RMAN Backup and Recovery Optimization

An Oracle White Paper
March 2005

RMAN Backup and Recovery Optimization

RMAN Backup and Recovery Optimization

INTRODUCTION

Backup performance is typically scrutinized when backups start to exceed their alloted time windows. With the new `DURATION` parameter in Oracle Database 10g, RMAN can be told how much time to spend on the backup, and when this time has expired, the backup will stop. The next time the backup is run, RMAN will pickup from where it left off. Though this feature allows the total backup window to be segmented into several smaller windows, the problem of optimizing overall backup performance still remains.

Restore and recovery performance issues are often brought to light when it is too late, i.e. the database has been recovering for 6 hours now, and it still hasn't finished, but it needs to be available *now*!

This paper explains the factors that affect the performance of backup, restore, and recovery operations, and provides guidance on proactively monitoring and improving that performance.

This paper assumes that the reader uses Recovery Manager (RMAN) to manage the backups of their Oracle databases, and so performance of user-managed backups (non-RMAN) will not be discussed. All tests carried out in this paper use Oracle Database 10g Release 10.1.0.3 on a dual Intel processor machine running Suse Linux Enterprise Server 7. This was not a high end or large server, so the tests conducted only illustrate on a simple basis, the areas of optimization.

Throughout this paper the amount of CPU time used for an activity is shown during testing. This value is retrieved from `v$sesstat` for the statistic '`CPU used by this session`' (statistic #12) where the session is the one created when a channel is allocated by RMAN. This shows the amount of CPU time used for reading, physical and logical block checking, compression, and writing to the backup set.

NOTE: The test results shown in this paper are not official Oracle performance figures and are simply used to demonstrate the performance characteristics of backup, restore, and recovery operations. Your results will vary greatly depending on your own hardware, operating system, network, and database configurations.

RMAN OVERVIEW

RMAN was introduced with Oracle8 as a tool that allows the DBA to easily manage backup and recovery for their Oracle databases. There have been improvements with each new release of Oracle and because it is built into the RDBMS kernel, it can take advantage of various database features such as corruption validation checks.

For a thorough description of the RMAN architecture and features, refer to the Oracle Database Backup and Recovery Basics Guide. This paper assumes that the reader understands the basic RMAN components and terminology, and will only discuss relevant features when appropriate.

NOTE: In this paper, the term 'backup performance views' corresponds to the `V$BACKUP_SYNC_IO` or `V$BACKUP_ASYNC_IO` views, depending on whether synchronous or asynchronous I/O is being used.

How RMAN Takes a Backup

RMAN allows the DBA to take a backup of data files, control files, SPFILE and archivelog files. The backups created are stored in one of two formats:

- Image Copies – This is an exact, bit-for-bit copy of the file being backed up that is stored only on disk, never written directly to tape. As tuning the performance of image copy creation is a simple disk-to-disk copy exercise, image copies will not be discussed further in this paper.
- Backup Sets – This is a logical grouping of data files, archived logs, control files or SPFILE (data files and archived logs cannot be combined in the same backup set). Backup sets can be created on disk or tape using the Media Management Layer (MML), which is a software layer that provides the interface from RMAN to a tape library. This is the type of backup discussed throughout this paper.

A backup set can contain multiple files, whose data is multiplexed together. When creating a backup set, RMAN does not back up blocks that have never been formatted.

FACTORS AFFECTING BACKUP AND RESTORE PERFORMANCE

There are a number of factors that can influence backup and restore performance:

- Speed of Backup Devices
- Parallelism
- Backup Set Multiplexing
- Buffer Sizes

- Synchronous vs Asynchronous I/O
- Use of Incremental Backups
- Use of Block Checking
- Use of Compression

The process of applying recovery to the database using archived logs and incremental backups will be discussed in a later section.

Speed Of Backup Devices

The maximum speed at which a backup can run will be dictated by:

Max Mb/Sec = min(disk read Mb/s, tape write Mb/s)

It is not possible to make a backup to go faster than this, period.

The first part of a backup, as already explained, involves reading each data file/archived log file and placing the blocks into read buffers. The maximum speed at which these buffers can be filled depends on the physical location/caching and maximum read speed of the disks. The throughput being achieved through RMAN from the disks can be monitored using the `effective_bytes_per_second` column in the `Backup performance` views, where type=`'INPUT'`. If this is lower than the expected read rate advertised for your disks, then you need to start investigating why this is so by looking at OS data like *sar* and at disk monitoring data of the particular disk/logical volume manager implementation.

The speed of the devices used for writing the backup can also be monitored using the backup performance views, in the `effective_bytes_per_second` column where type=`'OUTPUT'`. If you are seeing a slower I/O rate than expected on the tape devices, then you need to look at the MML and drive options to tune the physical tape block size (normally the larger the better), the tape compression (this can slow down backups), and to make sure you are streaming data to the drive. Increasing the number of files multiplexed into each channel may increase the tape streaming abilities, but it will reduce the performance of restoring a subset of the data files contained in one backup set.

Parallelism

A channel is the communication pathway from RMAN to the physical backup device, whether it is disk or tape. This section discusses best practices for allocating channels for tape and disk backup.

Tape Backup

It is recommended to allocate one channel per physical tape device available for the backup.

Each channel creates its own backup set. If you only have two tape devices but you allocate three channels, depending on the MML, either one channel will always be waiting for a device to finish on another channel before it can begin creating its backup set or two backup sets will be interlaced onto the tape. Interlacing backups in this way may provide shorter backup times but will lengthen restore times dramatically.

Disk Backup

It is recommended to allocate one channel per physical device that the output is striped over.

General Information

When using automatic channel parallelism, RMAN attempts to distribute the data files amongst the channels with the aim of creating evenly sized backup sets, and will:

- Split up the number of files to be backed up amongst the channels
- Try to even out the disks being read containing the files amongst the channels
- Attempt to make each backup set the same size

Automatic channel parallelism is documented in Chapter 2 of the Backup and Recovery Advanced Users Guide.

Backup Set Multiplexing

Adjusting the `filesperset` backup option may make differences to the speed of backups but it can make major difference with restore times when not restoring all the data files. This is demonstrated with a test taking several level 0 backups with a different number of files per set. A data file is then deleted and timing information is gathered when using each of the backup sets to restore it. The results are shown below in Table 3:

#Files in BS	BS Size (blocks)	Restored File size (blocks)	Time (secs)	CPU (Secs)
8	702320	97727	132	39.42
4	658221	97727	110	36.92
2	132773	97727	82	29.92
1	97730	97727	74	25.62

Table 3: Effects of `filesperset` *on restore speed*

It is obvious from the results that by keeping the number of filesperset to a smaller value, the speed at which the file can be restored is decreased. If the restore involved the entire database, having more backup sets may increase the time it takes to restore. This is due to more requests being sent to the MML. This may not be a significant performance decrease but should be monitored before permanently reducing the filesperset of your backups.

Another consideration is time taken for resuming a backup that has failed. RMAN resumes the backup at the backup set level, so a large backup set, that was interrupted, will take more time to re-create than a smaller one (i.e. one with a smaller multiplexing value)

Buffer Sizes

When RMAN takes a backup, it must read each block into an input buffer. The block is checked to make sure it needs to be backed up and then various block validation checks are made in order to detect corruptions. The block is then copied into an output buffer. Write buffers are used to store the multiplexed data blocks or archived log blocks, which are then written to the backup media (disk or tape). This is shown clearly in Figure 3 below:

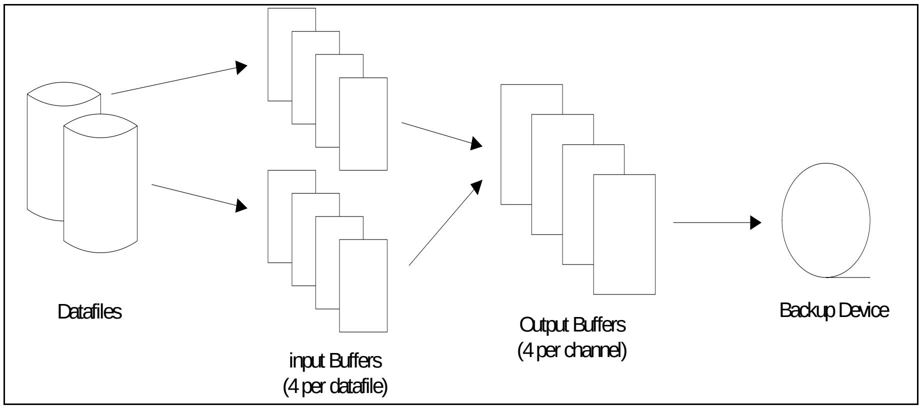

Figure 3: Use of memory buffers during a backup

Backup Reading Buffers

Oracle has documented the size and number of disk reading buffers used during a backup in the Backup and Recovery Advanced User's Guide:

Number of files per Allocated channel	**Buffer size**
Files < 4	Each buffer = 1Mb, total buffer size for channel is up to 16Mb
4 ≥Files ≤ 8	Each buffer = 512k, total buffer size for channel is up to 16Mb. Numbers of buffers per file will depend on number of files.
Files > 8	Each buffer = 128k, 4 buffers per file, so each file will have 512Kb buffer

Table 4: Read buffer allocation algorithm

The parameter that adjusts the number and size of read buffers allocated during the backup is the channel parameter `MAXOPENFILES`. This specifies the maximum number of data files that RMAN will concurrently open for input into the backup. The default value is *min(8, #files in backup set)*.

To show how the algorithm equates to the actual buffer allocation during a backup, the table below shows the buffers being split up amongst data files and backup sets when backing up a database using different numbers for MAXOPENFILES:

MAXOPENFILES	Block size (bytes)	Buffer Size (Kb)	#Buffers per file	#files open at one time	Total Buffer Size (MB)
1	8192	1024	16	1	16
2	8192	1024	8	2	16
3	8192	1024	5	3	15
4	8192	512	8	4	16
5	8192	512	6	5	15
6	8192	512	5	6	15
7	8192	512	4	7	14
8	8192	512	4	8	16
9	8192	128	4	9	4.5
10	8192	128	4	10	5

Table 5: Disk Read Buffer Allocations for Data file Backups

The number and the size of the buffers allocated for each file (data file and archived logs) can be seen using the following query:

```
SELECT set_count, device_type, type, filename, buffer_size,
buffer_count, open_time, close_time
FROM v$backup_async_io
ORDER BY set_count,type, open_time, close_time;
```

The algorithm that Oracle uses to allocate the read buffers seems to be adequate, but it is worth monitoring `v$backup_async_io` to show how much memory is being allocated to your backup read buffers to make sure you don't run into memory starvation issues. When running the tests shown in Table 5, there was no change to the amount of time taken to create the backup.

Backup Writing Buffers

The backup writing buffers are sized differently for tape and disk devices:

Device Type	Number of Buffers Per Channel	Buffer Size	Total of Buffers Allocated
DISK	4	1Mb	4Mb
SBT	4	256Kb	1Mb

Table 6: Default write buffer allocation

Another potential way to increase performance is to use larger write buffers.

To demonstrate this, we ran 3 backups of a database with different write buffer sizes. The first backup was taken using standard buffer settings (1Mb buffer). The next two backups were taken using a small and larger write buffer size, which was altered using the command:

Smaller buffer (32k * 4 = 128Kb buffer):

```
configure channel device type sbt parms=’BLKSIZE=32768';
```

Larger buffer (512k * 4 = 2Mb buffer):

```
configure channel device type sbt parms='BLKSIZE=524288’;
```

The following table showed the results of the backups from the backup performance views:

Total Buffer Size (bytes)	I/O Count	I/O Time (100ths sec)
131,072	60564	6174
1,048,576 (default)	7571	5959
2,097,152	3786	5053

Table 7: I/O Rates with varying write buffer sizes

Where is the memory allocated from?

The memory for the read and write buffers is allocated from the PGA unless I/O slaves are being used. Each channel has a distinct set of I/O buffers. If I/O slaves are used then the buffer memory is allocated from the shared pool to enable data transfer between the primary process and the I/O slave process. If a large pool area is allocated using the LARGE_POOL_SIZE parameter, this will be used instead of the shared pool. It is recommended to use the large pool to reduce contention in the shared pool.

Synchronous vs Asynchronous I/O

When using tape devices, the goal is to keep the tape streaming. Because all tape I/O is synchronous, the channel process will wait for the write to tape to complete before continuing and filling the read buffers. When filling the read buffers, the tape device will wait for the next set of write buffer data. This prevents continuous tape streaming, and is not the optimized way for taking backups.

Therefore, it is recommended to enable tape I/O slaves with the `BACKUP_TAPE_IO_SLAVES` initialization parameter`,` which emulates asynchronous I/O. The channel process allocates writes to the slave processes and does not wait for them to complete before refilling the buffers.

What is Tape Streaming?
If there is enough data being fed to the tape fast enough, the tape drive will write continuously, always having more to write at the next position on the tape after the current write is through. The drive motors can keep spinning continuously at full write speeds, so the tape just 'streams' by the heads being written.
If there is not enough data to keep the tape streaming the tape drive will normally stop when no data is received and rewind back to the place of the last block written, and then wait for new data to arrive. Constantly moving the tape forward and back reduces the life of the tape, and can also slow down backup times.

Use of Incremental Backups

Probably one of the best ways to reduce the time to carry out the backups is to simply reduce the amount of data being backed up.

Here are a few guidelines:

- If there is any static data contained in the database that never changes, or changes very infrequently, such as price lookup or definition data, then place it within its own tablespace. This tablespace can be made read-only and then backed up. When it is time to make changes to the data, put the tablespace back into read-write mode, make the changes, put back into read-only mode and take another backup of it.

  NOTE: When backing up tablespaces very infrequently, you must make sure the backup is not purged from the tape library based on the date at which it was created. If this happens you will no longer have the data needed for recovery. If there is a limit placed on the age of backups, then make sure that a fresh backup of the read-only data is taken before the expiration. RMAN can do this automatically when using the Backup Optimization feature.

- If your backup strategy includes differential incremental backups, you can query the database to find out how many blocks usually change between backups. By using the query listed below, it is possible to identify data files that are not changing much, and would be candidates to being backed up less frequently.

  ```
  SELECT set_count, set_stamp, file#, data file_blocks,
  blocks “BACKUP_BLOCKS”
  FROM v$backup_data file
  ORDER BY set_count, file#;
  ```

- Full backups of large data files that contain very little data, versus those with a large amount of data, can still be time-consuming, since each data block is read into the memory buffer and evaluated to see if it has ever been used or never used.

To demonstrate this, the following results compare taking a full backup of a 500Mb file with different free space counts:

%Free space	Number of Blocks Scanned	Number of Blocks Backed Up	Time Taken (secs)
100%	64000	8	22
50%	64000	31678	31
<0.79 %	64000	63496	49

Table 8: Backup time and data file freespace

On the small test system that was used backing up to disk, the amount of time was only 49 seconds in the worst case, but you can see that even when the file was empty, it took 45% of the worst case to backup only 8 blocks. Imagine this on a busier system, with larger data files and many more of them.

If you have large data files (2Gb or greater) which contain large free space amounts, it is worth while resizing them (if there are no object extents at the end of the file) and then setting them to autoextend to prevent wasted time in scanning empty space during the backup.

- Consider using Block Change Tracking introduced in Oracle Database 10g. This allows backups using the Optimized Incremental feature. Instead of scanning an entire data file during a backup, with block change tracking enabled, the change tracking file keeps track of the locations of changed data blocks using bitmap patterns to represent ranges of data blocks. There is very little performance overhead when tracking changed blocks.

  The improvement in backup speed using the *optimized incrementals* with the block change tracking enabled was dramatic. To test this, the database was first backed up using a level 0 incremental, which backs up the entire database. DML activity was generated in the database and then a level 1 incremental was taken. The exercise was repeated with block change tracking disabled and then enabled. The table below shows the results:

Optimized Incrementals?	#Blocks in Database	#Blocks Read	#Blocks in Backup	Time Taken (secs)
No	404160	404160	36567	156
Yes	404160	72832	37215	35

Table 9: Speed increase using Optimized Incremental backups

It is clear to see the increase in speed due to the reduced number of blocks being read when using the optimized incremental backups.

To check if a backup used the optimized incremental feature, look at the `USED_CHANGE_TRACKING` column in `v$backup_data file`.

The speed of restore is not affected by using optimized incremental backups, because the resulting backup set remains the same as if it were backed up using normal incremental backups, just created quicker.

Use of Block Checking

RMAN offers the ability to run several different block sanity checks to detect any corruptions at time of backup and restore.

Checksums

By default, RMAN calculates checksums for every block written as part of the backup, irrespective of the `db_block_checksum` parameter setting. If the block already contains a checksum value, RMAN will verify that it is correct. The checksum creation can be turned off by specifying the `NOCHECKSUM` clause as part of the backup command. However, this is not recommended. Checksums are always created for the `SYSTEM` data files and the control file.

Logical Block Checks

RMAN also offers a feature to check the logical structures contained within each data block being backed up. The checks are the same logical block checks as those used when the `db_block_checking` initialization parameter is used. To turn on logical checking during the backup, use the `CHECK LOGICAL` option with the `BACKUP` command. By default, logical checking is not used.

To demonstrate what effects on backup performance each check has, I ran a full database backup with each option enabled. The results are shown below:

Checksums?	Check Logical?	Blocks read	Time (secs)	CPU (secs)
No	No	825920	623.67	227.08
Yes	No	825920	622.33	229.93
Yes	Yes	825920	624.67	245.81

Table 10: Effect of checksum and logical block checks on backups

Table 10 shows that checksum validation has little effect on backup performance. Each data block already contains a checksum due to the default of init.ora value of `DB_BLOCK_CHECKSUMS` being set to `TRUE` so when RMAN reads each block, it validates that the block's checksum is correct. It is not recommended to turn off the checksums due to the value added by this RMAN feature in block corruption detection during backups.

Adding the `CHECK LOGICAL` parameter to the RMAN backup has a small but noticable impact on backup performance. This should be tested and benchmarked in your own environment to see what the performance effects are.

The effects of checksums and logical block checks are similar during RMAN restores:

Checksums?	Check Logical?	#Blocks in DB	Time (secs)	CPU (secs)
No	No	825920	559	210.53
Yes	No	825920	559	211.14
Yes	Yes	825920	610	218.53

Table 11: Effect of checksum and logical block checks on restores

Use of Compression

Before Oracle Database 10g the only type of compression that RMAN would use was that of unused block compression – unused blocks were not backed up. In Oracle Database 10g a binary compression feature was introduced that will compress backup set sizes by 50+%. Due to the increased amount of work involved in the compression, backup and restore times will increase. CPU usage will also increase.

If the MML also offers tape compression, testing should be carried out to see if it offers a better compression ratio and backup time than using RMAN compression. You should never use compression with RMAN and the MML due to decreased performance during backup and restore. It would be wise to test the time and CPU used for both RMAN and MML to see which offers the best value.

If you don't have any concerns about the amount of space a backup set occupies, then don't use compression.

Restoring a compressed backup set has similar performance effects on the CPU usage and overall time:

Compression	#Blocks Read in Backup Set	Backup Set Size (Mb)	Time (secs)	CPU (secs)
No	778109	6079	559	209.37
Yes	778109	517	1133	1093.37

Table 13: Effect of RMAN compression on restore speed

Similar to the backup compression tests, the time it takes to restore the backup is double, with a higher amount of time spent on the CPU.

The use of RMAN compression would be advantageous if the tape devices are mounted across the network. If this is the case, RMAN must transfer the backup pieces from the database server to the tape server. The use of compression will significantly reduce the amount of data traveling across the network, possibly reducing the backup and restore times as well as causing less inconvenience to other network users.

FACTORS AFFECTING MEDIA RECOVERY PERFORMANCE

There are a number of factors that will affect the performance of recovery, including:

- Number of Archived Logs and/or Incrementals Applied
- Number of Data Files Needing Recovery
- Archived Log Location
- Parallel Recovery
- General Database Performance

This section is making the assumption that at this stage of recovery, the data files have been restored from RMAN using a base level backup (level 0 or a full backup).

The Number of Archived Logs and/or Incrementals Applied

After the base level backup has been restored by RMAN, the recovery phase begins. RMAN will first look for any suitable incremental backups that can be used to roll the database forward. If no suitable incrementals backups are found, the required archived logs are retrieved (assuming you are running in ARCHIVELOG mode). It has long been documented that recovering the database using incremental backups is faster than using archive redo log files. This can easily be tested, by taking a base level 0 backup of the database in ARCHIVELOG mode. For a period of time run DML against the database so that it creates between 10 and 20 archived logs, and then take an incremental backup. Delete the database and restore the base level 0 backup. Then use RMAN to recover the database, which will restore the incremental backup, and time it. Restore the base level 0 again, but instead of using RMAN to recover the database, use SQL*Plus with a 'RECOVER AUTOMATIC DATABASE' command, and compare the time it takes with the incremental restore. I ran this test, and found the incremental restore to take 46 seconds compared to the recovery through applying archived logs taking 789 seconds.

On the test system the backups were created on disk, so the speed of restoring incremental backups was predictably much faster. If the incremental backups were coming from a tape device or tape silo that had to locate the tape, mount the tape and then find the start of the backup, it may have taken much longer. The point is, on each system the speed at which archived logs and restores from the backup media are going to vary significantly, so it is worth some investment in some time to test the differences. Only then can you implement the backup strategy that will meet the time expectations when a failure occurs.

As documented in an earlier test, the type of incremental backups (cumulative or differential) will also help determine if applying archived logs are slower than restoring an incremental backup.

Number of Data Files Needing Recovery

When media recovery is carried out, each data block affected by a redo record must be read into the buffer cache before the redo can be applied to it. If you recover more data files than are necessary, you are causing recovery to do more work than necessary. Only restore and recover the minimum amount of data files needed to fix the failure – there is no point restoring all 100 data files if only 10 need recovery.

If the failure is due to block corruptions within a data file, consider using Block Media Recovery, introduced in Oracle9*i*. Instead of restoring and recovering a whole data file, individual data blocks can be restored from a backup, and the archived logs and/or incremental backups are used to recover them.

To show the amount of time block media recovery can save, a test that corrupted a different number of data blocks in a 102,400 8Kb-block data file was carried out to show the total recovery time (including the restore) between block media recovery and data file recovery. The block corruptions were spread evenly throughout the data file. Before corrupting the blocks, a level 0 backup was taken of the data file, the TPCC tests were run against the database (changing 34,113 data blocks in the data file), and all the archived logs remained on disk so they didn't need to be restored during recovery.

Number of Corrupt Blocks	Data file Recovery Time (secs)	Block Media Recovery Time (secs)
10	941	145
99	925	155
991	937	219
5000	922	616
10000	938	1156

Table 14: Speed of block media recovery compared to data file recovery

As can be seen in Table 14, block media recovery showed a significant time saving for recovering individual blocks over restoring and recovering the whole data file. There will be some point at which block media recovery becomes more expensive than recovering the whole data file, and in the test this was seen somewhere around 10,000 blocks which equates to approximately ~10% of the data file. The point of diminishing return for block media recovery may vary for your own system. However this test demonstrates that using block media recovery on too many blocks within a data file can, in fact, be slower than restoring and recovering the whole data file.

Archived Log Location

If the archived logs needed for recovery are already on disk, recovery will complete faster than if the archived logs need to be restored from a backup. If backup compression (RMAN or MML) is being used, recovery times will take even longer. On a simple test to apply 20 archived logs or restore them and apply them on a test system, it took over 2 minutes extra to restore them, and this was using a disk backup. This time would be greatly increased if the backups are on tape and you need to apply a hundred or more archived logs.

It is a good idea to keep a number of archived logs available on disk to aid in recovery speed, should they be needed. The number of log files to keep depends on the frequency of backups and available disk space to store them.

Parallel Recovery

By default Oracle uses a single process, which is the one issuing the recovery command, to carry out media recovery unless using `PARALLEL_AUTOMATIC_TUNING` init.ora parameter. This single process will read the archived logs and determine which changes are required for the specific data files being recovered. The data block is read into the buffer cache, and the redo is applied to it. Each time an archived log is completely applied, a media recovery checkpoint occurs which signals DBWR to write all the dirty blocks in the buffer cache to the data files. The file headers and control file also get updated with checkpoint information. This is a lot to do for a single process, especially when applying a large amount of redo to recover a large number of data files.

To decrease the time it takes to carry out media recovery, Oracle provides the Parallel Recovery feature. On multiple CPU machines, you can specify a degree of parallelism for use with the RECOVER command. The process that issues the recovery command reads the archived logs as before, but instead of reading the blocks in the cache and applying the redo to them directly, it passes that work to the parallel execution slave processes. To make sure the redo is applied to the data files in SCN order, the parallel slaves work on different ranges of data blocks, so they will not interfere with each other, and also the redo will still get applied in SCN order for each data block.

When recovering a small number of data files using parallel recovery it may take longer to perform due to the extra computation involved in partitioning the work, plus the extra IPC communication between the slave and coordinator processes. Monitor `v$session_wait` and `v$system_events` for the top wait events. If you are seeing ‘`PX Deq’` events, for which there are several different types, then reducing the degree of parallelism may increase the recovery time.

You will also notice an increase in CPU usage when using parallel recovery due to the fact that the coordinator process is calculating the work partitioning for each redo change in the archived log, and there are more processes carrying out the recovery. As long as the CPU is not being saturated (look at something equivalent to '`sar –u`'), then parallel recovery is not causing CPU issues.

The final thing to note about parallel recovery is the way it uses memory. If the init.ora parameter `PARALLEL_AUTOMATIC_TUNING` is set to `FALSE` (its default value) the buffers used for messaging between the parallel processes is 2148 bytes and is allocated from the shared pool. If the `PARALLEL_AUTOMATIC_TUNING` is set to `TRUE`, the default buffer is 4096 bytes and allocated from the large pool. By adjusting the size of the buffers, with `PARALLEL_EXECUTION_MESSAGE_SIZE` init.ora parameter`,` the speed of parallel recovery may be decreased, but the use of memory in the shared or large pool should also be monitored to prevent resource starvation. For more information on media recovery, take a look at the MAA white paper on best practices for recovery.

General Database Performance

If the database performs slowly in day-to-day activities you shouldn't expect a fast recovery time. Recovery uses the underlying database server processes to carry out its tasks, so if there general performance issues already exist, recovery may have similar issues.

Common areas that should be optimized that can help recovery times include:

- I/O – Recovery is very read and write intensive due to having to read all the archived log contents, read the data blocks from the data files, and then write the dirty blocks backup to disk once the redo has been applied. By monitoring `v$filestat` along with OS-specific tools, you need to make sure the read and write times are acceptable for the hardware being used. Slow I/O can significantly slow down the time it takes to carry out media recovery.
- DBWR performance – DBWR's main responsibility is to ensure there are enough clean buffers in the buffer cache to be used when data is being read from the data files. When a block has been updated, it is DBWR's job to write the buffer to disk and return it for reuse within the buffer cache. If DBWR cannot keep up with cleaning the buffers, you will notice waits for the 'free buffer waits' event. If the I/O is not a problem, then using multiple DBWR process (or DBWR slaves if your OS does not support asynchronous I/O or asynchronous I/O is disabled) should reduce the waits.
- CPU Utilization – Because each data block that requires recovery is read into the buffer cache before the redo change is applied to it, there are a number of latches that must be acquired first. This includes the *cache buffers*

chains and the *cache buffers lru chain*. Acquiring latches and making the changes to the blocks all takes CPU cycles, so you should make sure there is enough CPU bandwidth for the database during media recovery. It is not uncommon to see a server with a slow trickle of transactions with lots of CPU to spare, turn into a CPU hog during media recovery due to having to apply a much more concentrated amount of data block changes. If you are using parallel recovery, CPU usage will be even higher as already discussed.

CONCLUSION

This paper has demonstrated a number of areas to consider to optimize the speed of backups, restores, and recoveries.

The speed of backup and restore activities can be controlled by:

- Using incremental backups, and cumulative incremental backups if restore speed is more important than backup speed
- Allocating one channel to each available tape device
- Increasing the size of the memory buffers used when creating backups to tape
- Reducing the amount of data that needs to be backed up – scanning empty or static data files can waste significant time and resources
- Using the Block Change Tracking feature to substantially increase performance of incremental backups
- The type of block checking features enabled for backup and restore
- The type of compression being used by RMAN or the Media Management Layer

The speed of media recovery can be controlled by:

- The number of archived logs that need applying with the greater the number, the slower recovery will take
- The number of incremental backups that need to be restored and applied
- The type of incremental backup being used (differential or cumulative)
- The number of data files needing recovery. A higher number data files that are restored and recovered, means a higher number of data blocks need to be read into the buffer cache and written back to the data files
- Using block media recovery instead of restoring and recovering whole data files due to corruption issues
- Using parallel recovery, which partitions the work of applying redo between parallel execution slave process to decrease the time it takes to apply the redo

- General database performance tuning. If the database performs slowly, so will the recoveries, so the database should be optimized before beginning recovery optimization.

The tests carried out in this paper demonstrate the key points to optimization, and it should be understood that each system will offer different performance gains by making any of the suggested adjustments. The speed of backup, restore and recovery is heavily dependent on the speed of the hardware being used by the system, plus any network latency introduced when the backups are stored on remote tape servers.

Backup, restore, and recovery should be thoroughly tested, not just to ensure it will protect you against failures, but also to make sure that backups occur in the allotted time window and that restore and recovery can conform to your Service Level Agreement.

RMAN Backup and Recovery Optimization
March 2005
Author: Stephan Haisley
Contributing Authors: RMAN Development

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

Copyright © 2005, Oracle. All rights reserved.
This document is provided for information purposes only and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission. Oracle, JD Edwards, and PeopleSoft are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.