Oracle9*i* Fast-Start Checkpointing Best Practices

An Oracle White Paper
August 2003

Table of Contents

Oracle9*i* Fast-Start Checkpointing Best Practices

EXECUTIVE OVERVIEW

The importance of high availability varies from application to application. However, the need to deliver increasing levels of availability continues to accelerate, as enterprises reengineer their solutions to gain competitive advantage. Most often these new solutions rely on immediate access to critical business data. When data is not available, the operation can cease to function. Downtime can lead to lost productivity, lost revenue, lost customer goodwill, bad press, and lawsuits.

One of the most common causes of application downtime is a system fault or crash. System faults are the result of hardware failures, power failures, and operating system or server crashes. One of the primary goals of Oracle9*i* is to make system fault recovery as near instantaneous as possible. Not only is Oracle9*i* system fault recovery extremely fast, but the database administrator, making recovery time predictable, and improving the ability to meet service level objectives, can also bound it.

Oracle's Fast-Start Fault Recovery technology enables the Oracle database to recover from crash failures after completing the roll forward phase, without having to do the roll back phase. This enables applications to be up and running much faster. This paper presents fast-start checkpointing best practices to aid you in meeting your recovery time objectives when an unexpected crash may occur.

Fast-Start Checkpointing

Fast-start checkpointing refers to the periodic writes by the database writer (DBWn) processes for the purpose of writing changed data blocks from the Oracle buffer cache to disk and advancing the thread-checkpoint. Setting the database parameter FAST_START_MTTR_TARGET to a value greater than zero enables the fast-start checkpointing feature.

Fast-start checkpointing should always be enabled in Oracle9*i* for the following reasons:

- It reduces the time required for cache recovery, and makes instance recovery time-bounded and predictable. This is accomplished by limiting the number of dirty buffers (data blocks which have changes in memory that still need to be written to disk) and the number of redo records (changes in the database) generated between the most recent redo record and the last checkpoint.

- If the system is not already near or at its maximum I/O capacity, fast-start checkpointing will have a negligible impact on performance. Although fast-start checkpointing results in increased write activity, there is little reduction in database throughout, provided the system has sufficient I/O capacity.
- Fast-Start checkpointing eliminates bulk writes and corresponding I/O spikes that occur traditionally with interval-based checkpoints, providing a smoother, more consistent I/O pattern that is more predictable and easier to manage.

If the time to recover from an instance failure is not critical to meet the required service levels, then FAST_START_MTTR_TARGET should be set to its highest value, which is 3600 (seconds). Using a value greater than 0 (zero) enables fast-start checkpointing for the database. A large value of 3600 will not lead to an MTTR of 1 hour in the case of instance recovery for reasons discussed below.

If determining the time to recover from an instance failure is a necessary component for reaching required service levels, then FAST_START_MTTR_TARGET should be set to the desired MTTR. For example, if service levels dictate that when a node fails instance recovery time can be no more than 3 minutes, FAST_START_MTTR_TARGET should be set to 180. If your chosen value has caused I/O to increase beyond capacity, the detailed MTTR sections below describe how the setting of FAST_START_MTTR_TARGET can be relaxed while still achieving a lower MTTR if in a RAC environment.

When enabling fast-start checkpointing, the following initialization parameters should be removed or disabled (set to 0):

```
LOG_CHECKPOINT_INTERVAL
LOG_CHECKPOINT_TIMEOUT
FAST_START_IO_TARGET
```

FAST_START_IO_TARGET has been deprecated in favor of FAST_START_MTTR_TARGET.

Although LOG_CHECKPOINT_INTERVAL and LOG_CHECKPOINT_TIMEOUT can also be used to control checkpointing to some degree (hence influencing MTTR), neither takes into consideration the number of data blocks that correspond to the amount of redo generated since the last checkpoint, nor do they consider the time it actually takes to read redo blocks or process data blocks during recovery. This results in checkpoint behavior that remains static as the environment changes, making it unfeasible to target a specific MTTR.

When fast-start checkpointing is enabled, Oracle automatically maintains the speed of checkpointing so that the requested MTTR is achieved. Setting a non-zero value for LOG_CHECKPOINT_INTERVAL or LOG_CHECKPOINT_TIMEOUT interferes with fast-start checkpointing, resulting in a different MTTR than expected.

Fast-Start Checkpointing and Performance

When using fast-start checkpointing, performance may be influenced in the following ways (as compared to typical interval-based checkpointing):

- Instance and crash recovery times are reduced significantly
- Database throughput is not affected unless using an aggressive FAST_START_MTTR_TARGET setting
- The number of physical writes to the disk increases
- The I/O profile is smoother, more consistent, and more predictable

Using a high-volume OLTP-type workload on a 2-node RAC cluster, experiments revealed that fast-start checkpointing has an insignificant impact on database throughput, as the table below shows. Some impact was observed for extremely aggressive FAST_START_MTTR_TARGET settings. Even with the most aggressive setting, only 6% throughput degradation was noticed. However, with fast-start checkpointing, database writer (DBWn) writes more aggressively to limit the number of dirty buffers in the cache. Note that writing a buffer from the cache does not equate to aging a buffer from the cache. Most hot buffers will remain in the cache but will be written to disk frequently to advance the checkpoint.

The following table shows a few points that are explained below, in particular the upper and lower bounds on MTTR and the need to exclude instance startup time and data file open time for instance recovery in a RAC environment.

Table 1: FAST_START_MTTR_TARGET Performance on 2-node RAC cluster

FAST_START_MTTR_TARGET	ESTIMATED_MTTR	Instance recovery time (secs)	Through-put (tps)	Physical writes per second
0 (fast-start checkpointing disabled)	1270	640	235	358
900	360	144	236	596
300	298	123	237	626
240	235	104	236	676
180	174	97	236	722
120	112	71	233	820
90	81	64	229	894
30 (reset to 71 automatically)	61	59	222	1145

Instance recovery times with fast-start checkpointing, even with a relaxed FAST_START_MTTR_TARGET setting, are significantly better than with fast-start checkpointing disabled.

Even though instance recovery times decreased and the number of physical writes increase as FAST_START_MTTR_TARGET becomes more aggressive, throughput is not impacted until fast-start checkpointing becomes very aggressive, provided sufficient I/O bandwidth exists.

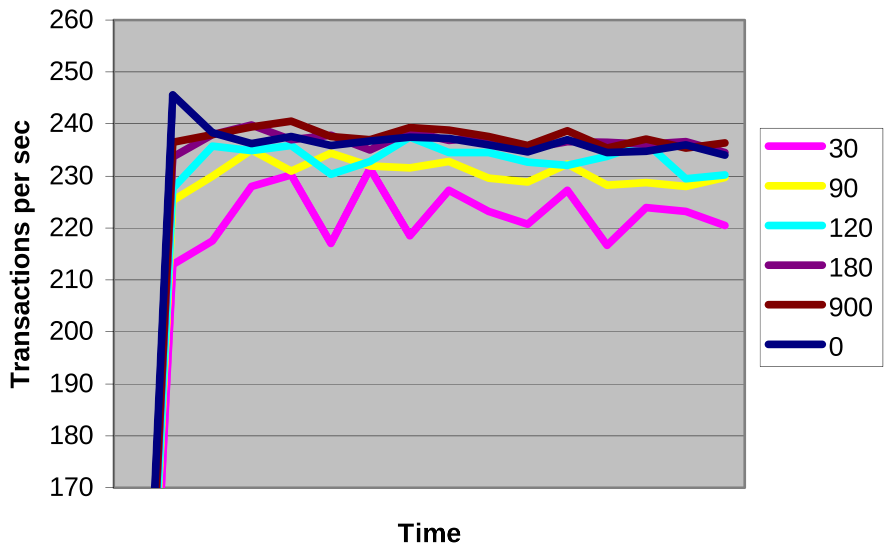

Figure 1: Fast-Start Checkpointing Transaction per Second

The figure above shows that fast-start checkpointing increases the number of total writes to the data files. However, fast-start checkpointing results in a smoother, more consistent I/O pattern, as depicted in the figure below.

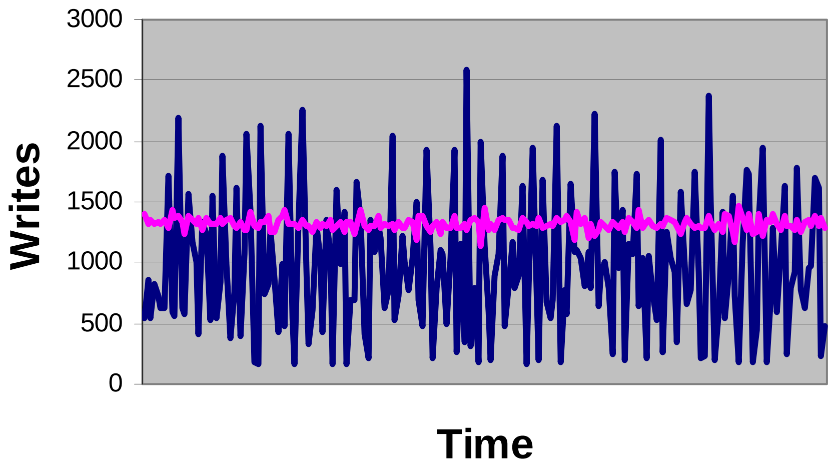

Figure 2: Effects of Fast-Start Checkpointing on I/O
----------- Fast-start checkpointing disabled [more linear line]
----------- Fast-start checkpointing enabled [more erratic line]

Overview of MTTR and Crash Recovery Components

FAST_START_MTTR_TARGET is designed to control the time it takes the database to perform crash recovery for a single instance. Crash recovery implies that there are no running instances and that an instance restart is a necessary step. The main components of crash recovery are:

- Instance startup
- Opening of all database files
- Cache recovery (rolling forward)
- Transaction recovery (rolling back)

There is a timing associated with each component. However, since transaction recovery is done by SMON in the background along with normal user activity after the database is opened, it is not factored into Oracle's fast-start checkpointing calculations.

The timings for instance startup and the opening of all datafiles are taken from the actual values for the currently running instance. For example, if it takes 50 seconds to start up the instance and 10 seconds to open and set up all the datafiles, those are the values used in calculating how aggressively the checkpoint needs to advance in order to ensure MTTR is within the FAST_START_MTTR_TARGET setting.

Cache recovery is determined by two factors:

1. The amount of redo (changes in the database) that needs to be processed during recovery. This is determined by the number of redo

log blocks between the last thread checkpoint and the end of the redo log.

2. The number of data blocks that need to be processed during recovery, which is determined by the number of dirty blocks (changes to blocks in memory which have not yet been written to disk) in the cache at the time of the crash.

With fast-start checkpointing, Oracle automatically advances the thread checkpoint to control the amount of redo needed for recovery and limit the number of dirty buffers remaining in the cache so that recovery time is bounded.

For example, in a large environment it may take 50 seconds to start up the instance and 10 seconds to open and set up all the datafiles. As noted above, the final component of the MTTR calculation is cache recovery, or the roll forward phase. With FAST_START_MTTR_TARGET set to 180, the Oracle server would use fast-start checkpointing to target a cache recovery time of 120 seconds.

Oracle maintains statistics about previous recoveries to estimate how long a future recovery would take. The statistics are weighted such that as more, larger recoveries are completed, the estimates are more accurate. This information is stored in the control file and is specific to each thread.

Upper and Lower Bounds on MTTR

The range of valid values for FAST_START_MTTR_TARGET is 0 to 3600 seconds. A setting of 0 disables fast-start checkpointing and is not recommended because it results in large instance recovery times and inconsistent I/O patterns.

Lower Bound on MTTR

A very low setting provides a low MTTR, but is limited by the low limit of the target number of dirty blocks in the cache, which is 1000. Remember that included in the FAST_START_MTTR_TARGET setting are the instance startup and datafile open values, so the lowest setting possible must factor in instance startup time, plus datafile open time, plus the time it would take to recover the minimum 1000 dirty blocks. If FAST_START_MTTR_TARGET is set to a value below this lower threshold, Oracle will automatically reset it to the lowest achievable MTTR and report the following message in the alert log:

```
FAST_START_MTTR_TARGET 30 is out of the valid MTTR range, use 71
instead.
```

This means that FAST_START_MTTR_TARGET was set to 30, but the lowest achievable MTTR is 71.

Upper Bound on MTTR

The upper limit for setting FAST_START_MTTR_TARGET is 3600. There are additional factors that may place a reduced upper bound on the

ESTIMATED_MTTR that will be maintained. For example, the amount of redo needed for recovery will be limited to 90% of the smallest redo log file, so small redo logs can artificially cap the upper bound for MTTR.

Additionally, a limit is placed on the time it can take for cache recovery to read all of the redo blocks required for recovery. Placing the redo log files on very fast disks reduces the effects of this limit. Although FAST_START_MTTR_TARGET will not be reset to a lower value, the ESTIMATED_MTTR will be maintained at the lower MTTR. For example, with a FAST_START_MTTR_TARGET setting of 900, V$INSTANCE_RECOVERY.ESTIMATED_MTTR may be consistently at 650 if one of these factors is placing an upper bound on MTTR.

FAST_START_MTTR_TARGET in a RAC Environment

As stated above, FAST_START_MTTR_TARGET is based upon crash recovery for a single instance. With RAC, recovery of a failed instance is done by another running instance. Since the recovering instance is typically already running and already has the datafiles open, these two phases need not be included when determining the time it will take to do instance recovery for the failed instance. However, these two components are still reflected in the FAST_START_MTTR_TARGET calculations and V$INSTANCE_RECOVERY.ESTIMATED_MTTR value.

Note that this discussion pertains to a RAC environment where a single instance crashes and is recovered by another surviving instance.

Determining FAST_START_MTTR_TARGET in a RAC Environment

To target an expected cache recovery time when an instance is recovering for a single failed instance in a RAC environment, the instance start time and the datafile open time must be subtracted from FAST_START_MTTR_TARGET. Instance startup time can be taken from the INIT_TIME_AVG column of the X$ESTIMATED_MTTR view. Datafile open time can be determined by multiplying the number of datafiles by the time it takes to open a single file. The time it takes to open a single file can be taken from the FOPEN_TIME_AVG column of the X$ESTIMATED_MTTR view. Both INIT_TIME_AVG and FOPEN_TIME_AVG are in microseconds. This leads to this formula for determining a proper FAST_START_MTTR_TARGET setting for RAC environments:

```
FAST_START_MTTR_TARGET=

desired_MTTR +

X$ESTIMATED_MTTR. INIT_TIME_AVG/1000000 +

X$ESTIMATED_MTTR.
FOPEN_TIME_AVG/1000000*number_of_datafiles
```

Example

For example, if INIT_TIME_AVG=50200342, the time to start the instance is slightly more than 50 seconds. If there are 500 datafiles and FOPEN_TIME_AVG=20120, then it takes about 10 seconds to open all datafiles. Since these activities do not occur for instance recovery in a RAC environment, this can be subtracted out when estimating the MTTR for *cache* recovery of a single instance. If a desired MTTR (the time desired to recover a failed instance) is 120 seconds, then FAST_START_MTTR_TARGET should be set to 180.

```
FAST_START_MTTR_TARGET=

120 + 50200342 / 1000000 + 20120 / 1000000 * 500


FAST_START_MTTR_TARGET=180
```

Monitoring MTTR in a RAC Environment

When monitoring MTTR, the column ESTIMATED_MTTR in V$INSTANCE_RECOVERY reports the expected MTTR if a crash were to occur right now. As with FAST_START_MTTR_TARGET, instance startup and datafile open timings are included in this value. In a RAC environment, to determine a proper expected MTTR, instance startup and datafile open times must be subtracted from the ESTIMATED_MTTR value.

```
Expected MTTR=

ESTIMATED_MTTR -

X$ESTIMATED_MTTR. INIT_TIME_AVG/1000000 -

X$ESTIMATED_MTTR.
FOPEN_TIME_AVG/1000000*number_of_datafiles
```

Determining a Proper FAST_START_MTTR_TARGET Value for Your Environment

Fast-start checkpointing should always be enabled, meaning that it should always have a value greater than zero. A proper setting for FAST_START_MTTR_TARGET depends on whether instance recovery time is a critical component in meeting your service levels.

When Instance Recovery Time is Not Critical

Set FAST_START_MTTR_TARGET = 3600

If instance recovery time is not a critical component in meeting your service levels, then FAST_START_MTTR_TARGET should be set to 3600.

When Bounding Instance Recovery Time is Necessary for Meeting Service Levels

Set FAST_START_MTTR_TARGET to your desired MTTR

If, however, MTTR from an instance failure is a necessary component in meeting your service levels, then set FAST_START_MTTR_TARGET based on the formula above. For example, if your desired MTTR is 120 seconds, instance startup time is 50 seconds, and the time to open all datafiles is 10 seconds, then FAST_START_MTTR_TARGET should be set to 180.

Determine the MTTR Achieved

Test under load to determine if the desired MTTR is achievable using the calculated setting. Also monitor the I/O performance to validate that the increased I/O does not put the system above its I/O capacity. It is important to test MTTR under your expected load because ESTIMATED_MTTR is a dynamic value that changes as workload changes.

The MTTR that is achieved using a specific setting is determined by using the following testing method:

1. Start all instances in your environment using the same FAST_START_MTTR_TARGET setting for all instances.
2. Generate load against the database and monitor I/O load.

3. Once the system reaches steady state and is at peak load, crash one instance by killing the PMON process at the OS level.
4. One of the remaining instances will detect the failure and do instance recovery for the failed instance.
5. Rerun the test 3 or 4 times so that the estimates Oracle uses to determine ESTIMATED_MTTR become more accurate in approximating recovery time.

After the final test, review the alert log for the recovering instance and look for the following lines and their corresponding timestamps to determine the total cache recovery time:

```
Sat Apr 13 00:01:59 2002
Beginning instance recovery of 1 threads

Sat Apr 13 00:04:00 2002
Ended recovery at
```

If the instance recovery time is within a few seconds of or less than the desired MTTR, and I/O load on the system has not exceeded capacity, then no further steps are necessary.

If MTTR is too High

If instance recovery time is more than a few seconds greater than the desired MTTR, then the lower MTTR bound may have been reached. Check the value of TARGET_MTTR in V$INSTANCE_RECOVERY. If Oracle's target MTTR is higher than the desired MTTR, your desired MTTR is too aggressive and Oracle has reset it to a higher value and printed a message similar to this in the alert log:

```
FAST_START_MTTR_TARGET 30 is out of the valid MTTR range, use 71
instead.
```

The lowest MTTR Oracle can maintain is determined by the low limit of the target number of dirty blocks in the cache.

If the instance recovery time is greater than the desired MTTR and the above message does not appear in the alert log, then the system has encountered some bottleneck. Statspack should be used to identify where the bottleneck exists. If there were no bottlenecks without fast-start checkpointing enabled, then it is likely that your I/O capacity has been exceeded as a result of the increased I/O from fast-start checkpointing. In this scenario, either a higher FAST_START_MTTR_TARGET setting must be used, or I/O capacity must be increased.

For a detailed example of calculating the amount of additional I/O introduced by fast-start checkpointing, refer to the section "Calculating Performance Overhead" in Chapter 17 “Configuring Instance Recovery Performance” of *Oracle9i Database Performance Tuning Guide and Reference*.

CONCLUSION

In conclusion, with fast-start checkpointing, there is no throughput degradation provided sufficient I/O capacity exists to absorb the additional writes generated. Fast-start checkpointing provides the benefits of lower instance and crash recovery times and a smoother, more consistent, and more predictable I/O pattern.

Oracle9*i* Fast-Start Checkpointing Best Practices
August 2003
Author: Douglas Utzig, High Availability Systems Group
Contributing Authors: High Availability Systems Group