required for further processing.

Figure 2 also shows a rare instance in which other processing stops, waiting for one of the asynchronous write I/O requests to complete. Again, other processing includes stalls for disk read I/O requests. It is also possible to start a new ABW when the previous one has not yet completed.

[Figure 2 (Simultaneous Asynchronous Batch Writes and Other Work) is not available in ASCII format.]

Read I/O Requests

Requests for database pages are often satisfied by a large global buffer pool. This is true when the whole database fits in a large memory, common on Alpha AXP systems. Under certain circumstances, however, the buffer pool is not large enough to satisfy all requests. Moreover, seldom-used data may be replaced by more frequently used data in the buffer pool.

The essential strategy is to submit asynchronous disk read I/O requests well before the data is really needed during query processing. If the asynchronous prefetch (APF) request is made far enough in advance of the actual request, the process will not

need to stall for the I/O. Critical to any prefetch strategy is to reliably determine the desirable database pages for the immediate future.

Fortunately, applications request data in sets of rows. Therefore, based on user requests and the query optimizer decisions, the database access patterns are known at the time query execution starts. This allows the mechanism to prefetch the data from the database into the buffer pool.

In DEC Rdb version 6.0, we implemented the mechanism to prefetch data from the database based on requests from higher layers of DEC Rdb. This is performed in an integrated manner in the buffer pool with the usual page locking protection in a cluster. We also implemented the policy to use asynchronous prefetch in the case of sequential scans.

Sequential scans are quite common in databases for batch applications producing large reports. They are also chosen by the optimizer when a large number of records are selected from a table for a query for processing.

The user can specify the parameter APF_DEPTH that controls the number of buffers to be used for prefetching database pages, hence the lead of prefetch ahead of the real fetches. With a sufficient level of prefetch, it is quite possible to exploit the parallelism in the disk subsystem as well as achieve close to the spiral transfer rate of disks, as we describe in the following test.

We placed one table in a mixed format area on an RA73 disk. The database server process used 400 buffers of 6-kilobyte (kB) size, and the APF_DEPTH parameter was set to 50 buffers. With these settings, the sequential scan of a 1-GB area took 593 seconds. This is equivalent to a transfer rate of 1.69 megabytes per second (MB/s), compared to the rated spiral transfer rate of 1.8 MB/s for the disk.

We performed another test to determine the improvement with the APF feature in DEC Rdb version 6.0. Again, we built a database with one table in a mixed format area, but this time on a stripe set of two RA92 disks. The database server process used 400 buffers of 3-kB size. The elapsed time to scan a 1-GB area in version 5.1 was 6,512 seconds, and the transfer rate was 0.15 MB/s. In version 6.0 the elapsed time was 569 seconds, and the transfer rate was 1.76 MB/s. An APF_DEPTH of 100 buffers was used for version 6.0. We find that APF has made sequential scan 10 times faster in DEC Rdb version 6.0. Note that this performance improvement can be made much better by using more disks in the stripe set and by using more powerful processors.

Commit Time Processing

Although the disk I/O stalls are significantly reduced by the APF and ABW in DEC Rdb version 6.0, the commit time processing remains a significant wait.

Prior to version 6.0, DEC Rdb software used a cooperative flushing protocol. Each database server produced the AIJ log records and then used the lock manager to determine the group committer. The group committer then formatted the AIJ log records for all the servers and flushed the data to the AIJ file in one disk write I/O. Each server thus competed for the AIJ lock in order to become the group committer. Each server also had to acquire the AIJ lock even to determine if it had committed. Figure 3 shows this algorithm. DEC Rdb software also supports timer-based tuning methods to increase the group size for group commit.[6]

Figure 3 Cooperative Flushing Algorithm

```
Put AIJ data in shared memory

Get AIJ lock

if      our data IS flushed
then    begin

        Release AIJ lock

        return
        end

! We are the group committer

Format AIJ data in shared memory

Reserve space in AIJ file

Release AIJ lock

Write AIJ data to AIJ file

Indicate AIJ data is flushed for group members
```

AIJ Log Server. With DEC Rdb version 6.0, a dedicated process called the AIJ log server (ALS) runs on every node of the cluster to perform the task of processing group writes to the AIJ file for all servers. The new algorithm is in two parts, one for the database servers and another for the ALS. Figure 4 shows these algorithms.

Figure 4 AIJ Log Server Process

(a) Database Server Algorithm:

```
Put AIJ data in shared memory

Until   committed

        sleep       ! Woken up after commit by ALS
```

(b) ALS Algorithm:

```
Get AIJ lock

loop    begin

        Format AIJ data in shared memory

        Write AIJ data to AIJ file

        Indicate AIJ data is flushed for group members

        Wake up all committed processes

        end

Release AIJ lock
```

At commit time, database servers generate AIJ log records, store them in shared memory, and go to "sleep." They are "woken up" by the ALS when their commit records are successfully written to the AIJ file. The ALS gathers the AIJ log records of all users, formats them, writes them to the AIJ file, and then wakes up servers waiting for commit. Thus, the ALS performs AIJ writes in a continuous loop.

The ALS allows DEC Rdb version 6.0 to scale up to thousands of transactions per second with one magnetic disk using the TPC-A benchmark.

With the ALS, the average stall time for a server to commit is 1.5 times more than the time taken to perform one log write I/O. This stall is of the order of 17 milliseconds with 5,400-rpm disks. Note that this stall time is a function of disk performance only and is independent of the workload. In a high-throughput TP environment, where transaction times are very short, this stall at commit time is still a significant wait. Considering the speed of Alpha AXP processors, the commit stall is many times more than the processing time for servers.

AIJ Cache on Electronic Disk. The AIJ cache on electronic disk (ACE) is a feature of the ALS that also helps high-throughput TP systems. ACE utilizes a small amount (less than 1 MB) of very low

latency, solid-state disk to reduce the commit stall time. Figure 5 shows the new ACE algorithm with ALS. The ACE file is on a solid-state disk shared by all nodes of a cluster. The file is partitioned for use by various ALS servers in the cluster.

Figure 5 ACE Algorithm

```
Get AIJ lock

loop    begin

        Format AIJ data in shared memory

        Save AIJ block # and length of I/O
            in ACE header

        Write AIJ data and ACE header to ACE file
            ! 1ms

        Wake up all committed processes

        Write AIJ data to AIJ file    ! 11ms

        Set flags to indicate AIJ data is flushed
            for group members

        end

Release AIJ lock
```

Basically, the data is first written to the ACE file, and the servers are allowed to proceed to the next transaction. The data is then flushed to the AIJ file in a second I/O. The write to the ACE disk includes the virtual block number of the AIJ file where the log data is supposed to be written. The ACE disk serves as a write-ahead log for the AIJ write. By doubling the number of disk writes per group, we reduced the response time for the servers to 0.5 times the stall for AIJ write and 1.5 times the stall for ACE write. This reduction in the stall time conversely increases the CPU utilization of servers, thereby reducing the number of servers required to saturate an AXP CPU.

Backup and Restore Operations

Simply stated, the way we traditionally perform backup does not scale well with system capacity, rarely uses its resources effectively, and is disruptive to other database activity. We designed the backup and restore operations to resolve these issues. Before we present that discussion, we first examine the problems with the traditional database backup operation.

Traditional Backup Process. Using OpenVMS BACKUP as an example of

the traditional backup process, we find that a single backup process performance does not scale with CPU performance, nor with system aggregate throughput. Instead, it is limited by device throughput. The only way to increase the performance is to perform multiple backups concurrently. However, concurrent backups executing on the same CPU interfere with one another to some extent. Each additional backup process provides less than 90 percent of the performance of the previous one. Five OpenVMS BACKUP operations executing on one CPU provide no greater performance than four operations, each executing on its own CPU. Consequently, five tape drives may provide only four times the performance of a single drive.

The OpenVMS BACKUP operation is limited by the lesser of the disk throughput and the tape throughput. Read performance for an RA73 disk may be as high as 1.8 MB/s, but OpenVMS BACKUP more typically achieves between 0.8 and 1.0 MB/s. Performance for a TA92 tape is 2.3 MB/s. Five tape drives, with an aggregate throughput of 11.5 MB/s, and 25 disks, with an aggregate throughput of 45.0 MB/s, can only be backed up at a rate of 4.0 MB/s (14.4 GB per hour).

Increasing the CPU capacity and aggregate throughput improves the backup performance but not proportionally. Low device utilization and nonlinear scaling mean that as the system capacity and database size increase, the cost in system time and hardware for a given level of backup performance becomes increasingly burdensome.

The traditional backup process, such as provided by OpenVMS BACKUP, is not coordinated with database activity. The database activity must be prohibited during the backup. If it is not, the restore operation will produce an inconsistent view of the data. In the latter case, database activity journals are required to return the database to consistency. Application of these journals significantly reduces the performance of the restore process.

To maximize the performance of the traditional backup process, the backup must consume a large portion of the entire throughput of the disks being backed up. As a result, database activity and backup activity severely impede each another when they compete for these disks.

RMU BACKUP Operation. The RMU BACKUP operation resolves the problems with the traditional backup operation. RMU BACKUP is coordinated with database activity. It produces a consistent image of the database at a point in time without restricting database activity or requiring application of journals after the restore operation.

The RMU BACKUP operation is a multithreaded process; therefore, it backs up multiple disks to multiple tapes and eliminates the limiting factor associated with throughput of a single disk or a single tape drive. The aggregate of disk and tape throughputs

determines the performance of RMU. Because the aggregate disk throughput is usually significantly higher than the aggregate tape throughput, all the disks have spare throughput at all times during RMU BACKUP. Consequently, the RMU BACKUP process and database activity interfere to a much lesser extent than is the case with traditional backup. Its multithreaded design also scales linearly with CPU capacity, aggregate disk throughput, and aggregate tape throughput.

The first steps of an RMU BACKUP are to evaluate the physical mapping of the database to disk devices and to determine the system I/O configuration. RMU then devises a plan to execute the backup. The goals of this plan are to divide the data among the tape drives equally, to prohibit interference between devices sharing a common I/O path, and to minimize disk head movement. The generated plan is a compromise because complete and accurate configuration data is difficult to collect and costly to assimilate. To implement the plan, RMU creates a network of interacting threads. Each thread operates asynchronously, performing asynchronous multibuffered I/O to its controlled device. Interthread communication occurs through buffer exchange and shared memory structures.

RMU uses the database page checksum to provide end-to-end error detection between the database updater and the backup operation. It uses a backup file block cyclic redundancy check (CRC) to provide end-to-end error detection between the backup and the restore operations. In addition, RMU uses XOR recovery blocks to provide single error correction on restore. As a consequence, the data being backed up must be processed four times, which has a major effect on CPU usage. The first time is to evaluate the database page checksum. The second time is to copy the data and to exclude unused storage and redundant structures. Here we are willing to expend extra CPU cycles to reduce the I/O load. The third time is to generate an error detection code (CRC), and the fourth time is to generate an XOR error recovery block.

The RMU BACKUP operation improves performance in other ways. It does not back up redundant database structures, nor does it back up allocated storage that does not contain accessible data. As a result, the size of the backup file is significantly reduced, and relative backup performance is improved. The incremental backup feature selectively backs up only those database pages that have been modified. This provides an additional, significant reduction in backup file size relative to a file-oriented backup. These reductions in backup file size further improve RMU BACKUP performance relative to traditional backup.

The performance of the RMU RESTORE operation mirrors that of the RMU BACKUP operation. In spite of this, a natural asymmetry between the operations allows RMU BACKUP to outperform RMU RESTORE by 20 percent to 25 percent. There are several reasons for the asymmetry: the cost of allocating files, the asymmetry of read versus write disk performance, the asymmetric response of

the I/O subsystem to read versus write I/O operations under heavy load, and the need to re-create or initialize redundant data that was not backed up by RMU BACKUP.

The RMU RESTORE operation can restore the entire database, selected database files, and even selected pages within the files. Restoring files or pages requires exclusive access to only the objects being restored. This is the only restriction on database activity during the restore operation.

We performed a test to illustrate the effectiveness and scalability of RMU BACKUP with database size and system capacity. Table 1 gives the results. This test also demonstrates the high level of backup performance that can be provided on AXP systems without the use of exotic or expensive technology. Although the results are limited by the aggregate tape I/O performance, the CPU does not appear to have sufficient excess capacity to justify a test with a sixth tape drive.

Table 1 Backup Performance on AXP Systems

Configuration	Type	Amount
System	DEC 7000 Model 610 (182 MHz)	1
Disks	RA73	23
	RZ23	2
Controllers	HSC95	6
	CIXCD	6
	KMC44	5
Tape drive	TA92	5
Operating system	OpenVMS V1.5	
Database management software	DEC Rdb V5.1	
Database size of 48.8 GB (71% Relational data or 34.7 GB)		

Performance	
Backup time	1:11:29
Backup rate	41.0 GB/hour

Sustained disk data rate of 455 kB/s per disk (25% utilization)

Sustained tape data rate of 2.3 MB/s per tape (100% utilization)

Restore time	1:32:49
Restore rate	31.6 GB/hour

Unfortunately, comparable numbers are not available for other database products on comparable platforms. Nevertheless, when any database system relies on the operating system's backup engine, the expected performance limits are the same as the scenario presented.

Sorting Mechanism

A sorting mechanism is a key component of a database system. For example, the relational operations of join, select, and project, which are executed to satisfy user queries, often sort records to enable more efficient algorithms. Furthermore, selected records must often be presented to the user in sorted order. Quite literally, a faster sorting mechanism directly translates to faster executing queries. Hence during the port to the Alpha AXP platform, we also focused on ensuring that the DEC Rdb sorting mechanism worked well with the Alpha AXP architecture. In the case of the sort mechanism, we reduced I/O stalls and optimized for processor cache hits rather than access main memory.

Prior to the port to Alpha AXP, DEC Rdb utilized only the replacement-selection sort algorithm. For AXP systems, we have modified the sort mechanism so that it can also utilize the "quicksort" algorithm under the right conditions.[7] In fact, we developed a modified quicksort that sorts (key prefix, pointer) pairs. Our patented modification allows a better address locality that exploits processor caching.[8] This is especially important on the Alpha AXP platform since the penalty for addressing memory is significant due to the relatively fast processor speed. The quicksort algorithm also performs better than the replacement-selection algorithm in a memory-rich environment, which is the trend for Alpha AXP systems.

To summarize, if the required sort fits in main memory, DEC Rdb version 6.0 utilizes the optimized quicksort algorithm; if the sort requires temporary results on disk, DEC Rdb uses the traditional replacement-selection sort algorithm. When writing temporary results to disk, both sort algorithms also utilize asynchronous I/O mechanisms to reduce I/O stalls.

The DEC Rdb version 6.0 implementation of the quicksort algorithm is based on the AlphaSort algorithm developed at Digital's San Francisco Systems Center. The AlphaSort algorithm achieved the world-record sort of 7 seconds on an industry-standard sort benchmark. This result is more than 3 times faster than the previous sort record of 26 seconds on a CRAY Y-MP system.[8]

Multi-statement Procedures

Prior to the implementation of multi-statement procedures in DEC Rdb, individual SQL statements were serially submitted to the database engine for execution. This method of execution incurs excessive code path because individual SQL statements must traverse different layers before they reach the database engine.

When clients and servers communicate over a network, each SQL statement incurs the additional overhead of two network I/O operations. The result is a long transaction code path as well as delays due to excessive network traffic. Figure 6 shows the execution path that individual SQL statements traverse in both the local and the remote cases.

[Figure 6 (SQL Statement Execution Flow) is not available in ASCII format.]

Network overhead is a significant problem for client-server applications. Without the services of the VAX ACMS TP monitor, 14 network I/O operations are required to complete a single TPC-A transaction. Table 2 lists the TPC-A pseudocode needed to complete one transaction.

Table 2 Individual SQL Statements for Single TPC-A Transaction

TPC-A Pseudocode	Network I/O Operations
Start an update transaction	2
Update a row in BRANCH table	2
Update a row in TELLER table	2
Update a row in ACCOUNT table	2
Select Account_balance from ACCOUNT and display it on the terminal	2
Insert a row in the HISTORY table	2
Commit transaction	2
Total network I/O operations per TPC-A transaction	14

Prior to DEC Rdb version 6.0, client applications accessing remote servers relied on the services of the VAX ACMS TP monitor to reduce the network overhead. With ACMS present on both the client and the server, a message carrying a transaction request is transferred to the server in one network I/O. As shown in Figure 7, an ACMS server is then selected to execute the request, and a message is returned to the client upon transaction

completion. This reduces the number of network I/O operations to two per transaction. In simple applications such as TPC-A, however, a TP monitor can be very intrusive. Measurements taken on a VAX 6300 system running the TPC-A benchmark and using ACMS revealed that the TP monitor consumes 20 percent to 25 percent of the cycles on the back-end server.

[Figure 7 (Client-server Application with ACMS) is not available in ASCII format.]

SQL multi-statement procedures in DEC Rdb version 6.0 address both these performance issues. They reduce the transaction code path by compounding a number of SQL statements in one BEGIN-END procedure that fully adheres to the VAX and AXP procedure calling standards. The BEGIN-END procedure is atomically submitted to the database manager to compile once per database session and to execute as often as the application requires for the duration of that particular session.

When multiple SQL statements are bundled into a single BEGIN-END block, only two network I/O operations are required between the client and the remote server for each database request. Through the DEC Rdb remote facility, which is an integral part of DEC Rdb, client-server applications no longer need the services of a TP monitor. Therefore many of the processing cycles that would have been dedicated to the TP monitor are regained and applied toward processing requests.

The DEC Rdb remote server is an ordinary VMS process that is created upon the first remote request to the database. It has less overhead than the TP monitor. The DEC Rdb remote server remains attached to the database; it communicates with and acts on behalf of its client for the duration of a database session. Figure 8 shows our implementation of the TPC-A client-server application with DEC Rdb version 6.0. With the implementation of multi-statement procedures for the TPC-A transaction, we reduced the code path approximately 20 percent to 25 percent.

[Figure 8 (Client-server Application with DEC Rdb Version 6.0) is not available in ASCII format.]

PERFORMANCE MEASUREMENT

In this section, we briefly describe a TPC-A transaction and the TPC-A benchmark. We discuss our goals for TPC-A and recount our progress. Finally, we present profiling and benchmark results.

TPC-A Transaction

The TPC-A transaction is a very simple database transaction: a user debits or credits some amount of money from an account. In database terms, that requires four updates within the database: modify the user account balance, modify the branch account

balance, modify the teller account balance, and store a history record. The resulting metric indicates how many transactions are performed per second (TPS).[9]

In terms of complexity, the TPC-A transaction falls somewhere in the middle range of benchmarks. In other words, the SPECmark class of benchmarks is very simple and tends to stress processors and caching behavior; the sorting benchmarks (e.g., AlphaSort) expand the scope somewhat to test processors, caches, memory, and I/O capabilities; the TPC-A benchmark tests all the above in addition to stressing the database software (based on the relatively simple transaction). Other benchmarks such as TPC-C and TPC-D place even more emphasis on the database software, thereby overshadowing the hardware ramifications. Hence the TPC-A benchmark is a combined test of a processor and the database software.

During the porting cycle, Digital was aware that the highest result in the industry for a single-processor TPC-A benchmark was approximately 185 TPS. Toward the end of the porting effort, we worked for six to nine months to ensure that DEC Rdb had attained optimal performance for the TPC-A transaction.

In April 1993, the DEC Rdb database system was officially audited on a DEC 10000 AXP system at the world-record rate of 327.99 TPS. As a result, DEC Rdb became the first database system to exceed the 300 TPS mark on a single processor. A few weeks later, DEC Rdb was again audited and achieved 527.73 TPS on a dual-processor DEC 10000 AXP system. Thus, DEC Rdb became the first database system to exceed 500 TPS on a dual-processor machine. Table 3 gives our results; the audits were performed by KPMG Peat Marwick. The mean qualified throughput (MQTh) is the transaction rate at steady state, and $K/tpsA is a measure of the price per transaction for the hardware and software configuration.

Table 3 DEC Rdb TPC-A Benchmarks

Processor	Cycle Time	MQTh	$K/tpsA
DEC 7000 AXP Model 610	5.5 ns	302.68	$6,643.00
DEC 7000 AXP Model 610	5.0 ns	327.99	$6,749.00
DEC 7000 AXP Model 620	5.0 ns	527.73	$6,431.00

Anatomy of the TPC-A Transaction

To understand how DEC Rdb achieves such fast transaction rates, we need to examine the effects of the optimizations to the software with respect to the execution of the TPC-A transaction.

To complete the four updates required by the TPC-A transaction, DEC Rdb actually incurs two physical I/O operations and two lock operations. More specifically, the branch and teller records are

located on a page that is cached in the database buffer pool; therefore, these two records are updated extremely fast. The update to the account record causes the account page/record to be fetched from disk and then modified. A history record is then stored on a page that also remains in the buffer pool.

Note that there are too many account records for all of them to be cached in the buffer pool. Hence, the account fetch is the operation that causes pages to cycle through the buffer pool and eventually be flushed to disk. The ABW protocols described previously are utilized to write groups of account pages back to disk asynchronously. (The branch, teller, and history pages never approach the end of the LRU queue.)

In regard to locking, the branch, teller, and history pages/records are all governed by locks that are carried over from one transaction to the next. There is no need to incur any locks to update these records. The account page/record, which must be fetched from disk, requires a new lock request.

At commit time, the after images of the record modifications are submitted to the AIJ log. The new protocols allow the user process to "sleep," while the ALS process flushes the AIJ records to the AIJ file. After the ACE I/O has completed, which occurs before the I/O to the AIJ file on disk, the user processes are "woken up" to begin processing their next transactions.

As shown in Figures 9 and 10, the transaction is dominated by stall times. Since the AXP processors are so fast, the branch, teller, and history updates, and the two locks are a very small fraction of the transaction duration. The synchronous account read is a big expense. The batched asynchronous account writes are interesting. Indeed, each transaction requires one account read, which then causes one account write since the buffer pool overflows. Because the account writes are batched into groups and written asynchronously, however, there is no stall time required in the path of the transaction.

[Figure 9 (Transaction Duration before Modifications) is not available in ASCII format.]

[Figure 10 (Improved Transaction Duration due to Performance Modifications) is not available in ASCII format.]

Another critical performance metric in the TPC-A benchmark is response time. Figure 11 shows the response times with an average of about 1 second. Figure 12 shows the response time during the steady-state period of the TPC-A experiment.

[Figure 11 (Response Time versus TPS) is not available in ASCII format.]

[Figure 12 (TPC-A Measurement at Steady State) is not available in

ASCII format.]

Performance Profiling of DEC Rdb

In this section, we describe in more detail the instruction profile (i.e., instruction counts, machine cycles, and processor modes) generated during the TPC-A tests.

Performance profiling of DEC Rdb was obtained using Digital's IPROBE tool, the RMU, and the VMS performance monitor. IPROBE is an internal tool built to capture information from the two processor counters that were established to count on-chip events and interrupts after a threshold value was reached.

The TP1 benchmark, a back-end-only version of the TPC-A benchmark, was used to measure DEC Rdb performance on a DEC 7000 AXP Model 610 configured with 5 KDM70 disk controllers and 20 RA70 disk devices. We relied on the IPROBE tool primarily to generate PC (program counter) sampling and to track transaction path length variations as new features were prototyped and performance optimizations were added. We also used IPROBE to verify cache efficiency and the instruction mix in TP applications. We used the RMU to measure the maximum throughput of the TP1 benchmark, the database and application behaviors.

Transaction Cycles Profile

We conducted experiments to determine the performance of TP1 transactions. We used the DEC 7000 AXP Model 610 system (with a 5.5-nanosecond processor) and the OpenVMS AXP version 1.5 operating system. With this configuration, DEC Rdb version 6.0 software achieved a maximum throughput of 334 TPS for the TP1 benchmark. More than 95 percent of the transactions completed in less than 1 second.

Table 4 gives the distribution of the cycles per transaction and the cycles in PALcode in the various CPU modes. The cycle count per TP1 transaction was measured at 544,000 cycles. PALcode calls represent approximately 13 percent of the overall cycle count. In measurements taken with early OpenVMS AXP base levels, PALcode calls represented 28 percent of the overall cycle count. The most frequently called PALcode functions were misses in the data and instruction translation buffers. Four major DEC Rdb images may now be installed "/RESIDENT" to take advantage of granularity hints and reduce misses in the instruction translation buffer. Dual issuing remained low throughout the experiments we conducted with various DEC Rdb and OpenVMS AXP base levels.

Table 4 Transaction Cycles in CPU Modes

	% Cycles in System Modes	

	Cycles per Transaction	Interrupt	Kernel	Executive	User
Cycles	544.0k	7.1	14.0	73.0	5.8
PALcode cycles	71.5k	13.0	18.0	61.4	7.3
Dual issues	19.0k	5.1	11.6	79.0	4.0

TP1 Transaction Path Length Profile

At the initial stage of the DEC Rdb port to the Alpha AXP platform in January 1993, the TP1 transaction path length was measured at 300,000 reduced instruction set computer (RISC) instructions with the available OpenVMS AXP base levels. In April 1993, the TP1 transaction path length dropped to 189,000 RISC instructions. As shown in Table 5, the TP1 transaction path length is currently at 133,500 RISC instructions. That measurement is 30 percent less than it was in April 1993. The cycles per instruction were measured at 3.8.

Table 5 TP1 Transaction Path Length

			% Time in CPU Modes			
MQTh	Transaction Path Length	Cycles per In-struction	Interrupt	Kernel	Executive	User
334 TP1 TPS	133.5k	3.8	4.8	11.3	79.9	3.4

SUMMARY

Based on current performance and future trends, the Alpha AXP family of processors and platforms will provide superb servers for high-end production systems. To keep pace with the phenomenal increases in CPU speed, database systems must incorporate features that reduce the I/O bottleneck. DEC Rdb version 6.0 software has incorporated a number of these features: asynchronous page fetch, asynchronous batch write, multithreaded backup and restore, multi-statement procedures, and AIJ log server using electronic caches. These enhancements not only allow optimal transaction processing performance but also permit systems to deal with very large data sets.

ACKNOWLEDGMENTS

The development of DEC Rdb V6.0 was a team effort involving more people than can be acknowledged here. We would, however, like to

recognize the significant contributions of Jay Feenan, Richard Pledereder, and Scott Matsumoto for their design of the multi-statement procedures feature of DEC Rdb and of Ed Fisher for his work on the Rdb code generator.

REFERENCES

1. L. Hobbs and K. England, Rdb/VMS: A Comprehensive Guide (Burlington, MA: Digital Press, 1991).

2. R. Sites, "Alpha AXP Architecture," Digital Technical Journal, vol. 4, no. 4 (1992): 19-34.

3. J. Ousterhout and F. Douglas, "Beating the I/O Bottleneck: A Case for Log-Structured File Systems," Technical Report, University of California at Berkeley (1988).

4. J. Coffler, Z. Mohamed, and P. Spiro, "Porting Digital's Database Management Products to the Alpha AXP Platform," Digital Technical Journal, vol. 4, no. 4 (1992): 153-164.

5. D. Lomet, R. Anderson, T. Rengarajan, and P. Spiro, "How the Rdb/VMS Data Sharing System Became Fast," Technical Report CRL 92/4, Digital Equipment Corporation, Cambridge Research Laboratory (1992).

6. P. Spiro, A. Joshi, and T. Rengarajan, "Designing an Optimized Transaction Commit Protocol," Digital Technical Journal, vol. 3, no. 1 (Winter 1991): 70-78.

7. E. Knuth, Sorting and Searching, The Art of Computer Programming (Reading, MA: Addison-Wesley Publishing Company, 1973).

8. C. Nyberg, T. Barclay, Z. Cvetanovic, J. Gray, and D. Lomet, "AlphaSort: A RISC Machine Sort," Technical Report 93.2, Digital Equipment Corporation, San Francisco Systems Center (1993).

9. J. Gray, The Benchmark Handbook (San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1993).

TRADEMARKS

TPC-A is a trademark of the Transaction Processing Performance Council.

BIOGRAPHIES

Lucien A. Dimino A principal software engineer, Lucien Dimino is a member of the Database Systems Group. He is responsible for the RMU relational database management utility. Formerly with AT&T Bell Telephone Laboratories, he joined Digital in 1974. At Digital he has worked in the areas of networking and communications, transaction processing, manufacturing productivity and automation, and database management. Lucien received a B.S. in mathematics (1966) from City College of New York and an M.S. in mathematics from Stevens Institute of Technology.

Rabah Mediouni Rabah Mediouni joined Digital in 1980 and is currently a principal engineer in the Rdb Consulting Group. His responsibilities include consulting on database design and tuning and performance characterization of new features in major Rdb releases. He was responsible for the performance characterization effort during the DEC Rdb port to the AXP platforms. Prior to this, Rabah worked in the Mid-range Systems Performance Group as a primary contributor to the VAX 8000 series performance characterization project. Rabah received an M.S. in computer science from Rivier College in 1985.

T. K. Rengarajan T. K. Rengarajan, a member of the Database Systems Group since 1987, works on the KODA software kernel of the DEC Rdb system. He has contributed in the areas of buffer management, high availability, OLTP performance on Alpha AXP systems, and multimedia databases. He designed high-performance logging, recoverable latches, asynchronous batch writes, and asynchronous prefetch features for DEC Rdb version 6.0. Ranga holds M.S. degrees in computer-aided design and computer science from the University of Kentucky and the University of Wisconsin, respectively.

Michael S. Rubino Michael Rubino joined Digital in 1983 and is a principal software engineer in the Database Systems Group. As the Alpha program manager for database systems, his primary role is the oversight of the porting of Rdb as well as other database products from VAX to Alpha AXP. Prior to this work, Michael was with VMS Engineering and contributed to the RMS and RMS/journaling projects. Prior to that, he was the KODA (database kernel) project leader. Michael received a B.S. in computer science from the State University of New York at Stony Brook in 1983.

Peter M. Spiro Peter Spiro, a consulting software engineer, is currently the technical director for the Rdb and DBMS software

product set. Peter's current focus is very large database issues as they relate to the information highway. Peter joined Digital in 1985, after receiving M.S. degrees in forest science and computer science from the University of Wisconsin-Madison. He has five patents related to database journaling and recovery, and he has authored three papers for earlier issues of the Digital Technical Journal. In his spare time, Peter is building a birch bark canoe.