Best Practices for Optimizing Availability During Unplanned Outages Using Oracle Clusterware and Oracle Real Application Clusters

Oracle Maximum Availability Architecture White Paper
March 2007

Maximum Availability Architecture

Oracle Best Practices for High Availability

ORACLE

Best Practices for Optimizing Availability During Unplanned Outages Using Oracle Clusterware and Oracle Real Application Clusters

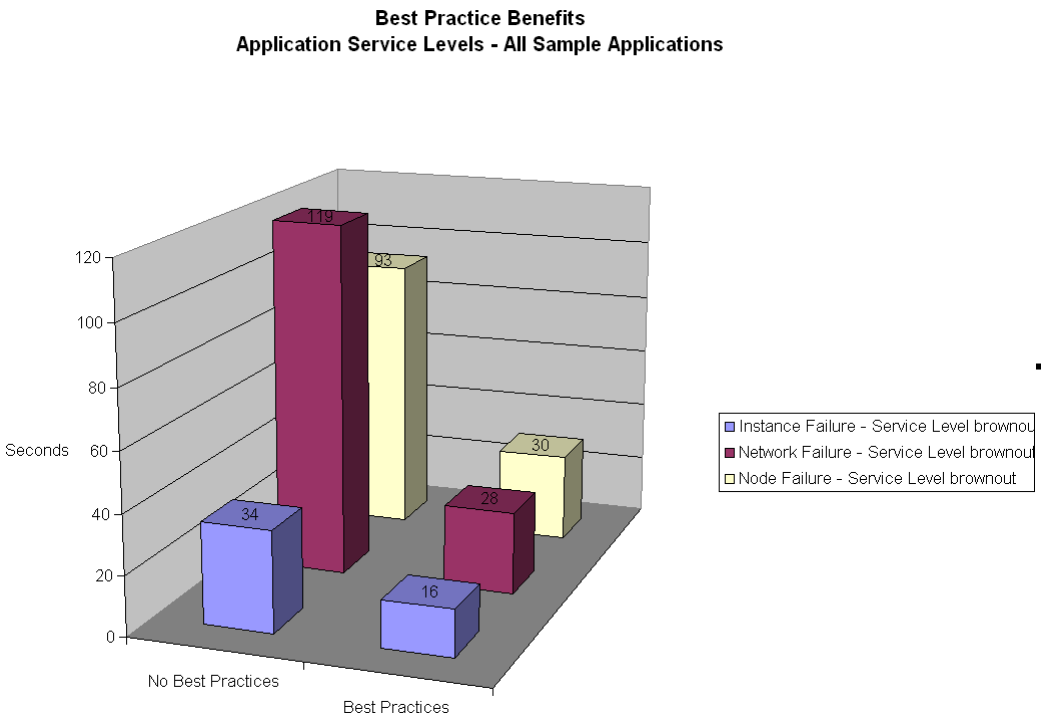
Best Practices for Optimizing Availability During Unplanned Outages Using Oracle Clusterware and Oracle Real Application Clusters

EXECUTIVE SUMMARY

Oracle Clusterware and Oracle Real Application Clusters (RAC) software version 10*g* Release 2 Patchset 2 (10.2.0.3) provide the infrastructure that enables high availability for end-user applications. The goal of this paper is to describe best practices that optimize the availability of applications supported by Oracle Clusterware and Oracle RAC during unplanned outages.

This paper explains best practices and illustrates the benefits through sample applications, with typical application service level objectives. A breakdown of components impacting availability is provided both with and without implementing the best practices. Availability of components is documented at both the server and client levels. Client level availability is measured against the defined service level objectives to understand true application service level impact.

The results of the testing done for this project show that, by implementing the best practices described in this document on default Oracle Clusterware and Oracle Database configurations, customers can expect at least a 50% reduction in impact on application availability during unplanned instance, node, and network outages. In many cases, the impact on application service level can be reduced even further to between 70% and 80%. The following chart illustrates the benefits resulting after implementing the best practices across all of the sample applications tested during this project:



Note: This paper focuses on the best practices that can reduce downtime when an unplanned outage occurs. For information about best practices you can implement to help *prevent* unplanned outages, see *Oracle Database High Availability Overview 10g Release 2* [2] and *Oracle Database High Availability Best Practices 10g Release 2* [3].

PREREQUISITES AND TERMS

This document assumes you are familiar with Oracle Clusterware and Oracle RAC concepts and administration. The *Oracle Clusterware and Oracle Real Application Clusters Administration and Deployment Guide* [4] book available with the Oracle Database 10*g* Release 2 documentation is a good source for this information. You should also be familiar with the following MAA white paper and high availability documentation that provide more details about supporting technologies used in this paper:

- Client Failover Best Practices for Highly Available Oracle Databases: Oracle Database 10*g* Release 2 [5]
- *Oracle Database High Availability Overview 10g Release 2* [2]

The following table defines the terminology used in this paper:

Term	Definition
`$ORACLE_HOME`	The location of the Oracle Database software.
<CRS HOME>	The location of the Oracle Clusterware software.
VIP	Virtual IP Address. A virtual IP address is an alternate public address that client connections use instead of the standard public IP address. If a node fails, then the node's VIP fails over to another node on which the VIP can accept connections. The VIP is one part of the Oracle solution to the TCP timeout problem associated with node and network failures.
Services	Entities that you can define to group database workloads and route work to the optimal instances that are assigned to offer the service. The Service definition eliminates the need for clients to know on which Oracle instance work needs to run. Services are administered using Cluster Ready Services (CRS) on the server side.

ASSESSING AVAILABILITY

The following sections explain how this project met the following assessment objectives:

- Ensure a Consistent Workload—Have a consistent, repeatable workload.
- Test a variety of failure scenarios—Test a variety of common failures.

- Understand the purpose of the failover components—Understand the components that provide availability, from both a server and client application perspective.
- Understand component relationships—It is important not to make any assumptions about the availability of components without first understanding their relationships.

For a more detailed overview of the project and its environment, see Appendix A.

Ensure a Consistent Workload

Before every failure test on this project was performed, the environment configuration, and database data was restored to the same original state. This ensured the test results were not skewed due to factors such as differing data or caching.

Test Multiple Failure Scenarios

This project tested the most common instance, node, and network failures, as follows:

- Oracle Database instance failure due to a failed background process
- Oracle Database instance failure due to a failed controlfile heartbeat
- ASM instance failure due to a failed background process
- Node failure due to operating system panic
- Node failure due to voting disk unavailability
- Node failure due to a failed CSS daemon process
- Network failure due to a failed cluster interconnect interface
- Network failure due to a failed public network interface

Failures were always initiated after generating a peak client workload, which was attaining the defined application service level objectives documented in Appendix A. The hardware resources were not saturated such that a hardware bottleneck occurred after the failure. This ensured that any effects observed were software related, not hardware related.

Understand Oracle Clusterware and Oracle RAC Failover Components

Before attempting to reduce failover time, it is important to understand the components that are involved in the failover process. The following table defines high-level components involved in Oracle Clusterware and Oracle RAC failover and explains the purpose of each component. The location of availability related log data is also specified.

Component	Description
CSS failure detection and reconfiguration	Cluster Synchronization Services (CSS) is the part of Oracle Clusterware that manages node and group membership. When a failure affecting CSS occurs (for example, a node is down), CSS ensures all surviving nodes in the cluster are aware of the failed node's membership change. The process of redefining the cluster node membership is called a *reconfiguration*. Reconfiguration itself is usually very fast, but the process for confirming an actual failure—called *CSS failure detection*—is not as fast. CSS failure detection can take more time because CSS must verify that the node should be removed as opposed to the scenario where a node is having a temporary problem that will resolve itself. Availability information for this project was gathered from the Cluster Synchronization Services log. (`<CRS HOME>/log/<hostname>/cssd/ocssd.log`).
GRD Reconfiguration	The GRD (Global Resource Directory) reconfiguration is the process of remastering enqueue and cache resources from the failed instance to surviving instances. GRD reconfiguration occurs in any instance failure, including ASM instance failure, though the semantics of ASM GRD resources are different than that of a database instance.[1] Availability information for this project was gathered from the alert log.
Database Instance Recovery	Database Instance Recovery is the process of a surviving instance recovering the redo thread from a failed instance. This is different than crash recovery, which is the process of recovering from the failure of all instances accessing a database. Availability information for this project was gathered from the Oracle Database alert log and SMON trace file
CRSD resource failure detection and failover	Cluster Ready Services Daemon (CRSD), along with the RACG infrastructure, is the part of Oracle Clusterware that detects resource failure and fails over a resource to a surviving node. Examples of critical resources are the VIP (Virtual IP) and database services. Availability information for this project was gathered from the CRSD log and database alert log.

[1] The GRD reconfiguration proved to be very fast during this project. The average recovery time for both Oracle Database instances and ASM instances was never more than 1 second.

Understand Application Components

Although Service Level Brownout is not a failover component, it represents pure application impact and is considered the most important availability metric. Service level brownout is referenced in this paper along with the other failover components to illustrate the relationship between server side component failover and application impact.

Component	Description
Service level brownout	Service level brownout represents the amount of time after an unplanned outage when the application was not achieving both its response time and throughput service levels documented in Appendix A Availability information for this project was gathered from client side logs.

Understand Component Relationships

Oracle can recover many components in parallel, so it is important not to make any assumptions about the availability of components without first understanding the relationships between components.

The following high-level relationships exist:

- During instance recovery, clients must wait for the following before limited GRD access[2] is allowed:
  - GRD reconfiguration
  - First pass redo scan and recovery claiming.

  The amount of time it takes for the redo scan phase and recovery claim phases of instance recovery is very important because it affects the availability of the entire cluster.

  An estimate of this time (in seconds) can be viewed with the following query:

  ```
  SELECT ESTD_CLUSTER_AVAILABLE_TIME FROM V$INSTANCE_RECOVERY;
  ```

  After the redo application phase of instance recovery is complete, full access to GRD resources is permitted.

[2] Limited GRD access means clients have access to blocks not in the recovery set. Their transactions will execute, those using data blocks that need recovery will be slower.

- During node and interconnect failures, CRSD cannot fail over resources such as the VIP and Services until it gets confirmation from CSS that the node is no longer a part of the cluster.
- During node and interconnect failures, a surviving instance cannot start instance recovery until it gets the new node membership bitmap from CSS. The LMON process accepts the bitmap and manages membership for the database.
- Service level brownout almost always represents the largest impact because it is highly dependent on the recovery of the other.[3]

Tier 1 - Critical Components Impacting Availability

After understanding the relationship between these components, it is important to understand the most critical components with respect to availability. The tier 1 critical components that have the biggest impact on availability are as follows:

- At the node level: CSS failure detection and reconfiguration.

  The database is a client to CSS, and during a failure, it will wait for a proper node membership bitmap before proceeding with the other aspects of recovery.

- At the instance level: instance recovery.

  Instance recovery affects the availability of GRD resources, which in turn impacts client applications. Sometimes, the efficiency of instance recovery can indirectly affect the availability of services.

Tuning these two components will make the biggest difference to an application's availability during an unplanned outage. The best practices defined in this paper will significantly reduce the availability impact of these two components.

Tier 2 - Important Components Impacting Availability

A second tier of components exists, which are important though not as critical as the Tier 1 critical components. The tier 2 components are as follows:

- Availability of the VIP:

  Increasing the availability of the VIP is very important in order to avoid TCP/IP timeouts. The sole purpose of the VIP failing over is so clients can connect to it and get the '`No listener`' error message. This message directs the Oracle Net layer to use connect time failover and try the next address in its address list. A client application to a RAC cluster should always specify all cluster VIPs in its connect string to enable connect-time failover.

---

[3] There are exceptions to this depending on the way the application client and database services are configured. An example is when an application is connecting to a service that is already available on another instance and therefore it doesn't need to wait for CRSD to make that service available.

- Availability of Services:

  Increasing the availability of services is especially important when services are not already readily available on surviving instances. For example, a client could receive its FAN event and try to failover to a surviving instance, but if the service is not registered there, a connection error will occur. In these cases, the client will need to continue to retry the connection until the service is made available.

BEST PRACTICES

It is highly recommended that all best practices be implemented rather than trying to choose individual best practices that you may think are more important in your environment. The reason for this is that some of the best practices are dependent on each other to work optimally.

Software Best Practices

The Oracle client software, Oracle Database software, and Oracle Clusterware software should all be running Oracle 10*g* Release 2 Patchset 2 (10.2.0.3) Enterprise Edition. This release contains critical functionality used by the other best practices.

Client Configuration Best Practices

Client configuration best practices can be found in the Oracle Database 10*g* Release 2 Best Practices: Client Failover for Highly Available Oracle Databases [5] white paper.

Oracle Clusterware Configuration Best Practices

Optimize Failure Detection Time

Recall that failure detection time has the highest impact on availability for Oracle Clusterware components. The sole Oracle Clusterware best practice is to optimize failure detection time.

To simplify and automate the configuration changes required for optimizing failure detection time, Oracle will provide a special failure detection optimization patch to customers requiring maximum availability. This patch will first verify that the system is capable of supporting the configuration changes required for maximum availability and then implement those changes if the system can support it. This patch will be delivered via inclusion in an upcoming Oracle Clusterware Bundle patch and it will also be included in Oracle 10g Release 2 Patchset 3 (10.2.0.4).

Database Configuration Best Practices

The following list summarizes the configuration best practices for Oracle Databases:

1. Understand the Instance Recovery Target and Optimize if Required
2. Maximize the number of processes performing transaction recovery.
3. Ensure asynchronous I/O is enabled.

The first best practice alone can comprise 80% of the total availability increase you achieve. The second and third best practices can comprise 20% of the total availability increase achieved at the database level, in most cases.

Understand the Instance Recovery Target and Optimize if Required

This is most important database configuration best practice, because in most cases, it will comprise 80% of the total availability increase.

Recall that instance recovery is critical component impacting availability. Instance recovery is the process of recovering the redo thread from the failed instance. The availability of the database during instance recovery has greatly increased over the last few major releases of the Oracle Database.

When using RAC, the SMON process in one of the surviving instances does instance recovery of the failed instance. This is different than crash recovery, which occurs when all instances accessing a database have failed. Crash recovery is the only kind of recovery when an instance fails using a single instance Oracle Database.

In both RAC and single-instance environments, checkpointing is the internal mechanism used to bind Mean Time To Recover. Checkpointing is the process of writing dirty buffers from the buffer cache to disk. With more aggressive checkpointing, less redo is required for recovery after a failure. Although the objective is the same, the parameters and metrics used to tune MTTR are different in a single-instance environment and a RAC environment for Oracle Database 10*g* Release 2.

In a single instance environment, the `FAST_START_MTTR_TARGET` parameter can be set to the number of seconds the crash recovery should take. Note that crash recovery time includes the time to startup, mount, recover, and open the database.

In a RAC 10g Release 2 Patchset 2 environment (10.2.0.3), the best practice is to use the `_FAST_START_INSTANCE_RECOVERY_TARGET` parameter instead of the `FAST_START_MTTR_TARGET` parameter. The `_FAST_START_INSTANCE_RECOVERY_TARGET` parameter represents the number of seconds that it will take a surviving instance to recover the redo thread of a failed instance. This includes the first pass redo scan, recovery claiming, and the second pass redo application.

So, an important distinction is the fact that `FAST_START_MTTR_TARGET` bounds the total time for startup, mount, crash recovery, and open whereas `_FAST_START_INSTANCE_RECOVERY_TARGET` bounds only instance recovery. Another important point is that `_FAST_START_INSTANCE_RECOVERY_TARGET` is based on the assumption that only one instance has failed. It is not correct to expect `_FAST_START_INSTANCE_RECOVERY_TARGET` to properly bound instance recovery when multiple instances have failed.

It is imperative that the capacity of the I/O subsystem being used, from a response time and throughput perspective, is understood when attempting to tune any Oracle parameters corresponding to checkpointing. You do not want to set checkpointing too aggressively such that your I/O subsystem is flooded, because this will result in worse performance.

The following command shows an example of how to change _FAST_START_INSTANCE_RECOVERY_TARGET:

```
ALTER SYSTEM SET "_FAST_START_INSTANCE_RECOVERY_TARGET"=5 SCOPE=BOTH;
```

Oracle provides a number of ways to help you understand the MTTR target your system is currently achieving and what your potential MTTR target could be given the I/O capacity.. See Appendix C for more information.

Maximize the Number of Processes Performing Transaction Recovery

This database configuration best practice along with the next one will comprise about 20% of the total availability increase achieved at the database level in most cases.

The FAST_START_PARALLEL_ROLLBACK parameter determines how many processes are used for transaction recovery, which is done after redo application. Optimizing transaction recovery is important to ensure an efficient workload after an unplanned failure. As long as the system is not CPU bound, setting this to a value of HIGH is a best practice. This causes Oracle to use four times the CPU count (4 X cpu_count) parallel processes for transaction recovery. The default for this parameter is LOW, or two times the CPU count (2 X cpu_count). Set this parameter as follows:

```
ALTER SYSTEM SET FAST_START_PARALLEL_ROLLBACK=HIGH SCOPE=BOTH;
```

Ensure Asynchronous I/O Is Enabled

This database configuration best practice along with the previous one will comprise about 20% of the total availability increase achieved at the database level in most cases

Under normal circumstances, Oracle will automatically detect if asynchronous I/O is available and appropriate for a particular platform and enable it through the DISK_ASYNCH_IO parameter. However, it is always a best practice to ensure that asynchronous I/O is actually being used. A simple method of checking whether asynchronous I/O is being used is to attach a system call tracer to the DBWR process in your test environment for a few seconds. The output should reveal the calls that are being used. For example, on Linux, the `pread` call is executed for synchronous I/O and the `io_submit` call is used for asynchronous I/O.

To explicitly enable asynchronous I/O, set the DISK_ASYNCH_IO parameter to TRUE. For example:

```
ALTER SYSTEM SET DISK_ASYNCH_IO=TRUE SCOPE=SPFILE SID='*';
```

Troubleshooting Best Practices

The following table explains where to find more detailed diagnostic information for components involved in failover.

Component	Best Practices for Troubleshooting

CSS failure detection and reconfiguration	CSS information is logged in the `<CRS HOME>/log/<nodename>/cssd/ocssd.log` file.
	
	You can use CSS information to determine why evictions occurred. For example, you might want to look for missed network heartbeat checkins, which would indicate either latency or a problem with the network heartbeat. The following example shows two consecutive missed checkins:
	
	`[ CSSD]2006-09-29 11:25:20.292 [217074608] >TRACE: clssnmPollingThread: node stbdc17 (2) missed(2) checkin(s)`
	`[ CSSD]2006-09-29 11:25:21.302 [217074608] >TRACE: clssnmPollingThread: node stbdc17 (2) missed(3) checkin(s)`
	
	Note: You may need to check CSS logs on surviving nodes to fine out if they evicted the failed node.
	
	Additional levels of tracing beyond the default level are available for CSS. Contact Oracle Support Services if you need more detailed tracing.

GRD reconfiguration	GRD reconfiguration information is logged in the alert log of both the database instance and the ASM instance. You will find messages such as 'Reconfiguration started' and 'Reconfiguration complete.'
Database Instance Recovery	Database Instance Recovery information is logged in the alert log and SMON trace file: • The alert log contains messages indicating the startup and completion of each phase such as redo scanning and redo application. • The SMON trace file has more detailed information on recovery, including recovery claiming timings
CRSD resource failure detection and failover	CRSD information is logged in the `<CRS HOME>/log/<nodename>/crsd/crsd.log` file and more detailed resource information can be found in the `$ORACLE_HOME/log/<nodename>/racg/<resource name>.log` file for database resources and `<CRS HOME>/log/<nodename>/racg/<resource name>.log` for CRS resources. • An example of a resource found in `$ORACLE_HOME/log/<nodename>/racg/<reso`

	urce name>.log directory would be a database service resource. • An example of a resource found in <*CRS HOME*>/log/<*nodename*>/racg/<resource name>.log file would be the VIP resource.

SAMPLE CONFIGURATIONS

This paper has emphasized how important it is to understand your particular environment before making Oracle Clusterware and Oracle Database configuration changes. You can use the following table as a reference for determining optimal best practices configuration settings for your environment.

Environment Attributes	Sample Configuration
Cluster with < 4 nodes Running close to capacity on interconnect; Some observable latency experienced during peak loads[4] Sufficient I/O capacity available Sufficient CPU capacity available even after application fails over to surviving instances	• Apply the bundle or Patchset containing the Oracle Clusterware failure detection optimization patch • `_FAST_START_INSTANCE_RECOVERY_TARGET` database parameter set to 5 • `FAST_START_PARALLEL_ROLLBACK` set to `HIGH`
Sufficient network bandwidth available; Low latency Interconnect Running close to capacity on I/O subsystem	• Apply the bundle or Patchset containing the Oracle Clusterware failure detection optimization patch • `_FAST_START_INSTANCE_RECOVERY_TARGET` database parameter set to 60 • `FAST_START_PARALLEL_ROLLBACK` set to `HIGH`

[4] For example, on Linux, this can be measured with tools like ifconfig, iptraf, and netperf

Sufficient CPU capacity available even after application fails over to surviving instances/nodes	
Sufficient network bandwidth available; Low latency Interconnect Sufficient I/O capacity available Running close to CPU capacity especially after application fails over to surviving instances/nodes	• Apply the bundle or Patchset containing the Oracle Clusterware failure detection optimization patch • `_FAST_START_INSTANCE_RECOVERY_TARGET` database parameter set to 10

APPENDIX A – THE PROJECT ENVIRONMENT

This section describes the hardware and software infrastructure used for this project.

Hardware and Software Client Infrastructure

The client hardware used was a single node, configured with 2 hyper-threaded 3.20GHz CPUs and 6GB memory.

The operating system used was RHEL3 Update 6 with the 2.4.21-37.Elsmp kernel. The following additional software was used:

- The Swingbench[5] load generator was used for all tests except the batch tests. It was instrumented with special client-side logging to obtain the required service level metrics. Swingbench was configured to use the following additional software:
  - Java v1.5.0_06
  - Oracle JDBC client software with aforementioned client software and configuration best practices
- SQL*Plus was used for the batch tests

The same client configuration was used when running tests against both the generic and best practice environments.

Hardware and Software Server Infrastructure

Most tests were run on two 2-node clusters although some 3-node validation tests were run at the end of the project. Each node had 2 hyperthreaded 3.20GHz CPUs and 6GB memory. The I/O subsystem used was Netapp, which presented a Write Anywhere File Layout (WAFL) filesystem to the hosts. Ten 50GB ASM disks were created on top of the WAFL filesystem. Two primary ASM disk groups were used—`DATA` for the database files and `FLASH` for the flash recovery area.

The operating system used was RHEL3 Update 6 with the 2.4.21-37.Elsmp kernel. The Oracle Clusterware and Oracle Database (including ASM) software version used was 10.2.0.2 for the generic testing and 10.2.0.3 for the best practice testing. Because one of the software best practices is to use release 10.2.0.3, it was important to compare results between release 10.2.0.3 and a different version. Release 10.2.0.2 was selected for this purpose.

The Oracle Cluster Registry (OCR) was mirrored two ways on the WAFL file system. The Oracle Clusterware voting disk was mirrored three ways on the WAFL file system.

[5] http://www.dominicgiles.com/swingbench.html

The services were configured as follows:

- Two services primary on node 1; available on node 2. These were not used by the application but used to assess service failover times during OLTP failure tests.
- Two services primary on node 2; available on node 1. These were not used by the application but used to assess service failover time during OLTP failure tests.
- One service primary on node 1 and node 2. This service was used by the Order Entry and Calling Circle applications.
- One service primary on node 1; available on node 2. This service was used by the DSS application.

Note: Unless otherwise noted in this paper, the hardware resources were not saturated such that a hardware bottleneck occurred. Careful baseline testing was done to ensure that the workload on the surviving nodes would not exceed 75%. This ensures the test results represent software impacts and not hardware impacts.

The Sample Applications

One of the goals of this project was to test multiple failures under a variety of application workloads. To that end, four distinctly different workloads were tested: Order Entry, Calling Circle, DSS, and batch schema maintenance.

The sample applications used to illustrate availability impacts have the following workload characteristics

- **Order Entry**—The Swingbench load generation tool is used to generate the Order Entry workload. It consists of five transactions types that are typical in an order entry application: Creating a Customer, Browsing Products, Placing Orders, Processing Orders and Reviewing Orders. The I/O size for this workload is small (generally the tablespace blocksize, which is 4K in this case). It connects to the database using thin JDBC and all client configuration best practices.

  The following Load Profile example from an AWR report reflects the Order Entry workload on a particular instance. Note that server side load balancing ensured that all instances were generating this same load profile during the tests. This means that the client application was achieving approximately double the metrics shows below since this is a two node cluster.

```
Load Profile
~~~~~~~~~~~~                            Per Second       Per Transaction
                                   ---------------       ---------------
                  Redo size:          227,022.83              2,553.49
              Logical reads:           27,493.38                309.24
              Block changes:            1,466.54                 16.50
             Physical reads:              137.72                  1.55
            Physical writes:              423.99                  4.77
                 User calls:              122.38                  1.38
                     Parses:               73.36                  0.83
                Hard parses:                0.10                  0.00
                      Sorts:               89.39                  1.01
                     Logons:                0.02                  0.00
                   Executes:              594.18                  6.68
               Transactions:               88.91
```

- **Calling Circle**—The Swingbench load generation tool is used to generate the Calling Circle workload. The Calling Circle workload represents a self-service OLTP application. The application models the customers of a telecommunications company registering, updating, and enquiring on a calling circle of their most frequently called numbers in order to receive discounted call pricing. The I/O size for this workload is small (generally the tablespace blocksize which is 8K in this case). It connects to the database using thin JDBC and all client configuration best practices.

  The following Load Profile example from an AWR report reflects the Order Entry workload on a particular instance. Note that server side load balancing ensured that all instances were generating this same load profile during the tests. This means that the client application was achieving approximately double the metrics shows below since this is a two node cluster.

```
Load Profile
~~~~~~~~~~~~                            Per Second       Per Transaction
                                   ---------------       ---------------
                  Redo size:          173,399.33              1,114.95
              Logical reads:           26,791.34                172.27
              Block changes:            1,076.09                  6.92
             Physical reads:              100.98                  0.65
            Physical writes:              178.52                  1.15
                 User calls:              487.96                  3.14
                     Parses:              483.53                  3.11
                Hard parses:                5.13                  0.03
                      Sorts:               16.09                  0.10
                     Logons:                0.05                  0.00
                   Executes:            3,446.32                 22.16
               Transactions:              155.52
```

- **DSS**—The Swingbench load generation tool is used to generate the DSS workload. The DSS workload is based on the 'Sales History' sample schema that ships with Oracle Database 10*g*. It consists of longer running, I/O intensive transactions that are typical in a Decision Support System. It connects to the database using thin JDBC and all client configuration best practices.

The following Load Profile example from an AWR report reflects the DSS workload on a particular instance. Note that an active/passive service configuration was used for the DSS application. Therefore, this workload was only active on one instance at a time.

```
Load Profile
~~~~~~~~~~~~                            Per Second       Per Transaction
                                   ---------------       ---------------
                  Redo size:          11,169.86             31,106.72
              Logical reads:           3,889.66             10,832.24
              Block changes:              28.64                 79.76
             Physical reads:           2,803.87              7,808.44
            Physical writes:               7.35                 20.48
                 User calls:               0.46                  1.28
                     Parses:              10.10                 28.12
                Hard parses:               0.80                  2.24
                      Sorts:              10.00                 27.84
                     Logons:               0.01                  0.04
                   Executes:              29.53                 82.24
               Transactions:               0.36
```

- **Batch schema maintenance**—The batch maintenance workload is the execution of parallel online index rebuilds against the larger indexes in the Calling Circle and Order Entry OLTP systems. The I/O size for this workload is large (between 128KB and 1 MB.) It connects to the database using a local bequeath connection via SQL*Plus and controls which instances the parallel processes are running on via the `INSTANCE_GROUPS` and `PARALLLEL_INSTANCE_GROUPS` parameters.

  The following Load Profile example from an AWR report reflects the Batch schema maintenance workload on a particular instance during a rebuild of a large Calling Circle index. Note that the batch schema maintenance is never active on more than one instance.

```
Load Profile
~~~~~~~~~~~~                            Per Second       Per Transaction
                                   ---------------       ---------------
                  Redo size:       7,005,271.89          2,370,408.88
              Logical reads:           1,841.34                623.06
              Block changes:             285.65                 96.66
             Physical reads:           1,900.54                643.09
            Physical writes:             887.79                300.41
                 User calls:               1.66                  0.56
                     Parses:              54.03                 18.28
                Hard parses:               5.45                  1.84
                      Sorts:              38.97                 13.19
                     Logons:               0.09                  0.03
                   Executes:             116.18                 39.31
               Transactions:               2.96
```

The Service Level Objectives

The following application service-level objectives will be assessed during unplanned failures:

- **Order Entry:** Measured 100 concurrent clients during the failure tests. Clients must achieve a response time of no more than 3 seconds for a

business transaction. A minimum of 100 business transactions per second (total) must be maintained for the 100 concurrent clients being measured.

- **Calling Circle:** Measured 100 concurrent clients during the failure tests. Clients must achieve a response time of no more than 9 seconds for a business transaction. A minimum of 20 business transactions per second (total) must be maintained for the 100 concurrent clients being measured
- **DSS:** Measured 10 concurrent clients during the failure tests. Because the DSS workload is I/O-bound, the service level is based on I/Os per second. The DSS clients must maintain a minimum of 2000 8K I/Os per second for the 10 users being measured.
- **Batch schema maintenance:** One SQL*Plus process, which was the query coordinator for many parallel query processes, executed the parallel create index. The service level for batch schema maintenance is at the instance level. The surviving instance should be available to accept the reexecution of the schema maintenance job by the foreground process immediately after failure

Note: The service level objectives for Order Entry and Calling Circle are based on a business transaction. A business transaction is not necessarily the same as a database transaction. A business transaction represents at least one database transaction and depending on the workload, it can be many database transactions. The business transaction metrics are gathered at the client level to give the most accurate service level data.

APPENDIX B - TEST RESULTS WITH AND WITHOUT BEST PRACTICES

Environment Configurations

Two environments were used to determine the benefits of using best practices:

- The environment without best practices is called the ***generic 10.2.0.2 environment***.
- The environment with best practices is called the ***best-practice 10.2.0.3 environment***.

Because one of the software best practices is to use release 10.2.0.3, it was important to compare results between release 10.2.0.3 and a different release. Release 10.2.0.2 was selected for this purpose.

The *generic 10.2.0.2 environment* (without best practices) consisted of all default parameters that come with a release 10.2.0.2 CRS and Oracle Database installation. The database parameters were configured by DBCA to be the defaults for a *General Purpose* database.

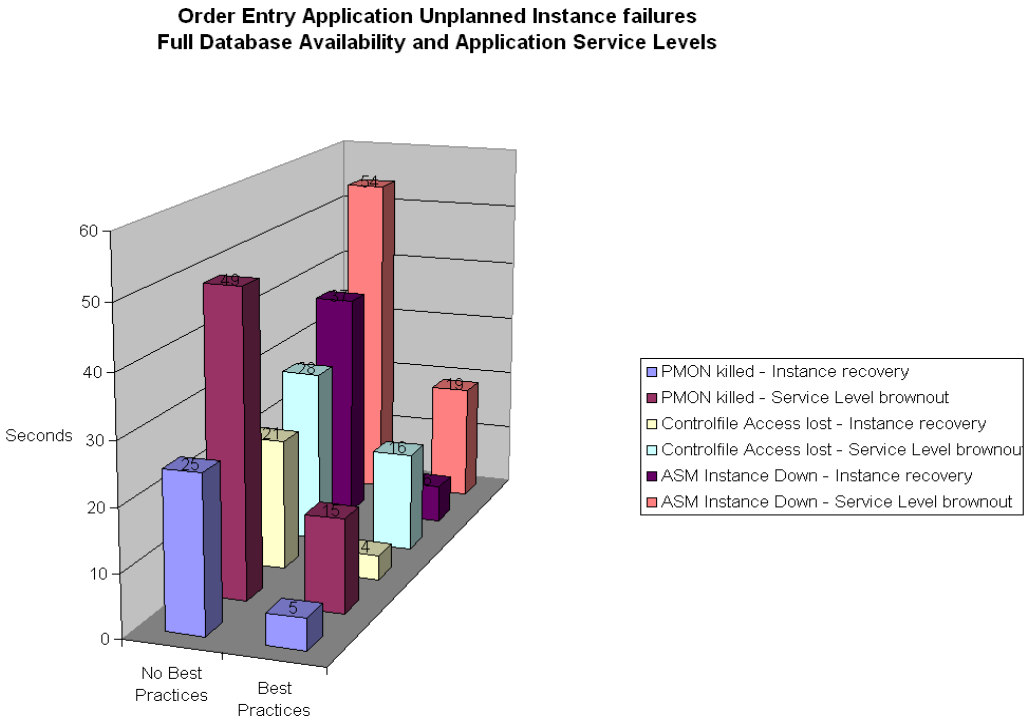
The *best-practice 10.2.0.3 environment* used for this project had the following configuration changes implemented:

- Applied same configuration changes supplied in the Oracle Clusterware failure detection optimization patch
- Set the `_FAST_START_INSTANCE_RECOVERY_TARGET` database parameter to 5
- Set the `FAST_START_PARALLEL_ROLLBACK` parameter to `HIGH`
- Set the `DISK_ASYNCH_IO` parameter to `TRUE`

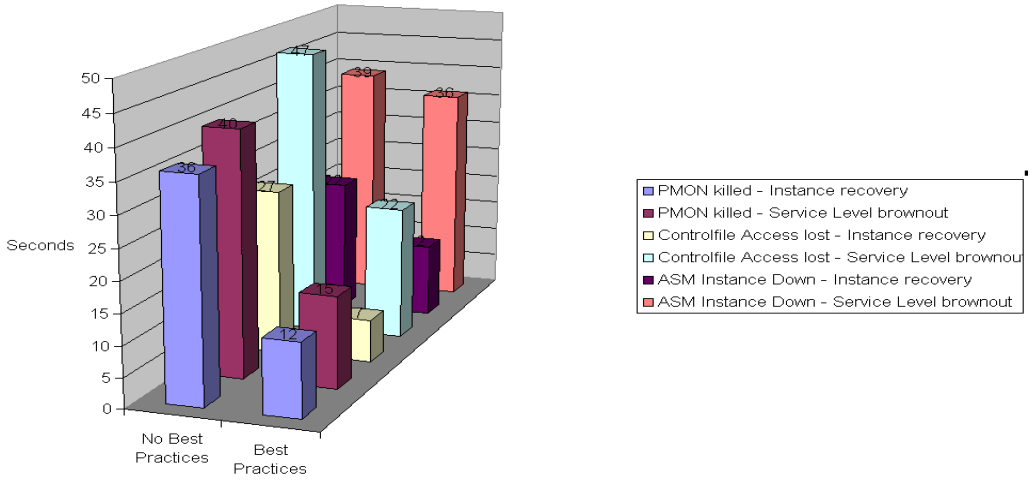
For a table of alternate configurations based on different environment attributes, see the Sample Configurations section.

Results Charts

The following charts show how much quicker the OLTP sample applications returned full database availability and application service levels after an unplanned instance failure when using best practices as opposed to not using best practices. The components charted are those that are most important to failover time for this application and the application service level component.

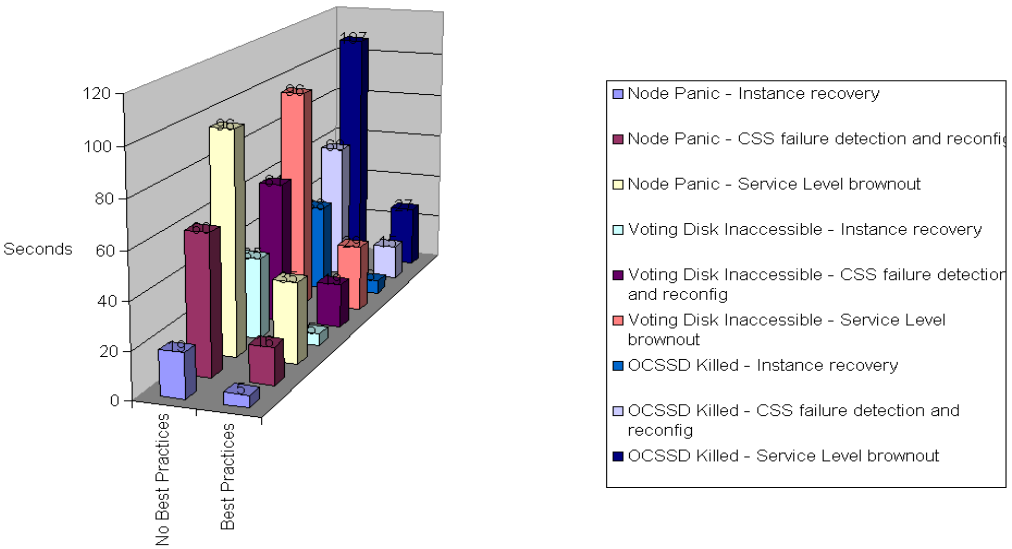


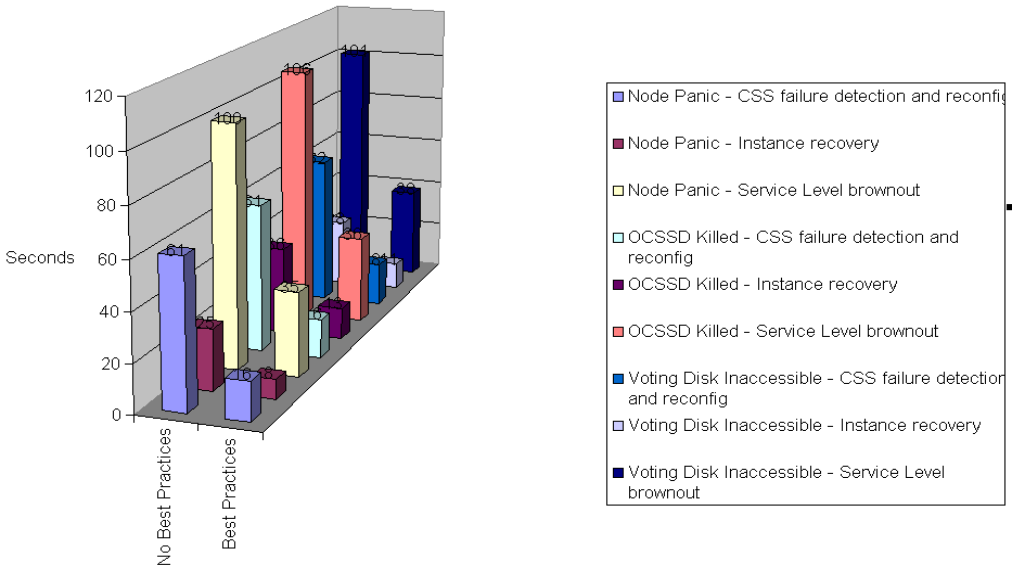
Calling Circle Unplanned Instance failures
Full Database Availability and Application Service Levels



The following charts show how much quicker the OLTP sample applications returned to full database availability and service levels after an unplanned node failure when using best practices as opposed to not using best practices. The components charted are those that are most important to failover time for this application and the application service level component.

Order Entry Unplanned Node Failures
Full Cluster, Database, and Application Service Levels

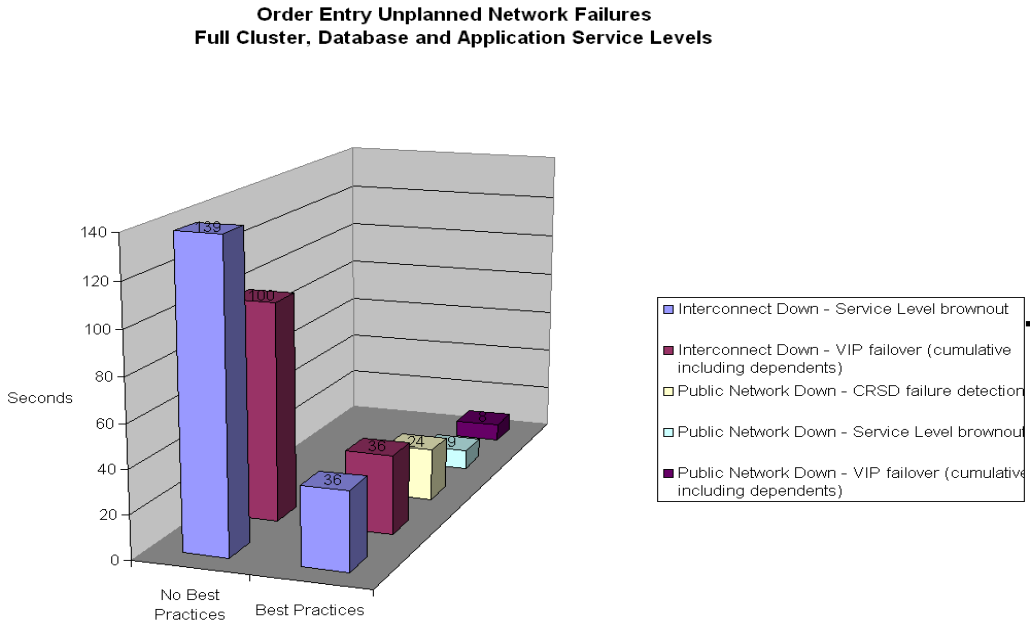


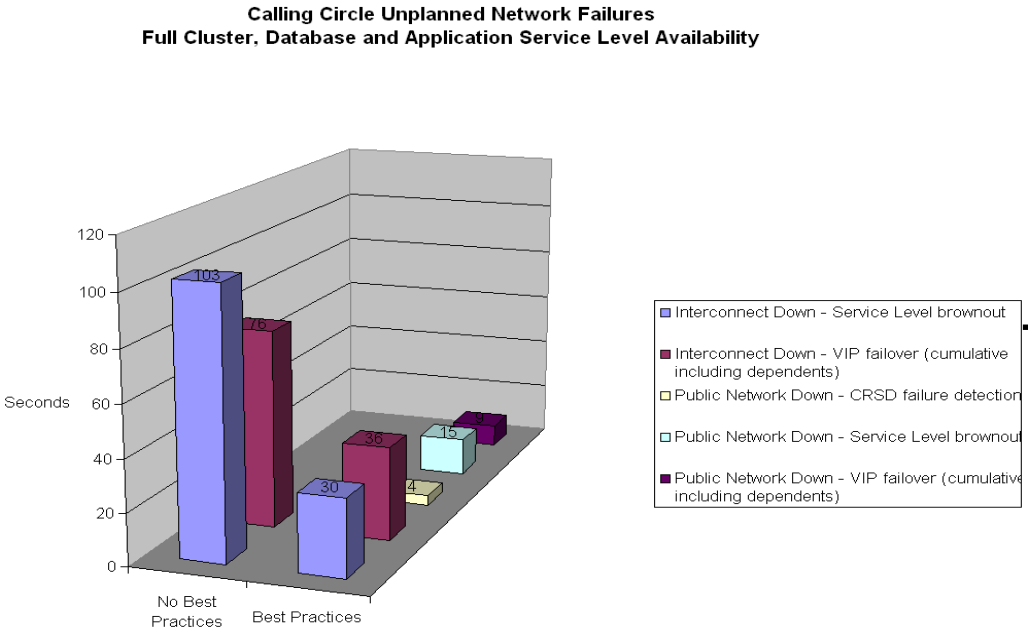


The following charts show how much quicker the OLTP sample applications returned to full database availability and service levels after an unplanned network outage when using best practices. The components charted are those that are most important to failover time for this application and the application service level component.

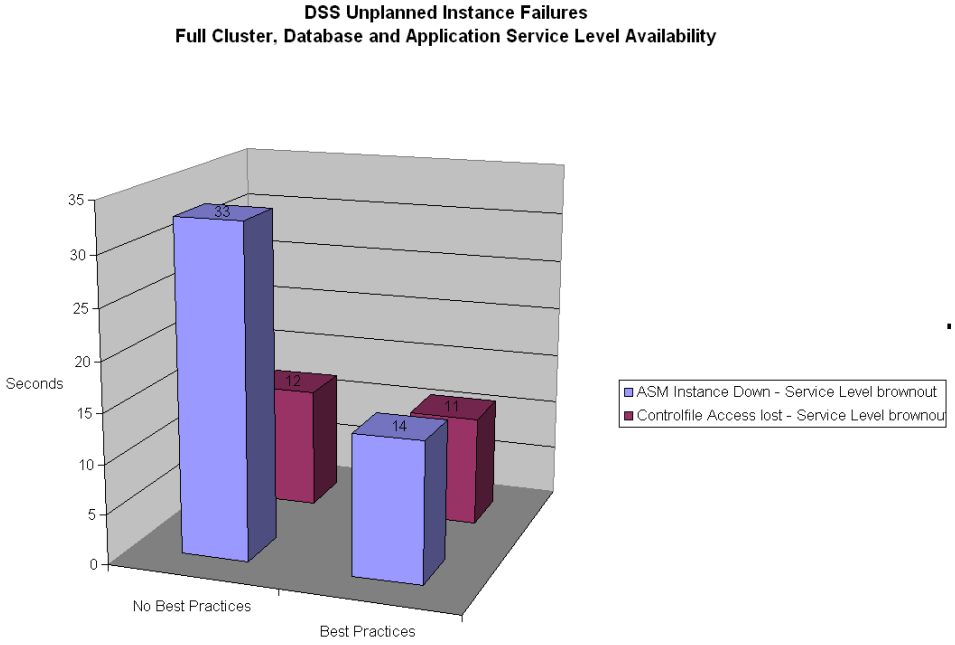
Note: These charts have no data for all public network down tests in the generic environment, because those test were not run.[6]

[6] There were certain fixes related to network outages included in the best practice software that were not in the generic environment. The lack of these fixes made for an unfair comparison between the two environments and therefore public network down tests were not done in the generic environment.

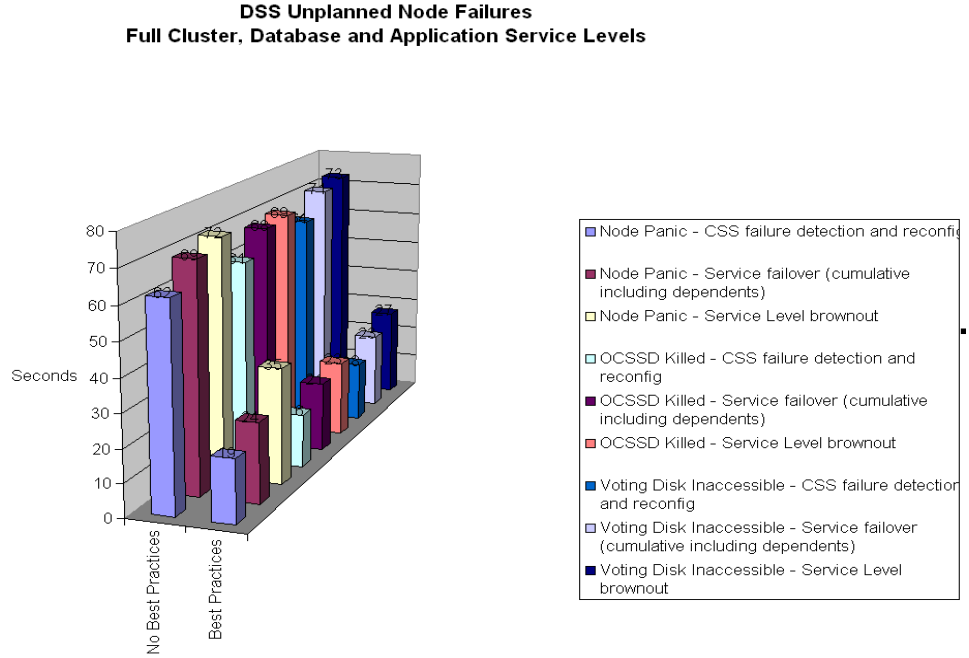




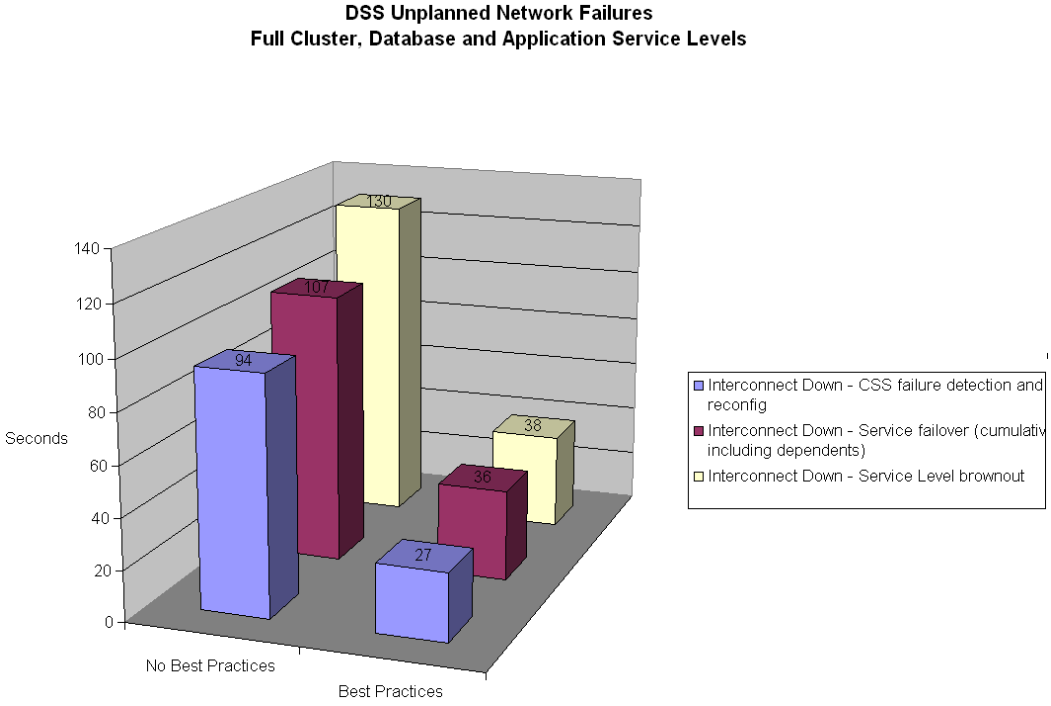
The following chart shows how much quicker the DSS sample applications returned to full database availability and service levels after an unplanned instance failure when using best practices as opposed to not using best practices. The components charted are those that are most important to failover time for this application and the application service level component



The following chart shows how much quicker the DSS sample applications returned to full database availability and service levels after an unplanned node failure when using best practices as opposed to not using best practices.

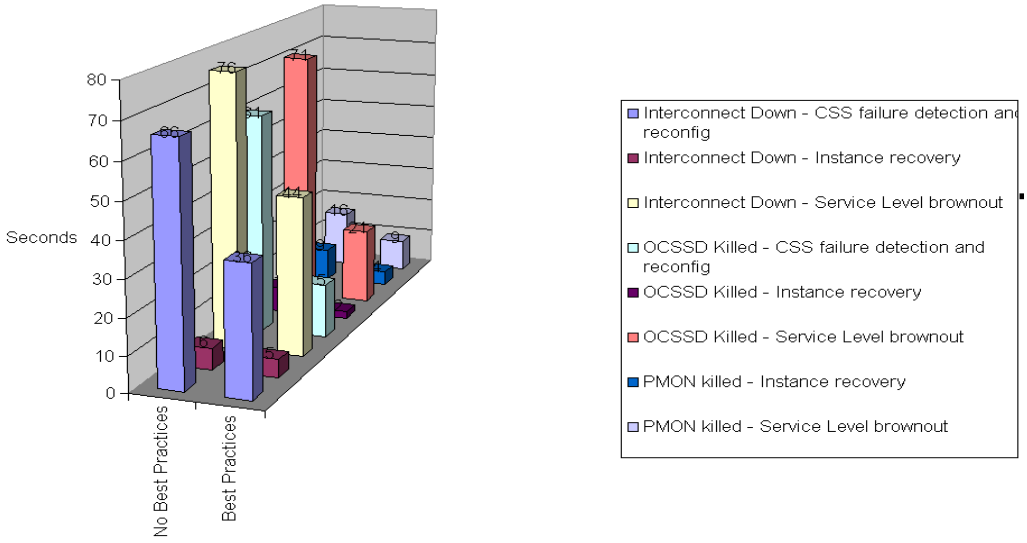


The following chart shows how much quicker the DSS sample applications returned to full database availability and service levels after an unplanned network failure when using best practices as opposed to not using best practices. The components charted are those that are most important to failover time for this application and the application service level component.

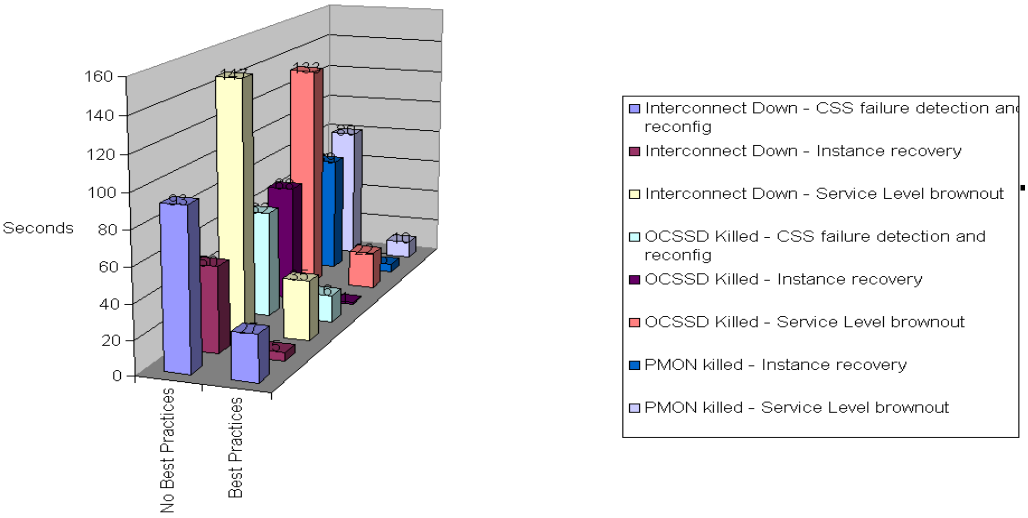


The following charts show how much quicker the batch sample applications returned to full database availability and service levels after an unplanned instance failure when using best practices as opposed to not using best practices. The components charted are those that are most important to failover time for this application and the application service level component.

Batch Calling Circle Unplanned Instance, Node and Network Failures
Full Cluster, Database and Application Service Level Availability



Batch Order Entry Unplanned Instance, Node and Network Failures
Full Cluster, Database and Application Service Level Availability



For a more detailed component breakdown for all sample applications, see Appendix D.

APPENDIX C – UNDERSTANDING THE CURRENT AND POTENTIAL

Current MTTR Target

When using FAST_START_MTTR_TARGET, the actual current MTTR target for a running system can be obtained from the V$INSTANCE_RECOVERY view. Recall this is the estimated time for crash recovery. The specific column is ESTIMATED_MTTR. Note this information is also available via AWR reports. Here is an example AWR report excerpt containing this information:

```
  Targt  Estd
  MTTR   MTTR   Recovery  Actual    Target
   (s)    (s)   Estd IOs Redo Blks Redo Blks
- ----- ----- ---------- --------- ---------
B    24    50      10552    186547   2071710
E    24    50      15704    260301   2071710
          ---------------------------------
```

The example shows that despite the target MTTR being at 24, the actual MTTR for this system is estimated at 50 seconds. This is a case where the I/O subsystem cannot handle the desired MTTR target.

When using RAC and the _FAST_START_INSTANCE_RECOVERY_TARGET parameter (recommended), an alternate method must be used to obtain estimated instance recovery time. With Oracle Database 10G release 10.2.0.3, the estimated instance recovery time can be determined with the following query:

```
SELECT CUR_EST_IR_SEC FROM X$ESTIMATED_MTTR;
```

The CUR_EST_IR_SEC column represents the estimated number of seconds that instance recovery will take (first pass redo scan, recovery claiming, seconds pass redo application). In the future, Oracle will be consolidating into one parameter, which will govern both single instance and RAC crash/instance recovery.

Potential MTTR Target

When using FAST_START_MTTR_TARGET in a single instance Oracle database, you can use the V$MTTR_TARGET_ADVICE view to help determine the best setting for FAST_START_MTTR_TARGET that will achieve the crash recovery time objective. The V$MTTR_TARGET_ADVICE view shows approximately how many more I/Os will be required to achieve a particular MTTR target. You should ensure that your I/O subsystem can handle that number of I/Os before actually adjusting the FAST_START_MTTR_TARGET parameter to a particular value.

When using RAC and the _FAST_START_INSTANCE_RECOVERY_TARGET parameter (recommended), the V$MTTR_TARGET_ADVICE view is not populated because that view is only valid when using the FAST_START_MTTR_TARGET parameter. Do not set both the FAST_START_MTTR_TARGET and

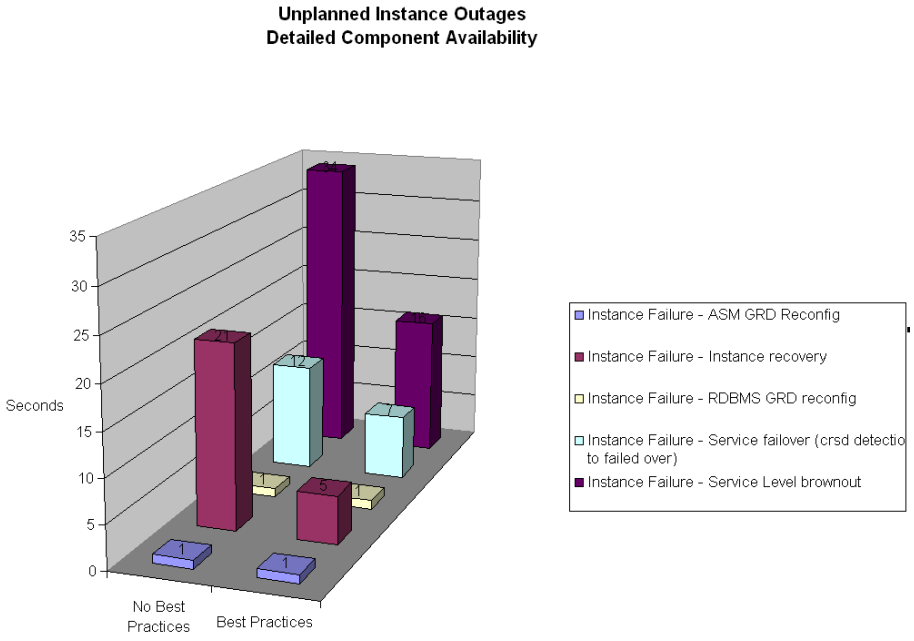
_FAST_START_INSTANCE_RECOVERY_TARGET parameters at the same time. When using RAC, turn off the MTTR advisor by issuing the following statement:

```
ALTER SYSTEM SET "_DB_MTTR_ADVICE"=OFF SCOPE=BOTH :
```

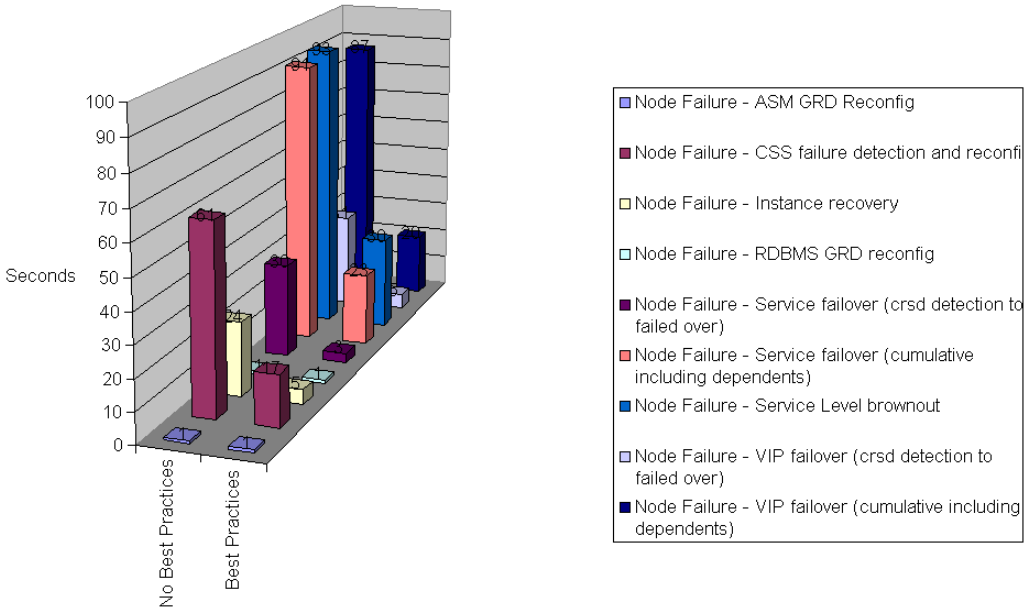
See the Current MTTR target section for help determining an appropriate setting for _FAST_START_INSTANCE_RECOVERY_TARGET by running a peak workload on a test system, trying different settings, and examining X$ESTIMATED_MTTR>EST_IR_REC value.

APPENDIX D – DETAILED COMPONENT AVAILABILITY

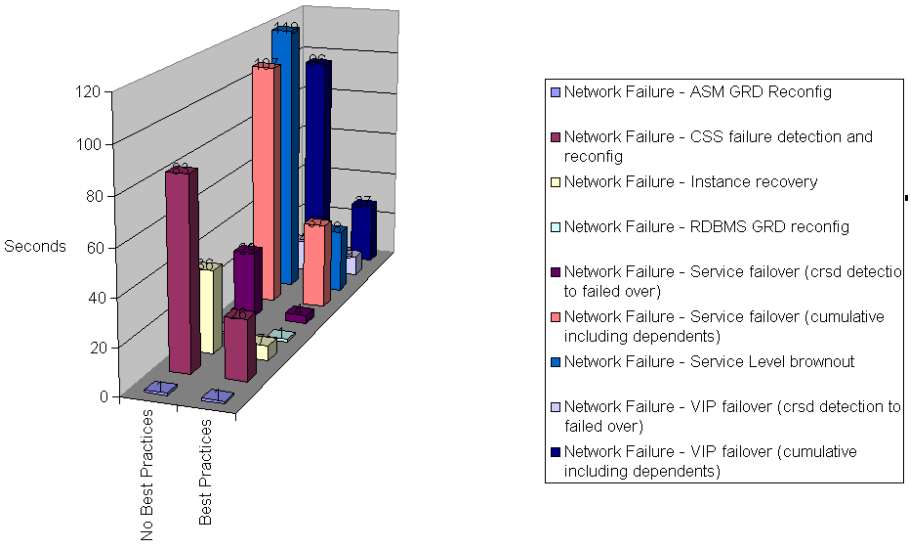
The following charts show a more detailed breakout of component availability when using best practices as opposed to the generic configuration. The charts average the numbers across all client applications and group numbers into instance, network and node failures.



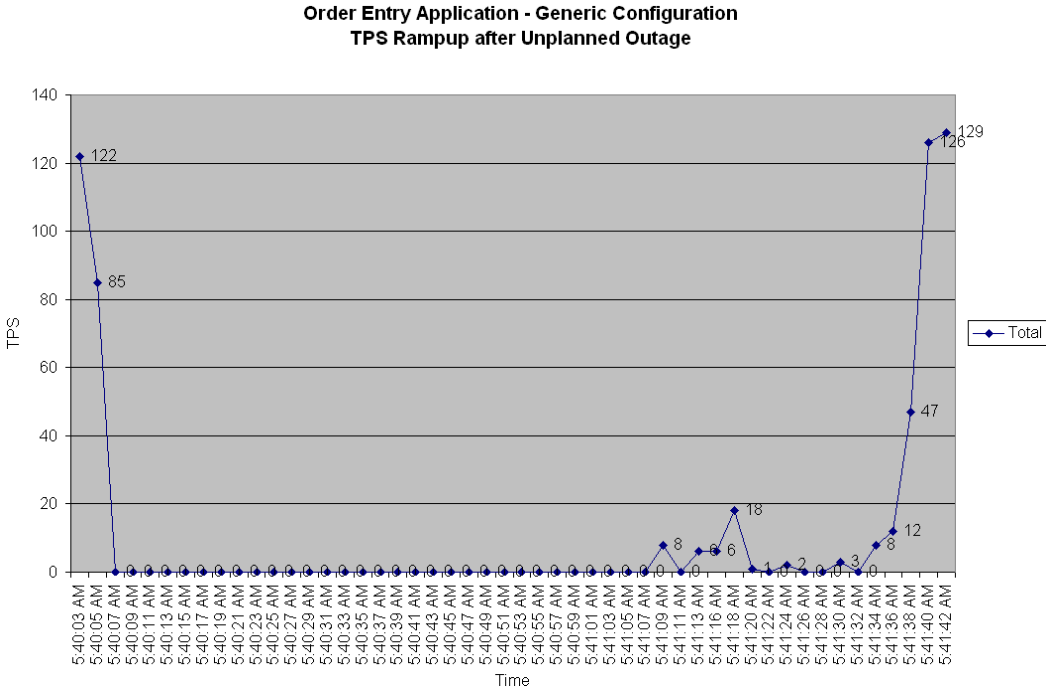
Unplanned Node Outages
Detailed Component Availability

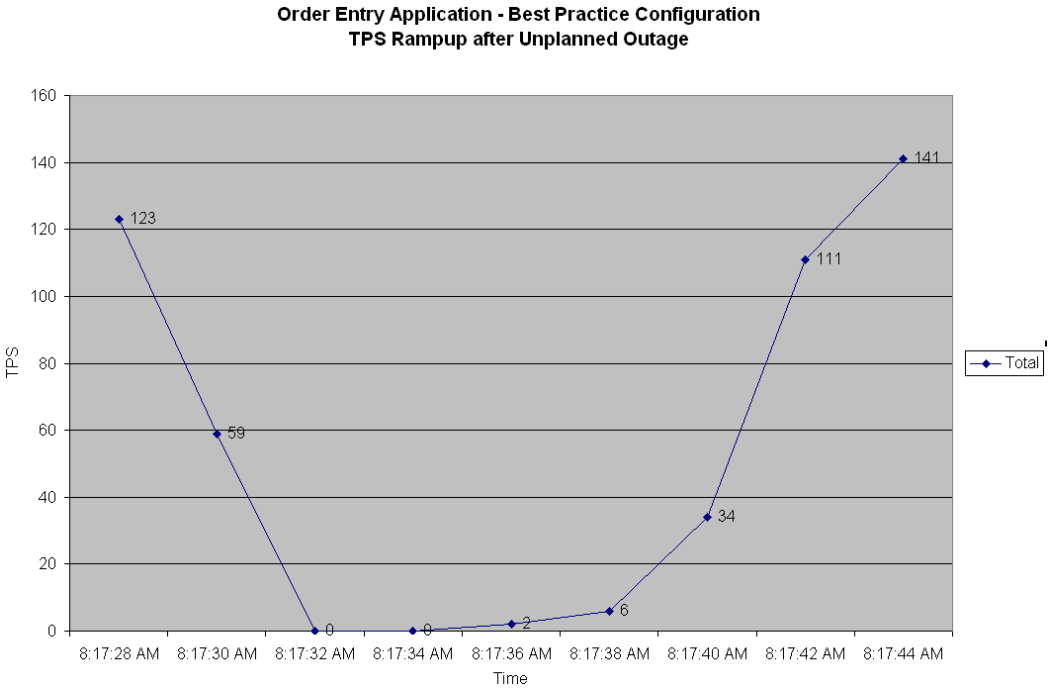


Unplanned Network Outages
Detailed Component Availability



The following charts show the actual ramp-up data points for the Order Entry application after a node panic in a generic configuration and a best-practice configuration. This illustrates the difference between the application's ability to access the database and when it has achieved full service levels.





APPENDIX E – LISTENER CONNECTION RATE THROTTLING

With Oracle Database 10*g* Release 2 Patchset 2 (10.2.0.3), the listener has the capability to throttle the rate at which connections are spawned to the database. Listener connection rate throttling is potentially useful for cases like login storms where a short burst of a high number of connections can cause system resource depletion, which impacts application service level availability.[7]

Listener throttling is potentially more attractive during unplanned outages because the sessions from the failed instance/node are trying to get connected to a surviving instance/node as quickly as possible. If testing proves that an unplanned outage will impact application service levels due to too many sessions failing over too quickly, then test listener throttling to see if it improves application service levels during the login storm.

Here is an example LISTENER.ORA file where just the first address has throttling enabled:

```
LISTENER_STBDC18 =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = stbdc18-vip)(PORT =
1521)(RATE_LIMIT=YES)(QUEUESIZE=1024)(IP = FIRST))
      )
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = 152.68.196.208)(PORT =
1521)(IP = FIRST))
      )
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC))
      )
    )
  )
CONNECTION_RATE_LISTENER_STBDC18=10
```

The `CONNECTION_RATE_<listener_name>` parameter indicates the connection rate per second for all addresses with `RATE_LIMIT` set to '`YES`', '`yes`', or a specific rate. In the example, 10 connections will be spawned per second only for the first address in the address list. The `RATE_LIMIT` parameter can also specify the rate itself instead of simply turning throttling on at the default rate. For example, `RATE_LIMIT=5` on a particular address would be a specific setting for just that address. When specifying a `RATE_LIMIT` at the address level, the `CONNECTION_RATE_<listener_name>` parameter should not be specified. If both the global rate and a specific rate are specified, the global rate is the one enforced.

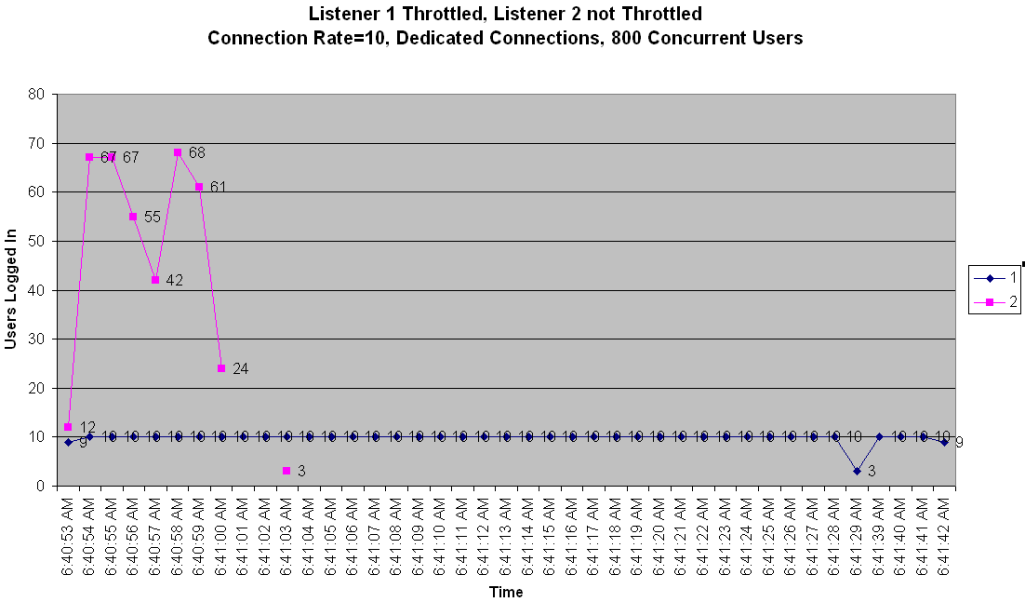
Depending on the way the addresses are used, it may be desirable to enable throttling on the address used by the application, but not on an address used for administration. This ensures that critical administrative tasks are never throttled.

---

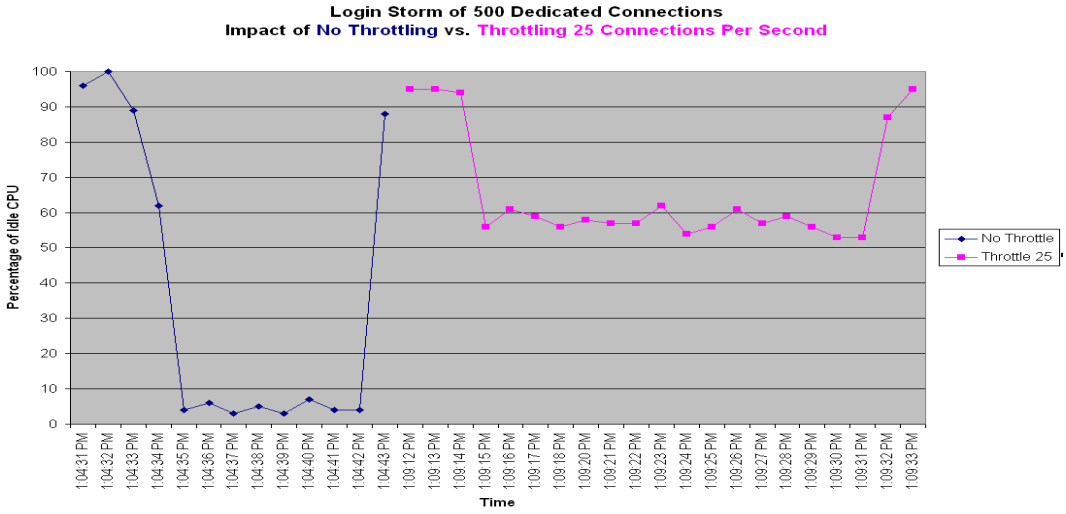
[7] It is important to understand that throttling is implemented on the actual connection requests, but not for any work done as a result of those connection requests.

The LISTENER.ORA example shown above is an example of such a configuration.

The following chart shows listener 1 throttling connections at 10 connections/second, but no throttling is in effect for listener 2. This data was collected during a login storm where both listeners are receiving approximately the same number of connections as well as work resulting from those connections. This chart illustrates the functionality of listener throttling, not the benefit of the feature. You can clearly see that connections are throttled at the defined rate.



The following example shows a case where throttling connections avoids a drastic temporary depletion in system idle CPU. In this case, throttling is a potential benefit because the operating system is not under the stress of a system resource spike.



It is very important to test the effects of listener connection rate throttling with your particular application to ensure the benefit before implementation.

REFERENCES

1. Oracle Maximum Availability Architecture
   http://www.oracle.com/technology/deploy/availability/htdocs/maa.htm
2. *Oracle Database High Availability Overview* (Part #B14210)
   http://otn.oracle.com/pls/db102/db102.to_toc?partno=b14210
3. *Oracle Database High Availability Best Practices* (Part B25159)
   http://otn.oracle.com/pls/db102/db102.to_toc?partno=b25159
4. *Oracle Clusterware and Oracle Real Application Clusters Administration and Deployment Guide* (Part B14197)

   http://otn.oracle.com/pls/db102/db102.to_toc?partno=b14197
5. Client Failover Best Practices for Highly Available Oracle Databases: Oracle Database 10*g* Release 2
   http://www.oracle.com/technology/deploy/availability/pdf/MAA_WP_10gR2_ClientFailoverBestPractices.pdf

Best Practices for Optimizing Availability During Unplanned Outages Using Oracle Clusterware and Oracle Real Application Clusters
March 2007
Author: Michael D. Nowak
Contributors: Carol Colrain and Stefan Pommerenk

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

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