50,000 Users on an Oracle8 Universal Server Database

Tirthankar Lahiri, Oracle Corporation, tlahiri@us.oracle.com
Ashok Joshi, Oracle Corporation, ajoshi@us.oracle.com
Amit Jasuja, Oracle Corporation, ajasuja@us.oracle.com
Sumanta Chatterjee, Oracle Corporation, schatter@us.oracle.com

Abstract

In this paper, we describe the Oracle Large User Population Demonstration and highlight the scalability mechanisms in the Oracle8 Universal Data Server which make it possible to support as many as 50,000 concurrent users on a single Oracle8 database without any middle-tier TP-monitor software. Supporting such large user populations requires many mechanisms for high concurrency and throughput. Algorithms in all areas of the server ranging from process and buffer management to SQL compilation and execution must be designed to be highly scalable. Efficient resource sharing mechanisms are required to prevent server-side resource requirements from growing unboundedly with the number of users. Parallel execution across multiple systems is necessary to allow user-population and throughput to scale beyond the restrictions of a single system. In addition to scalability, mechanisms for high availability, ease-of-use, and rich functionality are necessary for supporting complex user applications typical of realistic workloads. All mechanisms must be portable to a wide variety of installations ranging from desk-top systems to large scale enterprise servers and to a wide variety of operating systems.

1. Introduction

Network Computing Architecture (NCA) is an emerging computing paradigm in which large user communities use lightweight client applications to access centralized information repositories on the network. This architecture is designed to provide organizations and individuals with reliable and efficient access to vast quantities of information over local and wide area networks. At the Oracle8 inauguration event on June 24, 1997, Oracle Corporation demonstrated 50,000 simulated concurrent users connected to a single Oracle8 database. This system ran a corporate messaging application and sustained a throughput of 11,000 email messages per minute, equivalent to 15.8 million per day. The large population and high throughput of this demonstration proves that customized solutions are not required for deploying large-scale centralized information networks: the Oracle8 Universal Data Server running on off-the-shelf hardware provides all of the necessary scalable and reliable infrastructure for Network Computing.

2. Description of Demonstration

The client application used in the demonstration was the multi-threaded Oracle8 InterOffice 4.0.5 messaging application. This system uses Oracle8 tables for storing and retrieving messages and executes approximately 10 SQL statements for each email message sent and received. InterOffice therefore has the characteristics of a typical high-concurrency OLTP application. This workload was designed to represent an average day's email activity of a large enterprise. No TP-monitor software was used between InterOffice and the Oracle8 database.

An Oracle load generator program was used to simulate InterOffice users on client workstations. Each simulated user independently invokes the InterOffice API to send twelve 1K messages to three other users, and receives a total of thirty six messages. The InterOffice library in turn issues Oracle SQL and PL/SQL requests to an Oracle8 Parallel Server database running on a 2 node SUN Ultra-Enterprise cluster. A total of 8 SUN Ultra-2 client workstations was used, each driving 6250 InterOffice clients for a total of 50,000 users. The average message throughput of the system over the 30 minutes of the demonstration was 11,000 messages per minute and the peak observed throughput was 15,000 per minute. Table 1 below summarizes the key attributes of the demonstration:

Number of concurrent users	50,746
Maximum per minute message rate	15,000
Sustained per minute message rate	11,133
Size of message body of each message	1KB
SQL stmts executed per message sent	~10
SQL stmts executed per message rcvd	~10
Messages sent by each user	12
Messages rcvd by each user	36

Table 1: Attributes of large user population benchmark

3. Scalability Mechanisms

In this section, we highlight some of the important scalability mechanisms of the Oracle8 Universal Data Server. These mechanisms are mentioned in approximate order of importance to the large population and throughput of the demonstration.

The following list enumerates some of the common design philosophies underlying these mechanisms:

- *Clustering:* The architecture is designed to be extendable from a single system to a shared-disk cluster of multiple systems so that user-population and throughput are not limited by the capabilities of a single system.
- *Efficient resource sharing:* Resource management mechanisms have been designed so that resource requirements do not grow unboundedly with the user population. Processes are shared and data structures are cached and reused whenever feasible so that server-side resource utilization is maximized.
- *Partitioning and concurrency:* Algorithms have been designed to allow large numbers of concurrent operations. Wherever possible, frequently accessed data-structures have been partitioned, and single *latches* (lightweight mutual exclusion primitives) decomposed into multiple latches to minimize contention.

- *Layering:* The Oracle Universal server has a layered architecture with well-defined interfaces for buffer management, transaction management, SQL execution, etc. A layered architecture facilitates portability, ease of development, and allows enhancements to different layers to be performed independently.
- *Portability:* All the mechanisms mentioned in this paper are platform independent, and a *Virtual Operating System* (VOS) module within the Oracle Universal Server provides an efficient and generic interface to almost any Operating System and hardware platform. The VOS module encapsulates Operating System services such as file I/O, inter-process communication, memory management, and mechanisms for mutual exclusion. This allows convenient development of platform independent code in almost all areas of the server, since platform specific details are "hidden" by the VOS layer.

3.1 Oracle Parallel Server

The *Oracle Parallel Server* (OPS) mechanism transparently extends the capabilities of the Oracle Universal Server from single systems to shared-disk clusters. The OPS mechanism was of primary importance to the large-user population demonstration in which the 50,000 concurrent user community was achieved using a two-node cluster with 25,000 users on each node.

Each node in an OPS cluster runs a separate *instance* of Oracle - a self-contained group of Oracle server processes with their own buffer pool, a global shared memory region known as the *Shared Global Area* (SGA), and a set of redo log files collectively known as a *thread* of redo. The different instances use a *distributed lock manager* (DLM) to synchronize access to the shared datafiles constituting the database and to maintain buffer-pool coherency between the instances.

An OPS configuration is depicted in Figure 1. The Parallel Server mechanism is described in further detail in [2].

Since each instance has its own independent thread of redo, the OPS configuration partitions redo generation between instances and avoids bottlenecks in writing to the redo log in write-intensive workloads. Contention between instances can be further reduced by using the *free-list group mechanism* which allows free space in the database to be partitioned between instances. Partitioning free-space between instances reduces contention for free-space allocation when the workload has a high volume of concurrent operations requiring new space, such as INSERT operations. With all these mechanisms, OPS potentially allows throughput and population to scale linearly with the number of nodes in the cluster. Parallel Server also improves the availability of an Oracle database since the database continues to be available as long as at least one node in the cluster is alive.

3.2 Multi-Threaded Server

The *Multi-Threaded Server* (MTS) mode of operation is an example of an efficient resource-sharing mechanism: it allows a large number of client processes to connect to an Oracle8 database and share a fixed (and smaller) pool of server processes. This feature was also crucial to the demonstration: with the MTS configuration, it was possible to use 200 shared-server processes on each node to service the requests of 25,000 connected users. Figure 1 depicts an MTS configuration for Oracle instances in an OPS cluster.

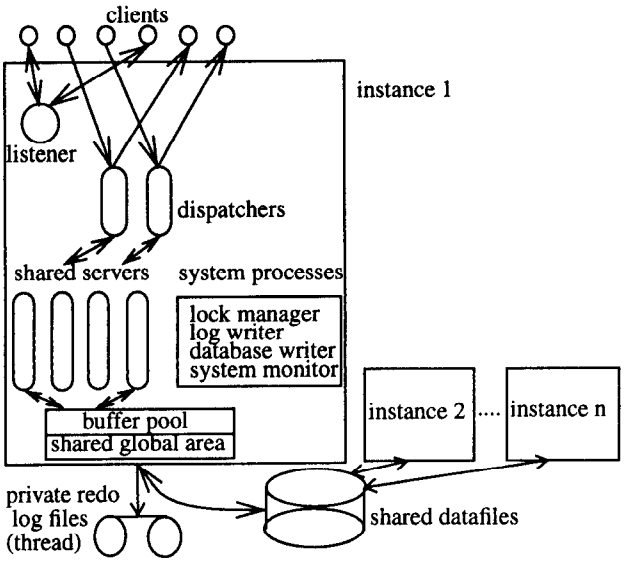

Figure 1. An OPS cluster with each instance in MTS mode

In most OLTP workloads, only a fraction of the clients are simultaneously active and a server process permanently dedicated to a client would be largely idle. MTS mode is effective in minimizing wasteful server-side idle time as well as in limiting the number of server processes required to support a large user population.

Three types of processes are involved in the MTS configuration: a *network listener* process, one or more *dispatcher* processes, and one or more *shared server* processes. The listener waits for connection requests from client processes and hands each client the address of one of the dispatchers. Each client request is queued by the client's dispatcher in a request queue located in the server's shared memory region, known as the *shared global area* (SGA). The next available shared-server process handles the request and queues the result in the response queue of the dispatcher. The dispatcher then returns the completed request to the client. In this way, client processing is multiplexed across a fixed set of server processes. Dispatchers process requests and responses in batches to avoid performing too many context switches. Distributing the client processes between multiple dispatchers avoids bottlenecks in request submission and response retrieval with large user populations. Details on the MTS mechanism can be found in [3].

3.3 Scalable Buffer Manager

The *buffer manager* component allows large numbers of processes to concurrently access disk block buffers in the buffer pool and was an important contributor to the high throughput (11,000 messages per minute) of the demonstration. The buffer manager uses a proprietary approximate-LRU replacement algorithm for sustaining high hit-rates in the buffer cache, an efficient checkpointing algorithm to limit crash-recovery times, a dedicated *Database Writer* (DBWR) process to achieve high throughput for writing data blocks, and a dedicated *Log-writer* (LGWR) process for sustaining high redo-log write-throughput. The buffer manager has been described in detail in [1].

The buffer manager achieves scalability by partitioning frequently accessed data structures and minimizing critical section code. The most frequently accessed buffer manager data structures are the *Buffer Hash Table* which is used to find buffers by *database block*

address (DBA), the *LRU replacement queue* of buffers which orders the buffers by recentness of usage, and the *redo buffer* - a circular queue of redo entries which are written out in FIFO order by LGWR. Each hash bucket in the buffer hash table is controlled by a distinct *hash latch*, allowing multiple processes to concurrently search for buffers whose DBAs hash to distinct values. The LRU list is also partitioned into multiple independent lists each of which is controlled by a distinct *LRU latch* to allow simultaneous list reordering by different processes. Lastly, multiple *redo-copy latches* guard access to the redo buffer entries, so that multiple processes making changes to data buffers can simultaneously copy the redo for their changes into their respective redo buffer entries. The number of hash buckets in the hash table, the number of lru replacement lists, and the number of redo-copy latches are configurable and can be tailored to suit the concurrency requirements of a workload.

3.4 Scalable concurrency control mechanisms

The locking mechanisms in the Oracle8 Universal Server for achieving transaction isolation and serializability are designed to allow large numbers of concurrent operations. and were also a vital component of the high throughput of the demonstration.

Row-Level locking is used by update operations to lock data in a table at the granularity of an individual row. It allows different transactions to concurrently modify distinct rows in the same table even if the rows are stored in the same physical disk block. This mechanism allows high update concurrency since an update of a row has to wait for another transaction to commit or abort only if that row is currently being modified by the second transaction. Oracle uses a proprietary block format to implement row-level locking and therefore allows *unlimited row-level locking*: since row-locks are represented by state within the blocks themselves and do not require any in-memory lock data structures, there is no limit to the total number of row locks that can be acquired within an instance.

The *Consistent Read* (CR) mechanism is a version based concurrency control protocol which allows transactions to read blocks without acquiring any locks. Each transaction is associated with a snapshot time, known as the *System Change Number* (SCN) and the CR mechanism guarantees that any data read by a transaction is transactionally consistent as of that SCN. When a transaction performs a change to a block, it stores the information required to undo that change in a *rollback segment*. The CR mechanism uses the stored undo information to create an earlier version of the block (a *clone*) which is consistent as of the reading transaction's SCN. Clones are created in-memory and are never written to disk. A read operation therefore never needs to wait for another transaction to commit or abort - the CR mechanism automatically reconstructs the version of the block required by the operation. This mechanism therefore allows high concurrency for read operations.

3.5 Shared Compilation Areas

Shared compilation areas reduce server-side memory requirements as well as query-processing times. This mechanism is important for large user populations since it prevents memory and processing requirements of SQL compilation and execution from growing explosively with the size of the population.

The query compilation component caches the parse tree and execution plan for every SQL statement and PL/SQL package in a *shared SQL area* so that similar statements can use the cached plans. Since many users in typical OLTP environments execute the same application, this *library cache* is effective in reducing the total number of compilation and optimization operations required for large OLTP workloads. The library cache is also effective in reducing the memory requirements of SQL compilation since only one query plan needs to be stored for many similar operations. Multiple latches guard access to the library cache to prevent contention on a single latch.

Each user also requires a *private SQL area* for storing runtime information during query processing. To prevent private SQL area memory from growing linearly with the number of users, the *serial reuse* mechanism allows a fixed pool of global (SGA) memory to be shared by the users for storing their private SQL areas. Apart from limiting memory consumption, serial-reuse also reduces the number of memory allocation and deallocation operations required during query processing, with significant performance benefits for large user populations.

4. Summary and Conclusions

To summarize, the *Oracle Parallel Server* mechanism allows a user population to scale beyond the limits of a single system: the two node cluster used in the demonstration was able to achieve a population twice as large as could be sustained by any one of the Enterprise-Server nodes, the *Multi-Threaded Server* mechanism allows a fixed number of shared-server processes to service requests from a much larger number of clients thereby minimizing server-process idle-time and limiting the number of processes that need to be scheduled on a node, the *Scalable Buffer Manager* allows a large number of server processes to simultaneously access disk block buffers and redo-log buffers, the *Row-level Locking* and *Consistent Read* mechanisms for isolation and serializability minimize conflicts between transactions and allow high transaction throughput, and finally, efficient memory management techniques such as *Shared SQL Areas* and *Serial Reuse* of private SQL area memory reduces per-user memory requirements for query compilation and query execution. All these mechanisms collectively allowed the Oracle8 Universal Server to *natively* support 50,000 user connections without any external TP-monitor software.

In addition to mechanisms for scalability and throughput, the Oracle8 Universal Server also has rich functionality for supporting diverse workloads and large data volumes expected in the Network Computing environment. *Binary large objects* (LOBs), *abstract datatypes*, and *objects* enable complex applications which store and process a rich variety of data in an Oracle8 database. The *Range Partitioning* mechanism supports very large volumes of data by decomposing large tables and indexes into smaller and more manageable partitions. System management is simplified by the *Recovery Manager* module which automates the tasks of periodic backups, checkpointing, and recovery.

All these mechanisms collectively make it possible for the Oracle8 Universal Server to support realistic large-scale network computing workloads across a diverse range of platforms.

5. References

[1] W.Bridge, A.Joshi, M.Keihl, T.Lahiri, J.Loaiza, N. Macnaughton "The Oracle Universal Server Buffer Manager" *Proceedings of the VLDB conference*, August 1997.

[2] B.Klots, S.Chatterjee "Cache Coherency In Oracle Parallel Server" *Proceedings of the VLDB Conference*, September 1996.

[3] Oracle Corporation Part Numbers A54646-01 and A54644-01 *Oracle8 Server Concepts Volume I and II*: June 1997