

# Oracle9i Media Recovery Best Practices

*An Oracle White Paper  
August 2003*

# Oracle9i Media Recovery Best Practices

Executive Overview .....	3
Recovery Best Practices and General Observations in Tuning Recovery .....	3
Testing Details .....	8
Description of Test Environment.....	8
Transaction Runs.....	8
Detailed Test Results.....	9
Recovery Monitoring Script and Monitoring Best Practices .....	12
Sample Recovery Monitoring Script - Korn Shell.....	14
Conclusion.....	15

# Oracle 9i Media Recovery Best Practices

## EXECUTIVE OVERVIEW

With the proliferation of Oracle Data Guard for disaster recovery, data protection and scheduled maintenance advantages, optimizing media recovery for physical standby databases becomes more important.

Media recovery occurs when one or more datafiles or the controlfiles are restored from a previous backup. The goal of media recovery is to recover the datafiles and the rest of the database to a consistent point in time or to apply all transactions that has every occurred when a standby database is used to protect the primary site.

In order to leverage a physical standby database for fast switchover or failover in the event of a disaster and service the data needs of the enterprise, you need to tune the time it takes to apply redo to your physical standby database (Data Guard redo apply) to keep it transactionally consistent with the production database with a minimal lag. This paper presents recovery best practices to implement transactionally consistent copies of a production database. These same best practices will assist in tuning the redo apply for any database requiring media recovery.

## Recovery Best Practices and General Observations in Tuning Recovery

These following recommendations will provide significant recovery gains and will provide additional insight in tuning your database for fast media recovery. These best practices were derived after extensive media recovery testing on Oracle9i databases and part of the performance studies within the Maximum Availability Architecture (MAA) projects.<sup>1</sup> For more information about MAA, please refer to <http://otn.oracle.com/deploy/availability/htdocs/maa.htm>. For more information about Oracle 9i Data Guard: Primary Site and Network Configuration Best Practices please refer to [http://otn.oracle.com/deploy/availability/pdf/MAA\\_DG\\_NetBestPrac.pdf](http://otn.oracle.com/deploy/availability/pdf/MAA_DG_NetBestPrac.pdf).

1. Set parallel recovery parallelism to 2 times number of CPUs on one standby host

---

<sup>1</sup> These general best practices should apply to most customer environments. However, these results are not indicative of what you may experience. Testing with serial recovery and different degrees of parallelism is imperative.

During media or standby recovery the redo log is read, and blocks that require redo application are parsed out. With parallel media recovery, these data blocks are subsequently distributed evenly to all recovery processes to be read into the buffer cache. The default is serial recovery or zero parallelism, which implies that the same recovery process reads the redo, reads the data blocks from disk and applies the redo changes. To implement parallel media recovery or standby recovery, you need to add an optional PARALLEL clause to the recovery command. Furthermore, the database parameter PARALLEL\_MAX\_SERVERS needs to be set to at least the degree of parallelism.

By utilizing parallel media or standby recovery, you may experience three times the performance gains over serial recovery for OLTP runs. Below are some examples of how to set recovery parallelism. RECOVER MANAGED STANDBY DATABASE PARALLEL <#CPUS \* 2>;

```
RECOVER STANDBY DATABASE PARALLEL <#CPUS * 2>;
```

```
RECOVER DATABASE PARALLEL <#CPUS * 2>;
```

You should compare several serial and parallel recovery runs to determine your optimal recovery performance. In our testing, minimal gains were found using the SQL\*Loader (batch) runs. Refer to the [“Description of Recovery Test Environment”](#) section for descriptions of OLTP and SQL\*Loader transactions.

Ensure that you have applied the fix for bug 2555509, which provides significant performance gains for parallel media recovery.

## 2. Set DB\_BLOCK\_CHECKING=FALSE for Faster Redo Apply Rates

Setting DB\_BLOCK\_CHECKING=FALSE during standby or media recovery will provide as much as 2 fold increase in apply rate. The lack of block checking during recovery must be an accepted risk. Block checking should always be enabled on the production database. Block checksum (DB\_BLOCK\_CHECKSUM=TRUE) should always be enabled for both production and standby databases.

## 3. Set PARALLEL\_EXECUTION\_MESSAGE\_SIZE =4096

When utilizing parallel media recovery or parallel standby recovery, increasing the PARALLEL\_EXECUTION\_MESSAGE\_SIZE database parameter to 4K (4096) improved parallel recovery by as much as 20%. This should be set on both the primary and the standby in preparation for switchover operations. Increasing this parameter will require more memory from the shared pool by each parallel execution slave process. This parameter is also used by parallel query operations and should be tested with any parallel query operations to ensure there is sufficient memory on the system. A large number of parallel query slaves on a 32-bit installation may reach memory limits and prohibit increasing the

PARALLEL\_EXECUTION\_MESSAGE\_SIZE from the default 2K (2048) to 4K.

#### 4. Tune I/O

Normally, the biggest bottlenecks in recovery are read and write I/O. Use native asynchronous I/O and set the database parameter, `disk_asynch_io=TRUE` (this the default). `disk_asynch_io` controls whether I/O to datafiles is asynchronous. Asynchronous I/O should significantly reduce db file parallel reads and should improve overall recovery time.

#### 5. Monitor Statistics

Monitoring I/O, CPU, recovery process CPU utilization, and database statistics on the standby database to ensure that media recovery is running optimally and that there are no system bottlenecks.

- Apply rate issues are easily detected by monitoring for database and system bottlenecks. Refer to [Recovery Monitoring Best Practices](#) section for more details.
- Reduce the degree of parallelism if CPU over-utilization on the standby database is observed.
- If there are I/O bottlenecks or excessive wait I/Os, then stripe across more spindles/devices or leverage more controllers. A stripe size between 256KB to 1MB is optimal to leverage your I/O subsystem. Verify that this is not a bus or controller bottleneck or any other I/O bottleneck.
- The top database wait events should be “db file parallel read” and ”checkpoint completed” during media recovery. Refer to [recovery monitoring best practices](#) section.
- If the top database wait event is “free buffer waits”, which indicates recovery was not able to find a free buffer, use multiple DBWR (set `DB_WRITER_PROCESSES > 1`) during media recovery. Configuring multiple DBWR processes benefits performance when a single DBWR process is unable to keep up with the required workload. However before configuring multiple DBWR processes, you need to check whether asynchronous I/O is available and configured on the system. If the system supports asynchronous I/O but it is not currently used, then enable asynchronous I/O to see if this alleviates the problem. If the system does not support asynchronous I/O, or if asynchronous I/O is already configured and there is still a DBWR bottleneck, then configure multiple DBWR processes.

- If the top PQ messaging waits are “PX Deq\*”, verify that the fix for bug 2555509 has been installed. If the fix for bug 2555509 has been applied, reduce recovery parallelism or try serial recovery performance.

## 6. Serial Recovery

Testing revealed that serial (no parallelism) media recovery for SQL\*Loader transactions or batch jobs that update contiguous blocks provide near optimal redo apply rates with the least amount of system overhead. Refer to the “[Description of Recovery Test Environment](#)” section for descriptions of OLTP and SQL\*Loader transactions.

## 7. Redo Apply Rates

Redo apply (recovery) rates for Oracle9i Release 2 should surpass very high OLTP redo generation rates and at least match SQL\*Loader or batch generation rates if there is sufficient system resources. Using 8 CPUs, our tests achieved approximately 12MB/sec apply rate for SQL\*Loader runs and 4MB/sec apply rate for OLTP type runs. Refer to the [Detailed Test Results](#) section.

## 8. Expect Larger CPU and Potential I/O Consumption with Higher Degrees of Recovery Parallelism.

With higher degrees of recovery parallelism, our tests showed higher CPU utilization, although different profiles showed different degrees of increase. With the SQL\*Loader profile, the CPU utilization increased gradually from 20% to 30%. However, with the small OLTP profile, CPU utilization increased dramatically from 15% to 50%. Potentially more I/O bandwidth will be required for higher degrees of parallelism if the apply rate increases. Parallel recovery and the recovery slaves consumed more system resources due to the parallel recovery communication and IPC messaging overhead.

## 9. Standby Database System Utilization

Standby database system utilization compared to production database CPU utilization depends dramatically on the read/write ratio on the production database, the type of production transactions, the variance of unique blocks that are being updated in the redo, other additional activities on those hosts, and other factors. There is no simple formula to predict the standby database system utilization. Here are some general observations that we found during our testing.

- The higher the read ratio on production, the greater the difference in CPU utilization between the production and standby databases. We

have experienced 10 times more CPU utilization on the production system with query intensive applications.

- Higher numbers of sorts or complex queries executed will require more CPU utilization on the production database. Queries and sorts do not create additional redo and thus does not create work on the standby database.
- Additional standby database CPU utilization is required when unique blocks are updated to account for the application of redo to the distinct blocks. The SQL\*Loader runs modified 5 times less unique blocks compared to an OLTP run leading to 30 to 40% less CPU utilization on the standby while having almost three times the apply rate of the OLTP runs.

## 10. Configuration Differentiators

Although it is recommended that the production and standby machines and databases are identically configured, you can reduce memory utilization on the standby by decreasing the database shared pool and the buffer caches. To conserve memory resources,

- check shared pool free memory and then reduce shared pool size accordingly
- check the buffer cache hit ratio and possibly lower the buffer cache if recovery rate and hit ratios are high.

Refer to the supplied [recovery monitoring script](#).

## 11. Real Application Clusters (RAC) and Redo Apply Rate

Recovery tests with and without the RAC installed on the standby site showed little difference in recovery rates. Minimal to no apply rate overhead was found when using Real Application Clusters on the standby database; however we recommend using one designated recovery instance.

## TESTING DETAILS

### Description of Test Environment

The profile of the database is a warehouse environment with a simplified OLTP transaction profile, approximately 200GB with full archive files being 1GB in size. The production system consisted of two EMC Symmetrix E3500s in a RAC cluster. Each node has 8 CPUs, 8 GB RAM with the database files striped across 36 spindles over 2 controllers. Stripe And Mirror Everything <sup>2</sup>(SAME) configuration was implemented with a stripe size of 1MB.

The standby system is configured identically to the production environment. Recovery uses half of the CPU, memory and I/O channel resources on the standby host since recovery is limited to one node of the RAC cluster.

### Transaction Runs

The test environment included 3 simulated transaction runs labeled Small OLTP, Big OLTP and SQL\*Loader. The [table](#) below describes the different transaction runs, their respective production redo generation rates on a two node RAC, and the CPU utilization. OLTP type runs updated, queried, and inserted into several large partitioned tables in which some containing 17 million and 170 million rows and many indexes.

**Table 1: Transaction Run Description**

Transaction Run	Redo Generation	Description	CPU Utilization
Small OLTP	500KB/sec	60 concurrent sessions with small think times 2 types of transactions: 1) select; update cust table; commit 2) select, insert ordr table; insert ordl table; commit Changes are scattered. Approximately 107 txn/sec or 2500 block changes/sec	11% on each node ➔ 22% total
Big OLTP	3 MB/sec	100 concurrent sessions with small think times 2 types of transactions: 1) select; update cust table; commit 2) select, insert ordr table; insert ordl table; commit Changes are scattered. Approximately 585 txn/sec or 14000 block changes/sec	65% on each node ➔ 130% total
SQL*Loader	11 MB/sec	4 parallel loads with logging on each node of 2 node RAC selects using index scans	30% on each node ➔ 60% total

<sup>2</sup> For more information about SAME, please refer to

[http://otn.oracle.com/deploy/availability/pdf/oow2000\\_same.pdf](http://otn.oracle.com/deploy/availability/pdf/oow2000_same.pdf)

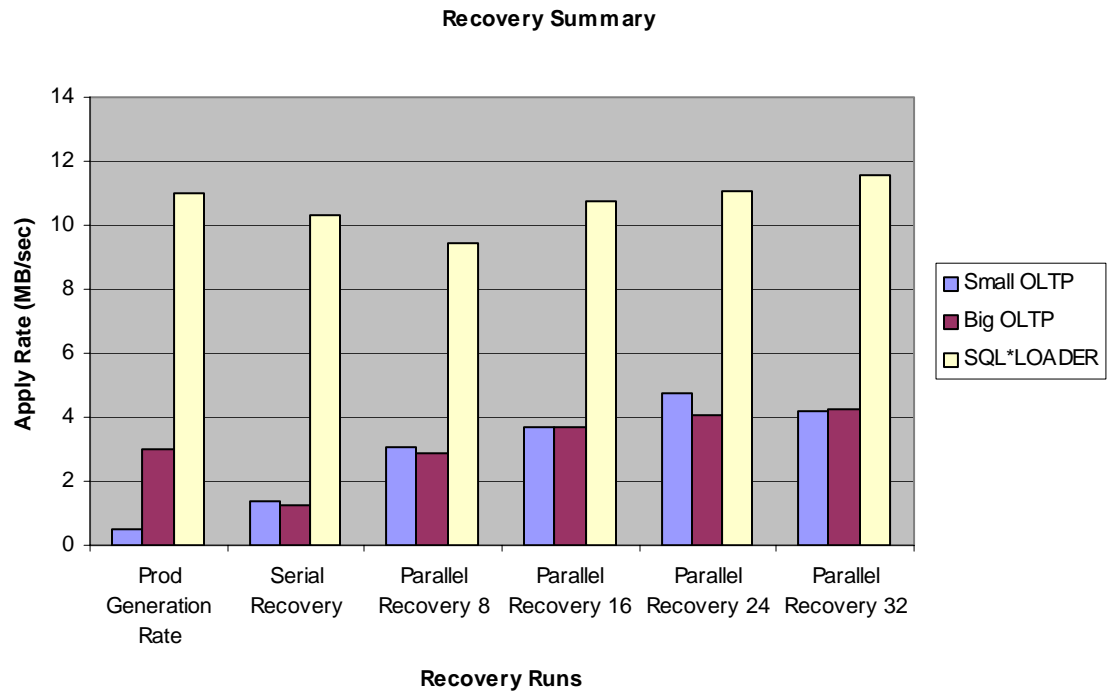
		<p>Changes are clustered together. A lot of sequential changed blocks.</p> <p>Approximately 0.05 txn/sec or 50 block changes/sec</p> <p>The block changes are very dense since these are sequential changes.</p>	
--	--	--	--

The simulated transactions may not be characteristic of typical customer environment; however, they provide good examples of update intensive OLTP profiles and SQL\*Loader /batch-oriented profiles.

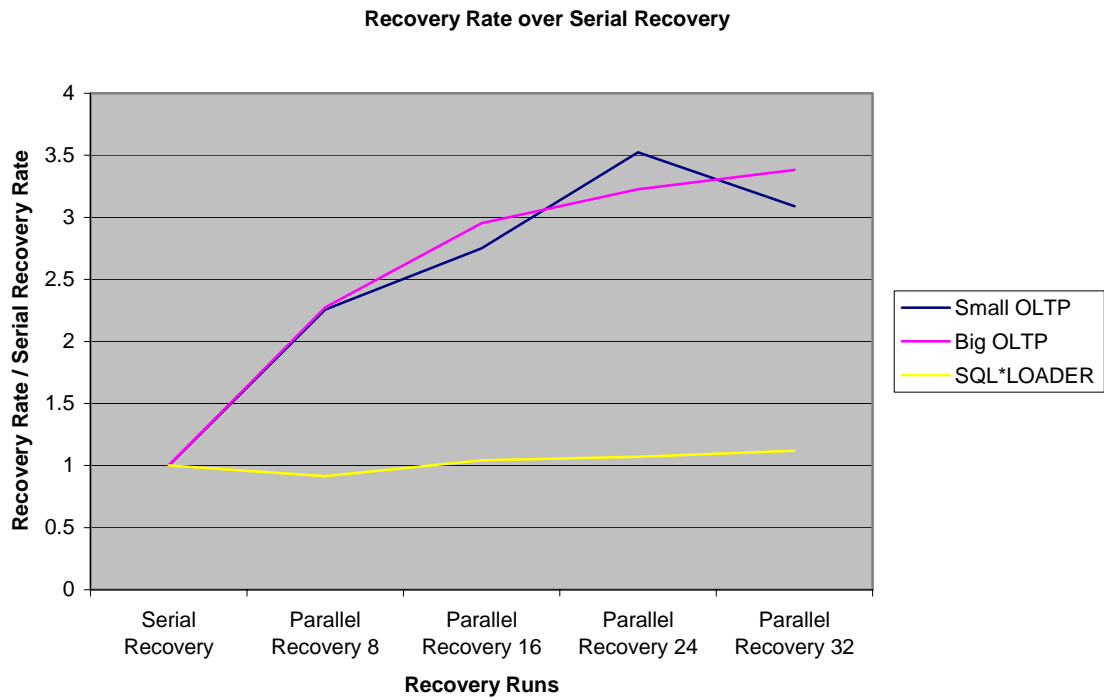
### Detailed Test Results

Parallel media recovery provided good performance scale up for the OLTP runs but very little performance gain for the SQL\*Loader runs. The following graphs show recovery rate improved 2 times for 8 parallel slaves and 3 times for 24 parallel slaves on an 8 CPU machine. If the redo generation rate is less than 500KB, serial recovery will be satisfactory for most systems; however if a databases experiences high redo generation rates for their OLTP applications, then parallel recovery will be required to ensure that the redo apply rate remains faster than the redo generation rate. Recovery parallelism does require more CPU and I/O resources.

Notice that the recovery apply rate for the two OLTP runs are similar since the redo profiles are comparable. The SQL\*Loader runs change a small set of sequential blocks, therefore the apply rates are higher due to more dense changes to a few number of blocks per read. The SQL\*Loader runs received marginal or negative gains with parallel recovery.



**Figure 1: Redo Apply Rate**

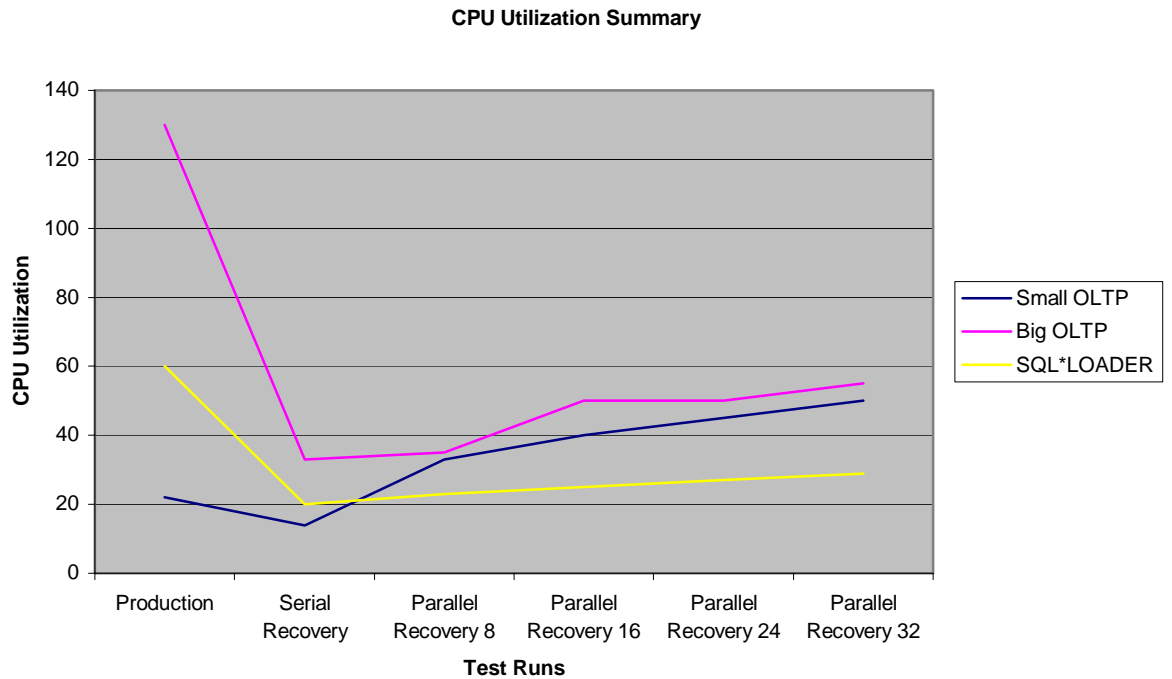


**Figure 2 Recovery Rate over Serial Recovery**

From our testing, we notice that the standby database

- Does not require as much memory resources compared to the production database so decreasing the shared pool size can reduce the memory utilization on the standby.
- Does not require as much CPU resource as the production database but the exact ratio depends on the application profile.
- CPU utilization increases with more recovery parallelism.
- I/O increases with higher apply rates and degrees of parallelism.

The following CPU utilization diagram shows a direct correlation between the degree of parallelism and CPU utilization on the standby host. The derived CPU utilization is the summation of the CPU utilized on a 2 node RAC; hence the Big OLTP production run utilized 130% of the CPU.



**Figure 3 : Recovery Parallelism and CPU Utilization**

### Recovery Monitoring Script and Monitoring Best Practices

To assess the standby database and host performance, monitor the following database statistics and wait events.

- Database wait events from `v$system_events` and `v$session_waits`
  - Refer to the top 3 system wait events and tune the biggest waits first. You can determine the top system and session wait events by querying `v$session_wait` and `v$system_event` and taking the top waits with the largest “TIME\_WAITED” value
  - Once the I/O and CPU resources are tuned, then the biggest database wait events should be “db file parallel write” and “checkpoint completed”
  - If “free buffer waits” is the top database wait event, then increase the number of `db_writer_processes` or the size of the database buffer cache.
  - If PQ messaging (“PX Dequeue\*”) waits are the top wait events, check for bug 2555509 and then attempt to reduce recovery parallelism or try serial recovery performance.
- Database SGA utilization from `v$sgastat`

- Check for excessive shared pool free memory over time. Reduce memory utilization by decreasing the shared pool size.
- Database system values from v\$sysstat
  - Keep a history of v\$sysstat values for more in-depth debugging. Currently there's no statspack utility for the standby database; so, you need to query the v\$sysstat view and load it into a separate read write database.
  - Check if buffer cache hit ratio > 95% and compare if the ratio dropped from previous results.
  - Check if recovery read ratio >= 5 and compare if the value dropped from previous results.
- Database file I/O information from v\$filestat
  - For each file, ensure that read and write I/O timings are reasonable for your system. Refer to a systems tuning guide.
  - Check if batch reads and writes are occurring. Refer to the database Performance Tuning Guide and Reference.
- System CPU utilization
  - Ensure you have sufficient CPU resources or reduce the degree of recovery parallelism
- System Memory utilization
  - Check for excessive paging and swapping.
  - Reduce database parameters such as shared\_pool\_size, sort area size and possibly db\_cache\_size. Refer to the SGA utilization information.
- System I/O utilization
  - Look for excessive wait I/Os or large queues of I/O. Assess if the read I/O and write I/Os times are excessive. Compare with the statistics found in v\$filestat
  - If there are I/O bottlenecks, then assess if more spindles or disks are required. Attempt to use SAME with a stripe size of 1 MB.
  - Verify that the disk controllers or I/O buses are not saturated.
- Recover Slave Process CPU utilization
  - If you notice that the CPU utilization is not evenly distributed across the recovery slaves, then recovery parallelism may not assist you in this environment or at this time. This implies that changes are probably occurring in the same set of blocks and those blocks are being passed to one particular slave or small

subset of slaves. This may be a temporary issue attributed to a specific transaction but may explain a slowdown in recovery.

The database utility, Statspack, does not gather statistics from physical standby databases or from mounted instances. The following script helps extract the most important database statistics; however the statistics, timings and averages are cumulative. You need to manually get the difference between two runs by subtracting the data between two snapshots.

#### **Sample Recovery Monitoring Script - Korn Shell**

Notice that for the UNIX shell, the “\$” is preceded with “\” to ensure that the shell does not attempt to translate the variable.

```
#!/bin/ksh

while true
do
sqlplus '/ as sysdba' << recovery_data
spool $1/dbstats.`date +%b%d_%T`
set pagesize 10000 echo off feedback off TERMOUT OFF

# status of recovery and current sequence# and thread#
select process, status , thread#, sequence#, blocks from v\managed_standby;
select max(sequence#), thread# from v\log_history group by thread#;

# session wait information
column event format a35
column p1text format a20
column p2text format a20
select sid, event, total_waits, time_waited from v\session_event
      where total_waits != 0 and time_waited !=0 and
event not in ('rdbms ipc message','smon timer') and time_waited > 100
order by time_waited;

# file I/O statistics within the database
select * from v\filestat
```

```
where PHYWRTS >0 order by writetim ;
```

```
# system event information
```

```
select * from v\system_event
```

```
where time_waited > 1000 order by time_waited;
```

```
# SGA stats
```

```
select * from v$sgastat;
```

```
# recovery stats such as average recovery read I/O
```

```
select name, value from v$sysstat where name like 'recovery%';
```

```
# buffer cache hit ratio
```

```
select (c.value + d.value - p.value)/(c.value + d.value)
```

```
from v$sysstat c, v$sysstat d, v$sysstat p
```

```
where c.name = 'consistent gets' and
```

```
d.name = 'db block gets' and p.name = 'physical reads';
```

```
spool off
```

```
exit
```

```
recovery_data
```

```
sleep <TIMEOUT_VALUE>
```

```
done
```

## CONCLUSION

By following the above recipe and integrating standby monitoring best practices, you can achieve optimal recovery or redo apply performance. Faster media recovery leads to reduced Data Guard switchover, failover or database media recovery times. This equates to more uptime and higher availability in the case of an outage.



Oracle9i Media Recovery Best Practices

August 2003

Author: Lawrence To, High Availability Systems Team

Contributing Authors: Raymond Dutcher, Vinay Srihari, Tammy Bednar, HA Systems Team

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

[www.oracle.com](http://www.oracle.com)

Oracle is a registered trademark of Oracle Corporation. Various product and service names referenced herein may be trademarks of Oracle Corporation. All other product and service names mentioned may be trademarks of their respective owners.

Copyright © 2002 Oracle Corporation

All rights reserved.