



Designing & Implementing Near Real Time Applications

An Oracle Technical White Paper

July 2000

INTRODUCTION

In recent years Near Real Time Systems (NRTS) have become increasingly more important in our society. These systems include, telephone networks, online stock quotes and entertainment software such as games and graphic animation.

NRTS are set apart from general software by their strict requirements for latency, throughput, reliability and availability. To quote a well known example: the telephone network. In the case of the telephone network, near real time systems are used to translate 1-800 toll free numbers into actual area codes and phone numbers. When a 1-800 number is dialed, the local exchange switch recognizes the number is a toll free phone number which needs to be translated. A request is then sent from the local exchange to the Service Control Point (SCP), asking for the translation. One of the SCP slave processes queries the database for the number translation and the network routing, and sends them back to the local exchange. The exchange then connects the caller with the actual number.

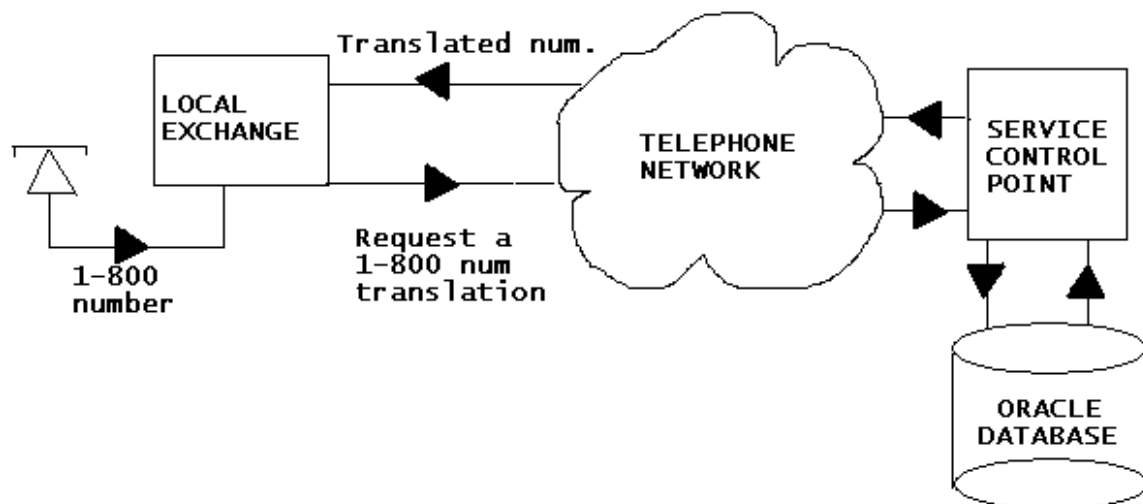


Figure 1 What happens when you dial a 1-800 telephone number.

When dialing a 1-800 number, people expect an immediate connection. In reality that translates into a 300 millisecond delay before the connection is established. This means all necessary actions required to make a connection must be complete within 300 milliseconds including network message transmissions, message queuing and database transactions. Based on the current speed of the telephone networks this requires the near real time application at the SCP to complete each transaction in less than 25 milliseconds.

Ten years ago the hardware components of a NRTS were too slow to allow developers the luxury of using any interpreted programming languages. Instead the systems were coded in assembly languages and used proprietary databases. By their very nature, these systems were extremely difficult to maintain and extend, as most changes required a complete rewrite of the system.

Due to the phenomenal performance improvement in the hardware components used for these systems, developers are now free to use third or fourth generation languages, and off the shelf software and hardware.

In this paper, the design issues faced by the NRTS developers of today will be discussed, with an emphasis on performance. Several possible solutions will be proposed and the pro's and con's for each will be discussed.

HARDWARE

One of the most likely hardware configurations for NRTS comprises of 3 main components (Figure 2).

- A front end which may be a computer terminal or a telephone switch, or any other appliance.
- Middle tiers, possible application servers or queue managers.
- A highly available, scalable database back end server.

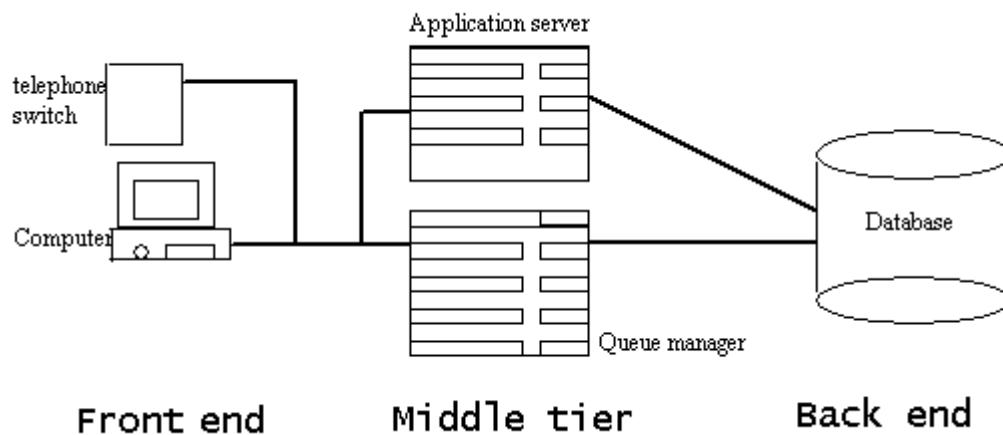


Figure 2 The 3-tier hardware configuration for near real time systems.

When using this N-tier architecture there are three main elements that dictate the performance of the system:

- CPU
- Memory
- Network (Bandwidth & Latency)

Each of these is a potential bottleneck.

CPU

The speed of the processors in the middle tier and database servers has always been a bottleneck on NRTS. In the past, the problem was overcome by adding as many processors as possible to the servers, in order to increase the number of transactions executed per clock cycle. In recent years the power of processors have more than doubled every 18 months. (See figure 3)

The size and complexity of a NRTS has been greatly reduced by the improvements in processor power. It is now possible to run the database back end on fewer processors. The number of middle tier servers has also been reduced as each individual server can now handle more load.

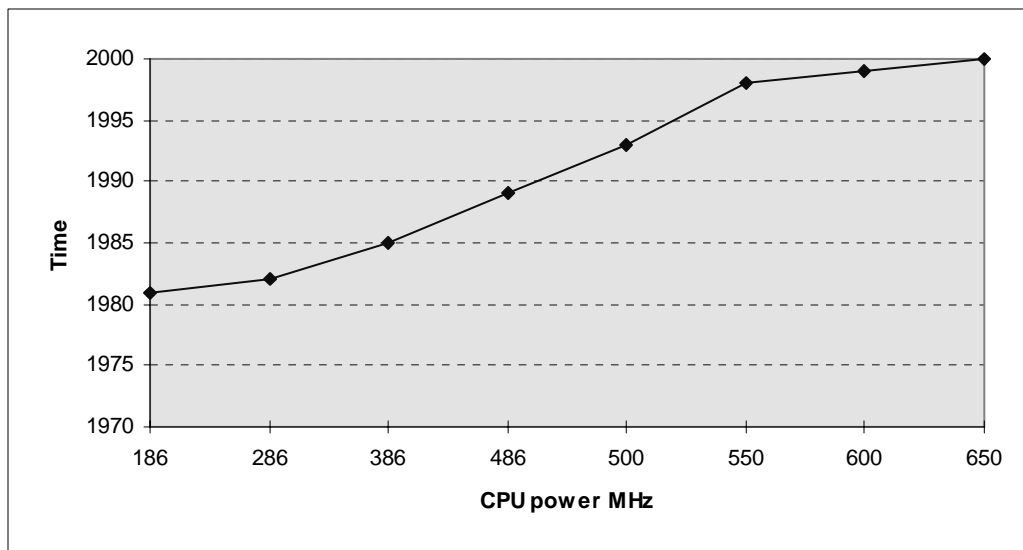


Figure 3 The increase in CPU power V's time.

MEMORY

On average each database transaction in a NRTS must complete in less than 25 milliseconds. A random access disk I/O can vary from 10 to 50 milliseconds so it is imperative that all data required to satisfy queries reside in memory, as memory accesses take only nanno-seconds.

In previous years the average amount of RAM in a machine was less than 100 Mb. NRTS designers were faced with the two choices; have a very small in memory database, or consider having some of the data on disk. Many chose to have the data on disk using flat files and embedded the schema of the data into their assembly code. In order to ensure they met the time requirements, the code actually contained the bit offset for specific data on disk. Using this approach made it virtually impossible to maintain or extend these systems.

The cost of RAM has dramatically reduced over the last ten years making it more feasible than ever before to have all the necessary data reside in memory. This allows developers to use an 'off the shelf' database to manage the schema. Using a database reduces the complexity and increases the extendibility and manageability of the software required for an NRTS, as the systems no longer have to know the offset of data on disk.

NETWORK

Bandwidth and latency are the two main factors governing the performance of a network. Originally the problem which faced NRTS developers was that the bandwidth was too small. Most of the time taken to process a users request was used to pass messages back and forth across the network. One way to reduce the amount of data transferred was to compress the data, so that there was physically less bytes to move. However this option was a trade off between CPU processing power and time spent in the network, as the message had to be compressed and uncompressed at each end.

As bandwidths started to improve a second problem was discovered in the network section of the system; latency. Although advertised bandwidths had more than doubled, it became apparent that the effective bandwidth was substantially less due to slow servers, inefficient data packing, excessive network hopping and high loads. Every effort has been made to remove latency from networks but it has not been completely eliminated.

The Internet backbone as a whole has very good latency. Take the example of trying to send a message from San Francisco to Boston.

- The distance from San Francisco to Boston is 4320km.
- The speed of light in a vacuum is 300×10^6 m/s.
- The speed of light in fiber is roughly 66% of the speed of light in a vacuum.
- The speed of light in fiber is 300×10^6 m/s * 0.66 = 200×10^6 m/s.
- The one-way delay to Boston is $4320\text{km} / 200 \times 10^6$ m/s = 21.6 ms.
- The round-trip time to Boston and back is 43.2ms.
- Currently a ping from San Francisco to Boston over the Internet takes 85 ms.

This example shows that Internet hardware can achieve within a factor of two of the speed of light however it won't ever get to the speed of light. The reason is network latency, be it due to bottlenecks, hardware problems or just high load.

Even if bandwidth was infinite and latency was negligible, applications still would not be able to take full advantage of network speeds up. In order for an application to communicate with another application via a network, the message must pass through many layers of user and kernel software before reaching the network. Once it reaches it's destination the message is again passed through many layers of user and kernel software before it is received by the other application. (See figure 4)

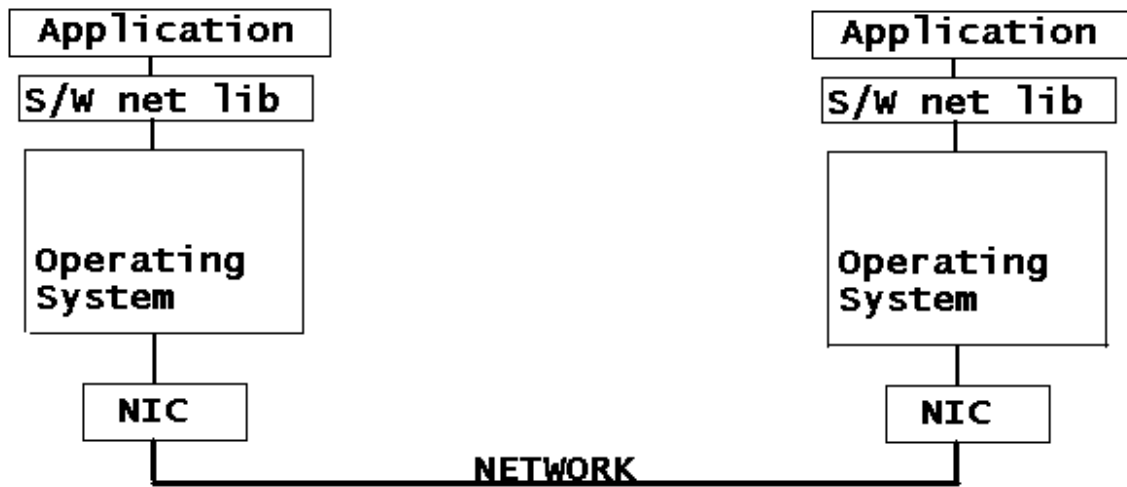


Figure 4 The levels of software a message passes through to get to the network.

A new architecture, Virtual Interface (VI) has been developed that significantly reduces the software overhead between an application and the network. VI, in conjunction with special interconnect hardware provides a mechanism to transfer data directly from an application running on a local node to another application running on a remote node without any intermediate buffering. This is achieved by allowing a user process to bypass the operating system in a fully protected fashion and access the network interface directly. (See Figure 5)

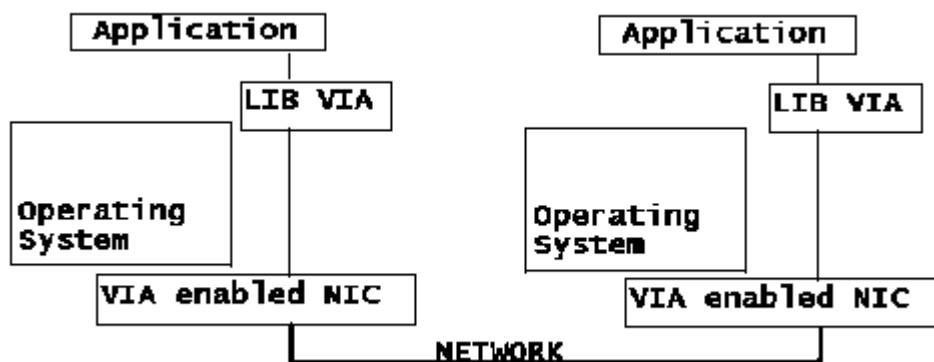


Figure 5 The path a message would take when using the VI architecture.

Bypassing the operating system and removing the need for intermediate copies of the data results in high throughput rates, low latency and very low CPU utilization.

SOFTWARE

There are several unique challenges when developing near real time software. The requirements for latency, throughput, reliability and availability are far more stringent than for general purpose software. Understanding the impact of design decisions and effective communication functionality is critical. If a design technique was to be followed blindly, it is possible that the application generated could be so thick with code that it will never meet the requirements. When developing a NRTS it is sometimes necessary to compromise the features of the design methodology (e.g. Object Orient design) in order to meet the performance and availability requirements.

Near real time applications are becoming increasingly more complex as this technology is incorporated into new and evolving industries such as online interactive computer games. It is also becoming a more competitive market with the amount of time allocated to pushing a product to market becoming shorter. These two factors have forced developers to move away from proprietary systems and use off the shelf hardware and software components instead.

One of the most common off the shelf components used in today's NRTS is a DBMS, such as Oracle. When designing the database schema for an NRTS, every effort should be made to minimize transaction response times as well as the variance of the response time.

DATABASE SCHEMA DESIGN OPTIONS

Typically, query performance is improved by creating an index on each of the tables accessed by the query. However to find a row stored in an indexed table pinned in memory, at least two block accesses are required. One or more to find the key value in the index, and another to read the row from the table. An alternative approach to improving query performance would be to hash *****

use Hash clusters. Oracle physically stores the rows of a table in a hash cluster and retrieves them by computing the data location from a hash function. Thus only a single block access is required, as Oracle uses the hash function to locate the block that has the row and performs a single block access to read the row.

Using a hash cluster can be an advantage if:

- Most queries are equality on the cluster key e.g. *SELECT... WHERE cluster_key=.....;*
- The table or tables in the cluster are primarily static in size such that the number of rows and the amount of space required for the tables can be pre-allocated. This is important as allocating extra blocks dynamically can cause a degradation in performance.
- If the tables are often accessed by the application in join statements

Using a hash cluster is not an advantage if:

- Most queries retrieve rows over a range of cluster key values, e.g.
- *SELECT....WHERE cluster_key <*; as it causes the equivalent of a full table scan.
- A table grows without limits such that the space required for the entire table cannot be predetermined.
- An application frequently modifies the cluster key value, as it takes longer to modify a row's cluster key value than to modify the value in an unclustered table.

With Oracle 8.1.5 a second alternative has become available, Index Organized Tables(IOT's). An IOT is like an ordinary table with a primary key index on one or more of its columns, but instead of maintaining two separate objects, the database system only maintains a single B*-tree index which contains the encoded key value, and rather than having the row's rowid as the second element, it has the actual data values for the rest of the columns in the row. Table one shows the fundamental differences between an ordinary table and an IOT.

Ordinary Table	Index - Organized table
Rowid uniquely identifies a row; primary key can be optionally specified	Primary key uniquely identifies a row; primary key must be specified
Physical rowid in ROWID pseudocolumn allows building secondary indexes	Logical rowid in ROWID pseudocolumn allows building secondary indexes
Rowid based access	Primary key based access
Sequential scan returns all rows	Full-index scan returns all rows in primary key order
Unique constraint and triggers allowed	Unique constraint not allowed but triggers are allowed
Can be stored in a cluster with other tables	Cannot be stored in a cluster
Can contain a column of the LONG datatype and columns of LOBdatatypes	Cannot contain a column of the LONG datatype and columns of LOBdatatypes
Distribution and replication supported	Distribution and replication not supported

Table 1 Comparison of Index - Organized Tables with Ordinary tables.

QUERY BATCHING

NRTS typically have very simple transactions that complete in a short amount of time. Each transaction would normally consist of one, or possibly two select statements. For example:

```
Select stock_price day_low, day_high
```

```
FROM stocks
```

```
where stock_code = : st_code
```

The average response time for this type of query on a single CPU (295 MHz) UNIX machine should be approximately 0.06 milliseconds. This yields a throughput of 1,500 transactions a second. What if the query was a result of an Internet user requesting an online stock quote? The average Internet user will tolerate a 1 to 2 second delay before they expect the quote to be displayed. Given the

network latency and number of stock quote requests at high peak times, the database needs to be able to handle 10,000 transactions a second.

The throughput of the system can be improved by executing multiple user requests in a single round trip to the database. By passing an array of requests to the database which will execute the query for each element in the array, n executions can be done in one round-trip. One of the easiest way of implementing this technique is to use the UNION ALL operator. The UNION ALL operator joins multiple select statements into a single query and returns all of the rows selected by each of the statements, including all of the duplicates.

```
Select stock_price day_low, day_high
FROM stocks
where stock_code = : st_code
UNION ALL
Select stock_price day_low, day_high
FROM stocks
where stock_code = : st_code
UNION ALL
```

Although some extra processing occurs when placing the requests into an array, and also subsequently to decode the results from the returned array, the over all gain is still substantial. On a single CPU (295 MHz) UNIX machine, using an array of 10 elements the throughput for this query is approximately 3,579 transactions a second. Which is more than double the original throughput. By increasing the number of, or power of the CPU, multiple threads of the application could run on the server to achieve a throughput of 10,000 transactions a second.

MINIMIZING NETWORK AND SYSTEM OVERHEAD

Batching the database queries at the client also minimizes network traffic between the middle tier and the database; this also frees up extra system resources.

AVAILABILITY AND MAINTENANCE.

Most NRTS are 24 7 systems. Only a very short amount of down time can be tolerated e.g. 3 minutes a year, if any at all. If the system should become unavailable the cost due to lost business would be phenomenal. In order to ensure such availability many NRTS designers build in a backup strategy. There are four main options

- A standby database.
- Use multi-master replication, with each database in a different geographical area.
- Oracle's Parallel Server (OPS)
- High Availability (HA) strategy from disk array suppliers such as EMC.

STANDBY DATABASE

To use a standby database, the primary database must be archiving redo logs. All DML performed on the primary database is recorded in the archive log which is automatically sent to the standby database. At the standby database the archive log is verified and then applied. (See Figure 6)

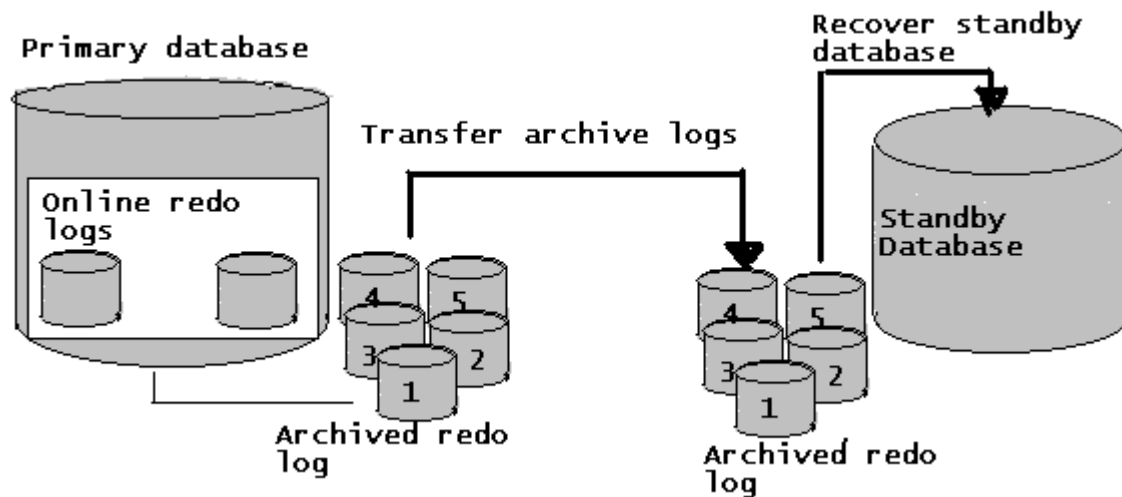


Figure 6 Transmitting and Applying Archived Redo Logs to a Standby Database

If for whatever reason the primary database goes down, the standby database can be switched in with minimum disturbance to the end user. Once activated, the standby database becomes a normal production database and loses its standby status. Another standby database must then be created.

Using the standby method, it is possible to help relieve some of the pressure on the primary database. The standby database can be converted from recovery mode to read-only mode and is then capable of handling read only queries. While the standby database is in read-only mode, none of the archive logs transferred from the primary database can be applied. Once the standby database is returned to recovery mode the backlog of archive logs can or should be applied.

MULTI-MASTER REPLICATION

If load balancing is also a main concern for the application, multi-master replication may be a better solution. With multi-master replication there are multiple copies of the database, each at a different location. The application can update any replicated table, at any location, and the change will be visible at all other sites. If a database fails at one location, it is possible to redirect all of the users accessing the database to another location.

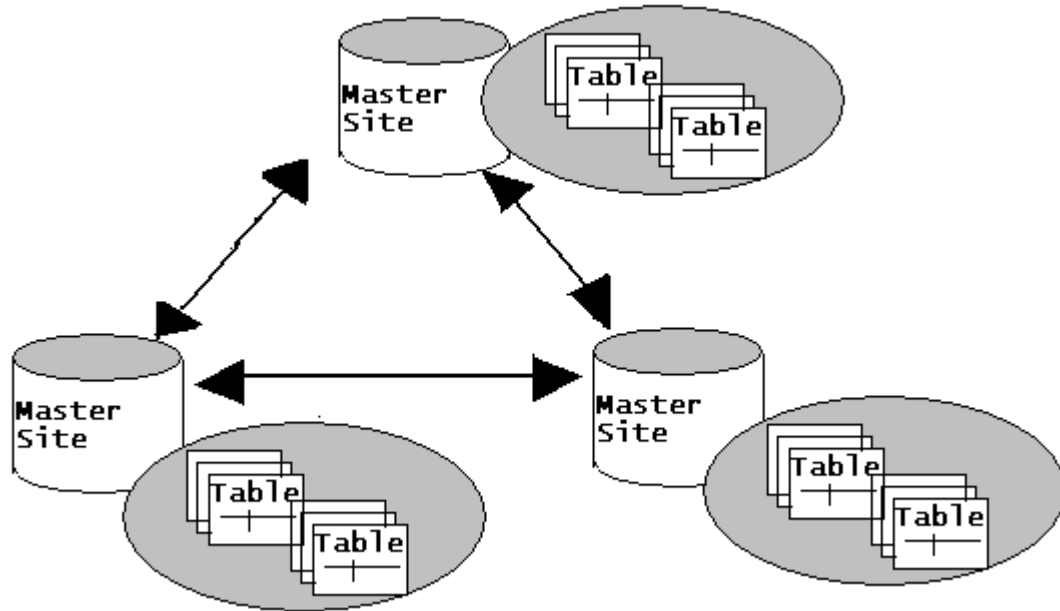


Figure 3 Multi-master replication.

With this method it is possible to distribute the load between the different locations in order to maintain a reasonable load on each database. For example, if a company had a 1-800 number for its customer care department, the company would want the customer's call to be directed to the nearest customer care center to minimize their phone bill. If the closest center was to experience an unusually high call volume or if the database at that location was down, the customer's call could be redirected to another center. The agent at the other center would still be able to access all of the customers details because each center has a complete copy of the company's database.

The ability to redirect transactions to another database provides a window of opportunity to upgrade both the hardware and software at each site in turn while still providing full availability.

If load balancing is a concern for an application but having several copies of the database at different location is not feasible then Oracle's Parallel Server (OPS) could be a solution.

ORACLE PARALLEL SERVER (OPS)

An OPS system consists of two or more Oracle instances on one or more nodes in the network. Each instance has a separate System Global Area (SGA) and set of background processes. All instances share the same datafiles and control files but each instance has its own set of redo log files. All instances can execute transactions concurrently against the same database, and each instance can have multiple users executing transactions.

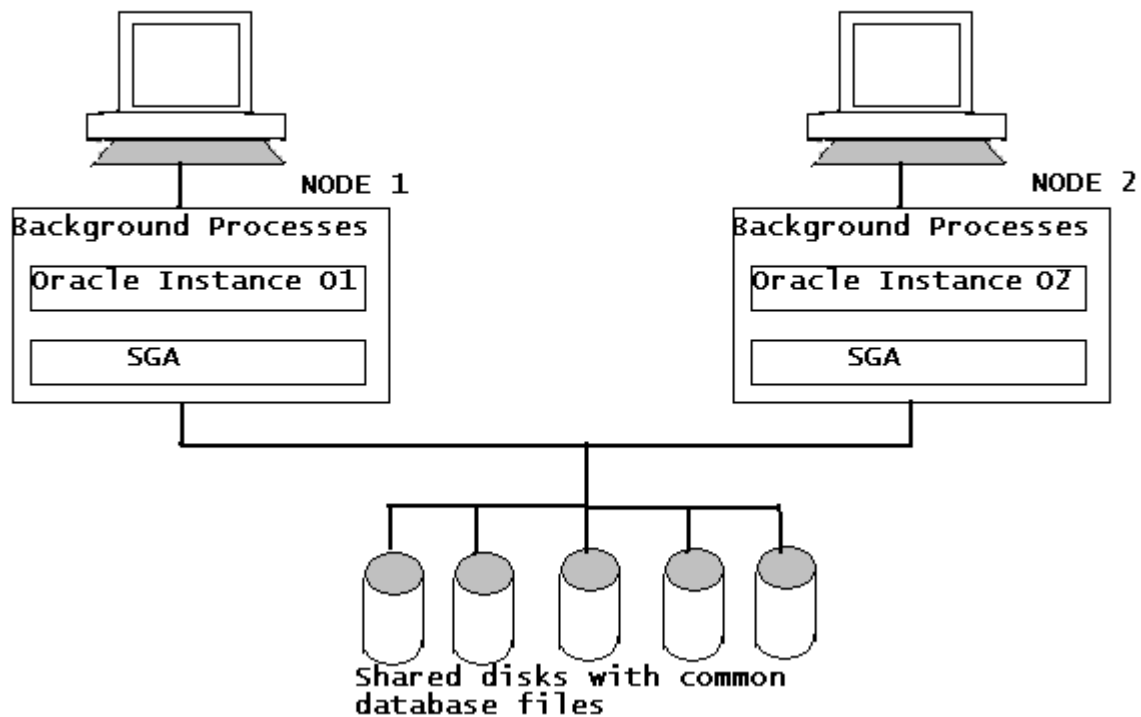


Figure 4 Oracle Parallel Server instance architecture.

If one of the instance accessing the database fails, online instance recovery is performed automatically using the online redo log files by one of the surviving instances. Instances that continue running on other nodes are not affected as long as they are reading from the buffer cache. If instances attempt to write, the transaction stops. All operations to the database are suspended until cache recovery of the failed instance is complete.

It is possible to upgrade the software and hardware of an OPS system without affecting the availability. Each node in turn should be brought down triggering an automatic online recovery. The system is now stable again minus the node to be upgraded. The upgrade can then be performed on the offline node. Once the upgrade is complete the node can be brought back on line.¹

High Availability system.

Recently more and more hardware vendors have release high availability (HA) products to further support 24 X 7 systems. Take EMC for example, they have several offerings in this section most notably the Symmetrix Remote Data Facility (SRDF). SRDF allows the functionality of mirroring (RAID 1) to occur between to physically different symmetrix systems connect via a fiber link. The mirroring is done at the logical volume level. If the source disk array goes down the target disk array can be swapped in momentarily with little or no interruption to the end user.

¹Support for different releases of Oracle within one OPS environment is operating system specific.

CONCLUSION

The demand for near real time system performance continues to rise with increasing number of web sites offering real time information at the click of a button. This increase in popularity does not however change the fundamental challenges faced by near real time system designers; how do you design an application that meets the strict response time and availability requirements but is still a maintainable and extendible piece of software?

The improvements in processor speeds, the reduction in the cost of RAM and the introduction of high speed inter-connects have all helped in paving the way for system developers to be able to use non-propriety products. They can now use third or fourth generation languages and off the shelf software and hardware components. This in turn has made the applications more maintainable as more engineers can work on the application, new functionality can be added more easily and the databases can be upgraded in order to take advantage of new features.

However, this new found freedom can sometimes be more of a hindrance than a help as developers are side tracked by different design techniques. Without strong architectural guidance it is possible to generate applications fat with code, and no comprehension of performance issues.

The key to designing near real time systems is understanding that the requirements for latency, throughput, reliability and availability are far more stringent then for general software and not being afraid to comprise design standards in order to meet the performance and availability requirements.

REFERENCES.

It's the Latency, Stupid by Stuart Cheshire, May 1996.

<http://rescomp.stanford.edu/~cheshire/rants/Latency.html>

Validation of Oracle 7.2.3 with EMC Data Manager, Symmetrix system by Paul Manning (EMC) and Jim Stone (Oracle)

ORACLE®

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

**Worldwide Inquiries:
+1.650.506.7000
Fax +1.650.506.7200
<http://www.oracle.com/>**

**Copyright © Oracle Corporation 1999
All Rights Reserved**

This document is provided for informational purposes only, and the information herein is subject to change without notice. Please report any errors herein to Oracle Corporation. Oracle Corporation does not provide any warranties covering and specifically disclaims any liability in connection with this document.

Oracle is a registered trademark, and Oracle8i, Oracle8, PL/SQL, and Oracle Expert are trademarks of Oracle Corporation. All other company and product names mentioned are used for identification purposes only and may be trademarks of their respective owners.