

Stayin' Alive: High availability without breaking the bank

Second Revision, June 2004
Jeremiah Wilton, ORA-600 Consulting
<http://www.ora-600.net>

ABOUT THE AUTHOR	2
PREFACE TO THE SECOND REVISION	2
INTRODUCTION	2
The planned outage oversight	3
REDUCING AND ELIMINATING PLANNED OUTAGES.....	3
Unnecessary Outages	4
Necessary outages	4
THE LITTLE THINGS	4
Shutdown abort!	4
DDL changes on popular segments	6
Sit 'n spin	6
Online Redefinition	6
Upgrades and Patches: Do them instantly!	6
One-off Patches	7
THE BIG THINGS	8
Moving a database without extended downtime	8
Floating (service-based) IP addresses	8
Moving a database using hot-copy and physical standby switchover	9
LOW-IMPACT REORGANIZATIONS AND CONVERSIONS.....	10
Replicated reorganization.....	10
Smoke and mirrors	11
A note about character sets.....	11
UNPLANNED OUTAGES	11
Monitoring.....	12
Alert file monitoring.....	12
Host monitoring	12
Availability monitoring.....	12
Wait monitoring.....	13
Oracle tablespace monitoring.....	13
Lock monitoring	14
Resource limits	14
maxextents unlimited.....	14
Initialization parameters.....	14
Practical Limits	15
Shared/large pool monitoring: ORA-04031	15
Corrective actions.....	15
Redundancy.....	15
Redundant Storage	16
RAID levels.....	16
SAME.....	16
Precious logfiles	16
Redundant Listeners	17
Cost vs. Rule	17
Filesystem maintenance and cleanup	18
Redo-related pauses	19
Checkpoint currency and recovery.....	20
Recovery from user oopses.....	20
Avoiding reliance on DNS.....	21
TIPS FOR HUMANS.....	21

Remote access / effective communications	21
On-site operators	22
Bridge line	22
Application failure tolerance	22
Dealing with new and unfamiliar problems	22
CONCLUSION	23

ABOUT THE AUTHOR

Jeremiah Wilton was the founding DBA and a 6-year veteran of Amazon.com's database group. A frequent specter at the lecterns of US and overseas Oracle conferences, he enjoys sharing his novel approaches to availability and scalability with the world community of Oracle DBAs. He was a recipient of an honorary Oracle Certified Master certificate in 2000, and a keynote speaker for IOUG-A at Oracle Openworld 2000. Jeremiah now teaches seminars and classes to groups on crisis management and Oracle administration. He is also available for remote emergency DBA services. For more information, visit his website at <http://www.speakeasy.net/~jwilton>.

PREFACE TO THE SECOND REVISION

This is a revised edition of a paper I wrote in 1999 for Oracle Openworld. The industry has seen many changes since the days of Oracle 8.0, and I have tried to integrate those changes into this revision of the paper. Many of my original methods and advice still hold true, but have been embraced and integrated or productized by Oracle, in packages such as DataGuard and RMAN. Overall, the biggest change is that there are simply a lot more ways to achieve high availability now. By including as many of the approaches as I know, I hope that I have made the paper relevant to the widest variety of environments possible.

INTRODUCTION

Oracle's principal corporate direction in recent years shows an overwhelming attentiveness to features that promote availability and scalability. From Real Application Clusters to DataGuard to Flashback Query, much effort has gone into keeping Oracle's customers from suffering debilitating outages and data losses.

Oracle's emphasis on reliability was reflected in their *9i* slogan, "unbreakable." Alongside the huge technical effort, an equally great marketing effort has sought to promote Oracle's reliability. From the customer point of view, this often took the form of a constant drumbeat touting the universal applicability of RAC as a solution to all scalability and availability problems.

In my view, Oracle's RAC-centric marketing sells Oracle short by ignoring a broad spectrum of huge advances in *9i* and *10g*. These features, including logical standby (part of DataGuard), streams, improved multimaster replication, flashback query, Cross-platform transportable tablespaces, improved RMAN (including the pipe interface), constitute the great adaptability for any size organization that is Oracle's greatest strength.

In addition to the diverse features that allow any Oracle customer to engineer the optimal availability-for-cost configuration for their organization, there are hundreds of low-tech techniques and approaches that can contribute as much, if not more, to the overall reliability of a system than any specific availability feature. This paper covers the judicious application of such creatively tailored approaches. They will provide far greater overall improvement in stability per dollar spent than simply implementing RAC and Dataguard as Oracle's marketing seems to suggest.

Globalization and e-commerce have completely altered the IT availability landscape over the past seven years. Beginning in 1997, as the dot coms were making their initial public offerings, with the limelight focused on them in earnest, news sources including the Wall Street Journal and C|net made conspicuous – and gleeful – note of extended website outages, including a couple multi-hour failures by Amazon.com and a spectacular two-day Ebay outage.

Prior to the dot com boom, many financial institutions and other large Oracle clients were not true 24x7 systems, and allowed weekend and nighttime maintenance windows. They did not concern themselves greatly with the impact of uptime-reducing activities such as upgrades and migrations. Those few organizations that needed true high availability had to do a great deal of engineering on their own. Some went as far as implementing standby databases using AIJ files as early as v.5. The downside of these efforts was a lack of good support from Oracle and few peers to consult in the user community.

Although Oracle Parallel Server (OPS), the precursor of RAC, was available on Oracle v.7, this platform was difficult and impractical to deploy in any environment with a diverse workload. Each node could only share cached blocks with another node by writing those blocks down to disk, where the other node could take possession of them. The whole process meant that it was extremely resource-intensive to run workloads that required the same data on different nodes at the same time. This limitation begged the question, “If we have to separate the workloads anyway, why not use two separate databases?”

Replication was in its infancy in the mid-nineties, and did not attain stable bug-free maturity until version 8. All in all, Oracle was weak on high-availability until the dot-coms and the Internet boom forced them to change direction and focus on it. The rest of the industry can thank the dot coms for precipitating the change of direction resulted ultimately in Streams, logical standby, flashback query, advanced queues, cross-platform transportable tablespaces, solid asynchronous multimaster replication, and RAC.

The planned outage oversight

Because many IT professionals fail to consider planned downtime as outages, there is a tendency to focus on prevention of unplanned outages without sufficiently seeking to reduce the length and number of planned downtime. Perhaps because of the overall focus on unplanned outages, companies seldom regard planned maintenance as an availability problem, and therefore seldom actively seek to reduce or eliminate them. Such attitudes must fall by the wayside in the arena of global and Internet commerce, as managers and staff realize that all downtime, planned or unplanned, detrimentally affects the bottom line.

I often encounter DBAs and SAs that claim to run 24x7 systems, but still perform cold backups or other unnecessary tasks that make the system unavailable. The common notion that planned outages, and “maintenance windows” somehow are exempt or don’t count as downtime, is an idea that needs to be banished from our field. While it is true that you can schedule such tasks during periods of lowest usage, the mere act of shutting the instance down and starting it back up can be destabilizing due to human error (typo in the initialization file), O/S peculiarity (unremovable shared memory segments), or human error (open database file deleted by mistake, but held open by running instance until shutdown). In addition, many companies that tolerate frequent planned outages may find that business processes could be made more efficient by moving some batch jobs or other work into the timeslot that a scheduled outage window used to occupy.

REDUCING AND ELIMINATING PLANNED OUTAGES

Unnecessary Outages

Some outages are unnecessary. The king of unnecessary outages is the cold backup. With minimal engineering, and the tiniest bit of effort, any Oracle database can be backed up on line. The prerequisite to most online backups, such as tablespace hot backup and RMAN, is archivelog mode. It can be safely stated that anyone running in archivelog mode has no valid reason to be taking cold backups. Hot backups achieve the same result, but avoid the destabilization of shutting down and starting up the instance(s). Many scripts for performing online backups on a variety of platforms are available in the public domain.

Another maddeningly common planned outage is the scheduled reboot. Believe it or not, a huge number of companies restart their Oracle instances, or even reboot their server hosts on a periodic basis. Server hosts, regardless of the platform, are designed to run for months at a time without rebooting. Any problem that necessitates frequent rebooting should be thoroughly investigated for the root cause, and eliminated through the vendor's support channels. Any such routine rebooting should be done only as a temporary measure until the vendor has provided a patch or upgrade that addresses the specific technical problem that is forcing periodic rebooting.

Necessary outages

Other planned outages, such as migrations and reorganizations, are not altogether unnecessary. But any planned outage can be dramatically reduced in duration and impact through the use of novel approaches, and careful scripting and management. No effort should be wasted on planned necessary outages, because **they are the outages over which DBAs can have the most control**. The mantra for planned outages should be "why waste time when seconds count?"

THE LITTLE THINGS

Few major architectural configurations will have as much impact on a system's availability as just maximizing the efficiency of the small tasks that contribute to the timeline of an outage. Any outage that is worked manually could be scripted to some degree, saving the time spent fumbling and typing. Beyond scripting, a variety of low-tech techniques can drop hours of annual downtime off your availability numbers. A selection of these techniques follows.

Shutdown abort!

As I have mentioned, all efforts should be made to avoid shutting a service down. With each new Oracle version, fewer and fewer configuration changes require a restart to take effect. Many initialization parameters can be changed in a running instance. If you think you must restart, make sure you really have to.

But if you must shut down, how do you do it? Many DBAs issue a shutdown immediate, then wait. Shutdown immediate kicks users off and locks out new logins as soon as you issue it, but often takes a long time to finish in a large system. Usually, DBAs of large systems eventually resort to a shutdown abort. If this happens every time, why not just shutdown abort the first time?

The answer I consistently hear is that it is dangerous. This is certainly not so. Simply put, the best way to shut down an Oracle database is like this:

```
SQL> alter system checkpoint;
SQL> shutdown abort
```

In the few situations such as cold backups and version upgrades where consistent datafiles and controlfiles are required, you can do this:

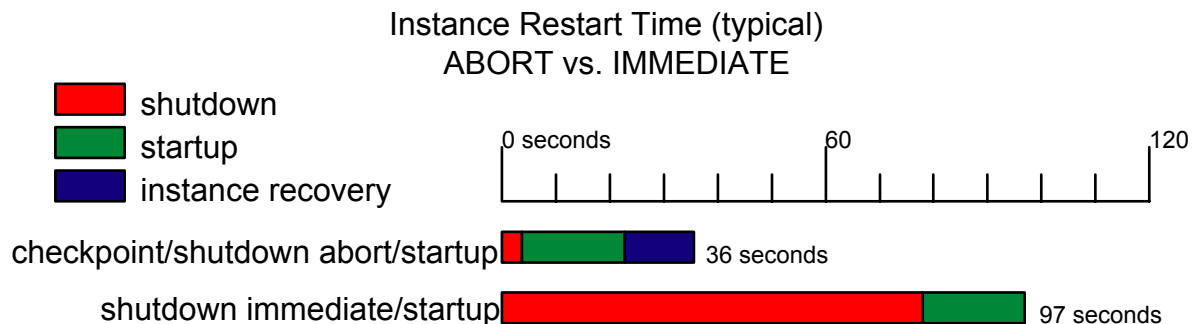
```
SQL> alter system checkpoint;
SQL> shutdown abort
$ !snrctl stop
SQL> startup restrict
SQL> shutdown immediate
```

Shutting down the listener prevents CPU resources from being used by processes forking while crash recovery is completing. This speeds startup time in some environments.

A common misconception holds that shutdown abort is somehow more dangerous or reckless than the other shutdown modes, and can result in data being lost. This is not the case. True, shutdown abort kills the Oracle background and shadow processes, and removes the shared memory segments. This may seem ugly, but esthetics matter little when it comes to availability. The fact remains that **all committed transactions are intact in the online redologs**, even if the data associated with them has not been written to the datafiles. On startup, crash recovery applies online redologs beginning from the time of the most recent checkpoint. When was the most recent checkpoint? How about one second before you shut down?

When recovery is complete, the database is opened for use. Transaction rollback and temp segment cleanup (the two big time wasters for SHUTDOWN IMMEDIATE) take place in the background after startup, when applications can already log in and do work. No user's time is wasted waiting for all uncommitted transactions to roll back.

Consider the following diagram comparing total outage times using ABORT vs. IMMEDIATE shutdown on a typical instance:



When starting up after a shutdown abort, the amount of time spent in instance recovery depends largely upon how recently the last checkpoint was issued. By forcing a checkpoint immediately prior to issuing shutdown abort, the redo required to complete crash recovery and bring the database open will be minimal. The alternative in an active environment to shutdown abort is shutdown immediate, but immediate shutdowns often take too long, rolling back transactions and cleaning up temp segments while precious seconds pass by.

Shutdown abort is useful for very brief downtimes, such as those required to change a non-dynamic initialization parameter. In practice on Oracle instances with very large SGAs, such quick bounces can typically take as little as 30 to 40 seconds. In order to expedite planned shutdowns and startups, the same scripts that are devised for reliable and fast startups at machine boot and shutdown should be used for manual shutdowns and startups. The scripts should be used because they can issue commands faster and more accurately than you can type, and can be designed to resolve potential complications such as datafiles left in backup mode.

DDL changes on popular segments

Another commonly cited reason for having to restart is to perform DDL on heavily-accessed objects. When trying to apply a DDL change to some segments, a DBA may encounter the error ORA-00054: resource busy and acquire with NOWAIT specified. This can occur even if the DBA acquires an exclusive DML lock on the table using the lock table command. Unfortunately, locking a table does not guarantee the success of any subsequent DDL statement on the table or associated indexes. DDL statements must obtain a library cache lock in order to perform DDL, and there is no manual mechanism to guarantee possession of a library cache lock.

A common reaction is to restart into RESTRICT mode, perform the DDL, then reopen the database to applications. However, in most cases it should not be necessary to resort to these measures.

Sit 'n spin

To get a DDL change to apply on such an active segment, write a script that loops trying to execute DDL, stopping only when it finally succeeds. In the unlikely event this tactic continues to fail, one might try disabling selected applications that heavily access the object in question for long enough to perform the DDL, thus leaving the database up and available to other applications.

Jonathan Lewis (<http://www.jlcomp.demon.co.uk>) wrote a simple stored procedure to do this, and it is included here with his permission:

```
create or replace procedure do_ddl(m_sql varchar2) as
  in_use exception ;
  pragma exception_init(in_use, -54);
begin
  while true loop
    begin
      execute immediate m_sql;
      exit;
    exception
      when in_use then null;
    end;
    dbms_lock.sleep(0.01);
  end loop;
end;
```

Online Redefinition

Online redefinition is also possible in Oracle 9i and 10g using the dbms_redefinition package. This package creates an interim table to take DML and store data while the original table is being redefined. It is useful for a variety of activities that cannot be accomplished with a single DDL statement, such as moving or reorganizing a segment. For details on using this package, consult Chapter 14 of the *Oracle Database Administrator's Guide 10g Rel. 1*. For most DDL, however, the PL/SQL spinning method is sufficient and avoids the unnecessary complexity of dbms_redefinition.

Upgrades and Patches: Do them instantly!

Oracle patches and upgrades are another huge waste of downtime. Incredibly, many DBAs wait until after they have shut down an instance to get the upgraded or patched software ready. In almost every case, it should be possible to have upgraded software installed and ready before you shut anything down.

- **Upgrades and patchsets** are similar in that they are installed using the Oracle Universal Installer (OUI).
- **One-off patches** are applied with the opatch utility (or patch.sh prior to 9i), which is far less cumbersome than the OUI.

One-off Patches

One-off patches are typically modifications to a small piece of Oracle's code to address a specific bug. On Unix operating systems, they usually consist of a new version of a single .o file that the Opatch utility places into a .a file (usually libserver<version>.a for the database) in place of an old version using the unix ar command. Afterwards, Opatch relinks Oracle using the Unix make command.

Many DBAs have had the unpleasant experience of taking an instance down to patch it, only to discover that there is something wrong with the patch, or that Oracle will not relink. Many unnecessary minutes of downtime have been racked up while the DBA searches around for the path to the 'make' command. Fortunately, it is not actually necessary to have the instance down while building a patched executable. Patch application really only requires a few seconds of downtime.

By making small modifications to the patch's configuration file, you can make it call an alternate makefile of your own to relink Oracle. By changing the name of the Oracle (or any other) binary in the alternate makefile, you can stage a new version of Oracle that contains your patch, without ever taking down the instance.

Preparing to relink Oracle on Unix without shutting down

- Find the Oracle makefile, \$ORACLE_HOME/rdbms/lib/ins_rdbms.mk.
- Make a copy of the makefile with an alternate name, such as ins_rdbms_new.mk
- Edit the file, and find the makefile target that corresponds to the executable you want to relink. For the Oracle executable, the target is ioracle.
- Change the makefile for the target so that it creates a differently named executable from the one that is running. The following example is for the Oracle executable. Changes are underlined:

```
ioracle: $(ORACLE)
-mv -f $(ORACLE_HOME)/bin/oracle_new $(ORACLE_HOME)/bin/oracle_newO
-mv oracle $(ORACLE_HOME)/bin/oracle_new
-chmod 6751 $(ORACLE_HOME)/bin/oracle_new
```

Through 8i: Modifying patch.sh

Before 9i, one-off patches for Unix came with a patch.sh script that installs new modules into library files, and then calls make to relink. To make patch.sh ignore the running instance and call the modified makefile, make changes to the setup() section at the top of the file to ignore the running instance and call the alternate makefile:

```
SHUTDOWN=false
MAKE_TRIPLETS="rdbms/lib:ins_rdbms_new.mk:ioracle"
```

Now running patch.sh will create a new patched Oracle executable called \$ORACLE_HOME/bin/oracle_new.

9i and up: Modifying etc/config/actions and etc/config/inventory

For Oracle 9i and higher, one-off patches on Unix come with two files under etc/config/:

Change the **actions** file to call your alternate makefile:

```
<make change_dir="%ORACLE_HOME%/rdbms/lib" make_file="ins_rdbms_new.mk" make_target="ioracle"/>
```

Change the **inventory** file to ignore the running instance:

```
<instance_shutdown>false</instance_shutdown>
```

Now running opatch apply will create a new patched Oracle executable called \$ORACLE_HOME/bin/oracle_new.

The 30-second patch outage

With the successfully patched executable in place, a fast shutdown and restart are all that is required to apply the patch:

```
SQL> alter system checkpoint;
SQL> shutdown abort
$ mv -f $ORACLE_HOME/bin/oracle_new $ORACLE_HOME/bin/oracle
SQL> startup
```

THE BIG THINGS

While many activities require either no or little downtime to perform, some do require major outages. For those, you can make the difference between a 2-hour and a 5 hour outage by planning and scripting carefully ahead of time. A selection of efficient approaches to major planned outages follows.

Moving a database without extended downtime

From time to time, especially in a rapidly growing or changing organization, the need arises to move a critical service from one server to another, either locally or over great distances. For those who do not wish to incur extended outages for such operations, there are number of strategies. Depending on the approach, micro-downtime strategies can be employed to move a database, change database block size, or split the contents of one database into several other databases. The following techniques can be used together, or individually to make planned maintenance shorter.

Floating (service-based) IP addresses

Oracle databases can be moved from one host to another more easily if, a special IP address, other than the host IP address, is established for each Oracle service. That IP address can then be moved from one host to another, allowing a diverse application base to find the newly moved service. All Unix platforms on which Oracle is supported, and also windows, support the configuration of multiple IP addresses on an network interface. On many Unices, this can be accomplished with the ifconfig tool:

First, after any Oracle listeners are stopped, the IP address is removed on the original host:

```
root# ifconfig eth0 del 192.168.1.110
```

Then, after the database is made available on the other host, either by standby failover, or restore from backup, or any other method, the IP address is added to an interface on the destination host.

```
root# ifconfig eth0 add 192.168.1.110
```

On 32-bit Windows, IP addresses may be similarly added and deleted via the advanced settings in the networking control panel. Good luck scripting this.

If you tried to move a service without moving a service-based IP address, you would have to update all client tnsnames.ora files manually wherever they exist to reflect the new machine's IP address. A less cumbersome change could be made using Oracle Names Server, but that is not guaranteed to cover all clients who may have set up a connection to the server unbeknownst to Names Server. For instance, programs that use the JDBC thin client have host and port hardcoded.

Service-based IP addresses are the native failover redirection method included in the vendor scripts for hardware cluster products such as HP MC Serviceguard, SunCluster, and Compaq TruCluster, where one node must rapidly assume the services of another. The technique is equally applicable to a variety of planned maintenance situations involving failover from one host to another. By including this method in standby failover scripts, the time spent fiddling around during a failover can be reduced.

Moving a database using hot-copy and physical standby switchover

A database can be moved to a new host almost instantaneously and with minimal interruption of service using physical standby switchover.

It is important to understand the difference between a physical standby failover and a physical standby switchover. In a switchover, the standby neatly picks up where the primary left off in the redolog sequence. No resetlogs is performed when opening the standby, so if the primary is shut down at SCN 12345, the standby picks up at SCN 12346. Prior backups remain valid, because there is an unbroken redolog stream that can still be applied to them.

In a failover, something prevents the DBA from obtaining all the logs from the primary before opening the standby. In this scenario, **actual transactions** that were successfully committed on the primary **are lost forever**. This procedure is never used as part of a planned outage, because it loses transactions and invalidates prior backups by performing a resetlogs when the standby is opened.

The first step to performing a switchover is to establish a standby on a secondary on the target host. Copy the database there while open and operating using tablespace backup mode or RMAN. As of Oracle 10g, RMAN can perform copies directly to a secondary host using the "RMAN Pipe Interface." Roll forward the database copy in standby mode, applying redologs. When the predetermined outage time arrives, shut both the old and new recovering databases down, and perform a graceful standby failover to bring the service up on a new host, using the following steps:

- Shut down the primary
- Copy the last few archived redologs from the primary to the standby
- Begin copying the online redologs from the primary to the standby
- Meanwhile, roll forward the standby until it has consumed all the archived redologs
- Copy the controlfiles from the primary to the standby
- Disable the primary by renaming the controlfiles
- Start up the standby using 'startup mount'
- Issue a 'recover database' command and apply the last few logs
- Open the database using 'alter database open'
- Move the service-based IP address from the primary to the standby
- Start the listeners on the standby, and the service is now running on the target host

This same procedure can be accomplished using DataGuard role management services, using the following steps:

- Shut down the listeners on the primary
- Make the primary into a standby. On the primary:
SQL> alter database commit switchover to physical standby with session shutdown;
SQL> alter system checkpoint;
SQL> shutdown abort
SQL> startup mount
- Make the standby into a primary.
SQL> alter database commit switchover to primary;
SQL> shutdown abort
SQL> startup

- Move the service-based IP address from the primary to the standby
- Reconfigure DataGuard's log apply services so that the new primary's logs are shipped to the new standby.

If you plan to move the service back to the original host, you can replace the controlfiles on the old primary with standby controlfiles and keep it recovering the archived redologs from the newly opened service.

Many users of the standby feature do not realize how useful it can be for planned outages. The graceful failover method, where the database is opened *noresetlogs*, allows the former primary to be immediately pressed into service as a standby. This allows easy maintenance on one set of hardware (host and disks), while the database runs on the secondary host. Once maintenance is completed, and the database on the primary server (now running as a standby) is caught up, the failover can be performed again, in the other direction, so that the database is once again running on the original server, and the standby on the secondary. This is often referred to as "role reversal" or "graceful switchover and switchback."

To perform a graceful failover to a running standby, you can write your own scripts, or use DataGuard. Using my own scripts, I have performed very rapid failovers of one or two minutes total unavailable time for the service. Since some of the activities can be performed simultaneously and in parallel (such as copying the online logs while still recovering the standby), the total time to perform the failover can be kept very short.

When the service has been moved, it is important to disable the database on the old host by moving or renaming the initialization file or the controlfiles so that nobody accidentally starts it up, causing it to diverge from the carefully preserved redolog stream. DataGuard achieves implicitly this by converting the controlfile to a standby controlfile.

LOW-IMPACT REORGANIZATIONS AND CONVERSIONS

Some types of migrations between hosts of the same platform require changes to the physical datablocks comprising the database. Such reorganizations are not candidates for physical standby switchover, because a physical standby must be block-for-block identical to the primary database. Some of the activities that fall under this category are:

- Block size changes
- Character set conversions

Replicated reorganization

There is a novel low-impact technique that can be used to make major conversions with minimal interruption. The technique uses multimaster replication or streams.

- First, establish a physical standby database of the source database on the target host, and get it rolling forward and caught up with the primary
- During a brief planned outage, open the standby and establish multimaster replication between the tables on the source database and target database.
- As long as applications are not allowed to log into the target database, it can be shut down and reorganized using export/import. In the mean time, replication changes bound for the target database are queued up on the source.
- Once reorganized, allow the target database to accept all the queued replication changes until the databases are once again logically in sync.

- Point the users and applications at the target database, and remove replication during a second brief downtime.

Smoke and mirrors

As of Oracle 9i, multiple block sizes and multiple character sets are permitted in a single database. One technique for block size or character set conversion even simpler than above is to gradually move segments from tables of one block size or character set to tables of another block size or character set using 'alter table ... move' and 'alter index ... rebuild'. This can be done in a series of brief low-impact outages to one table or index at a time. If a very large table is to be converted, consider the following smoke-and-mirrors technique:

- Create an empty table with the new character set or block size
- Create a view that is defined as a UNION ALL of the source and target tables
- Create a synonym that points applications at the view instead of the source table
- Write a PL/SQL, Java or OCI program that atomically moves rows from the source table to the target table in discrete transactions using row locking
- Upon completion, point the synonym at the target table, and drop the view and source tables

This procedure is largely incorporated into the Online Redefinition feature, available in 9i and 10g. For details on using this package, consult Chapter 14 of the *Oracle Database Administrator's Guide 10g Rel. 1*.

A note about character sets

Many character sets are actually supersets or subsets of others. For instance, Latin 1 (WE8ISO8859P1) is a superset of US7ASCII. In such cases, no actual block changes are needed to convert from a subset character set to a superset. An old method that was popular before Oracle addressed this was to simply update the SYS.PROPS\$ table and restart. This worked in many cases because the ASCII values were the same under the new character set. From 9i, it is possible to 'alter database character set...;' For more information on the gotchas of character set conversion, see chapter 11 of the *Oracle Database Globalization Support Guide 10g Rel. 1*.

UNPLANNED OUTAGES

Unplanned outages are the sexiest area of availability, understandable not only to all DBAs, but also to managers and executives. Because such failures are inherently unpredictable, DBAs often feel the need to be prepared not only for the most probable and easily preventable failures, such as disk faults and instance crashes, but also for large-scale natural calamities. The challenge in planning such strategies is walking the thin line between prudence and overzealousness.

When discussed in conference rooms, the topic of availability is fraught with scenarios of calamity and apocalypse of biblical proportions. While I by no means seek to diminish the likelihood of Armageddon, DBAs and corporate executives alike must seriously consider that a disk failure or even a bad motherboard is statistically a great deal more likely, by several orders of magnitude. Complex solutions that hold the promise of near-100% infallibility must not be pursued at the expense of common-sense measures that provide basic protection against the most common failures. While great ships may be unsinkable, common sense reminds us not to underestimate the necessity of lifeboats. Or lifejackets. Or a paddle.

Monitoring

The most basic measure you can take to maintain high availability is to monitor your systems. To this end, many companies have produced software for monitoring logs, disk space, CPU, memory utilization and database statistics. Some sites have built their own monitoring mechanisms in Perl or using shell scripts. When considering whether to buy or build, be sure to take into account the extensive time and effort required to customize any off-the-shelf package to meet the business requirements of your system.

At a minimum, monitoring scripts must have the ability to notify staff of critical events, so that appropriate measures can be taken to avert an outage or rectify a critical problem.

Alert file monitoring

Many critical events, such as block corruption, memory and space problems can be detected by monitoring the alert file. A good monitor will allow a user-definable list of normal messages (beginning checkpoint, checkpoint complete, alter tablespace) to be excluded from the notification system, leaving only unusual messages to be reported to the DBA. Significant downtime can be shaved off an outage if the DBA receives a page such as "Instance terminated by PMON," rather than waiting for users to complain, then logging in to investigate.

Additionally, it is helpful if low-priority messages can also be differentiated from critical ones, so that a single ORA-7445 or ORA-600 in a shadow process does not wake the support staff at 3:00AM. Such messages should instead be reported to an email address where the issue can be investigated at lower priority at a later time.

Host monitoring

It is typically the responsibility of a systems administrator to monitor critical aspects of system performance, including CPU, disk and memory utilization, as well as the system messages file. Such monitoring, however, is absolutely critical to the successful continued operation of an Oracle database. So if such monitoring does not exist, it is manifestly in the interest of the DBA to make sure that it is established.

The LOG_ARCHIVE_DEST directory, for instance, must never fill to 100% capacity, or the archiver will suspend operations on the Oracle instance once all online redologs have been filled but remain unarchived. Similarly, if listener logging is enabled, listeners will cease to accept connections if the logging directory is full. For this reason, a DBA must either work very closely with the systems administrators, or monitor critical systems resources independently.

Availability monitoring

Probably the single most important monitor is one that will determine if a system is unavailable. Many DBAs write this monitor to look for background processes, such as PMON and SMON, and to alarm if they are missing. Unfortunately this method does not determine availability. True, you can be reasonably sure that without PMON the service is or will very soon be unavailable. But the presence of PMON does nothing to guarantee that people can access a service.

A better method for guaranteeing availability is to write a program that runs on an application server where a typical user application runs. The program should use the same database API that a typical application uses, and should attempt to connect to a service and fetch a row of data. If an availability monitor can't do that, it isn't really measuring availability.

Availability monitors are important record keepers. If you build one, you should have it record availability over in a file or a database. This allows accurate record to be kept of a system's availability over time, and provides a metric for the DBA and management to determine if current availability strategies are adequate or not.

Wait monitoring

Along the same lines of ‘the proof is in the pudding,’ comes the practice of monitoring for excessive waits in an instance. If too many sessions are waiting, then the service isn’t truly available to those sessions stuck waiting. It is technically an outage (however brief) for those users. An effective wait monitor will alarm when the percentage of sessions waiting on nontrivial wait events exceed a threshold percentage. Trivial events that can be ignored include ‘rdbms ipc message’ and ‘SQL*Net message from client.’ This type of monitoring is the next step up from availability monitoring as discussed above.

Frequent monitoring of system statistics and wait events allow DBAs to identify the cause of slowness problems and also allow accurate projections to be made of resource utilization based on historical trends. The critical views to monitor and record are as follows:

- v\$system_event for all-time greatest wait events on the instance
- v\$sysstat for all-time greatest resource utilization on the instance
- v\$session_event for each logged-in session’s greatest wait events since login
- v\$session_wait for each logged-in session’s current wait event
- v\$sesstat for each logged-in session’s greatest resource utilization since login

Statspack does a good job of recording these and other tables over time, and presenting trends and data from the historical information. In Oracle 10g, Active Session History (ASH) provides a superior fine-grained mechanism for recording wait and resource information on a per-session level over time.

By recording the v\$session_wait session event information, outages and general slowness problems can be easily diagnosed and prevented, even if a DBA did not actually log in and witness the problem at the time it happened. By going back through logs of v\$session_wait output, the exact event that a particular session was waiting on at a given time can be determined. The v\$session_wait view provides further information that a good monitoring script would be able to decode.

- If v\$session_wait indicates that a session is waiting on an enqueue, then the monitoring script should also look and record the blockers and waiters.
- If a session is seen waiting on “latch free,” then the monitor should record the name of the latch from v\$latch, and the session holding the latch at the time identified from v\$latchholder.
- If a session is performing a “db file sequential read,” then the monitor should record the table or index being scanned from the p1 and p2 columns in v\$session_wait and joining to dba_extents.

On a database-wide basis, statistics and events should be monitored and recorded on a periodic basis using Statspack or your own scripts. Using this information, generally increased resource utilization can be narrowed down to the time it started and correlated with the specific changes or applications that might have triggered a problem. Also, gradual increases in resource utilization can be tracked to accurately determine when the current system will run out of capacity.

Oracle tablespace monitoring

It is unacceptable for users of a high-availability site to encounter “ORA-01653: unable to extend table...” or an “ORA-01654: unable to extend index...” Effective monitoring should be able to determine if the NEXT extent of any segment in a tablespace is bigger than the largest available piece of free space in that tablespace, and raise an alarm if it is. Similarly, any table close to reaching MAX_EXTENTS should be noticed and the appropriate people notified. This kind of monitoring is fundamental to the continued smooth operation of a mission-critical system.

The most advanced monitoring of this type will maintain a daily log of the growth of individual segments, and project growth into the future. In this way, effective monitoring will notify the DBA well in advance of space being exhausted. It is much easier to understand the impact of 5 days-worth of space remaining in a tablespace than 512Mb of space.

Lock monitoring

Often, complaints of slowness in the database are traced back to users contending for the same rows in tables, when one user is holding a row lock for too long and blocking others. Preventing locking problems in the first place relies upon good application design. A well designed OLTP application should not hold row locks for more than a few seconds. But even the best applications, using optimistic locking approaches, can exit abnormally while holding a lock, and PMON may take some time to clean it up.

Monitoring can be implemented to detect blocking locks and to automatically take a series of steps to resolve the problem. One good approach is time-based escalation. With time-based lock notification escalation, a blocking lock is initially detected by a monitoring script, and notification sent to the blocking user, either by email or terminal warning. Unfortunately, some end users do not use terminals that can receive server-initiated messages (such as a web browser), may not be reading their email, or may in fact not be people but rather batch processes. If a user does not release their lock, then the notification level escalates, and a page is sent to a DBA. In extremely critical applications, or in applications that require lights-out operation, the blocking session can simply be terminated if it blocks for too long, allowing waiting sessions to proceed.

Resource limits

Certain limits do not make sense in a high-availability environment. Limitations on extents, sessions, processes and similar parameters should be set at theoretical maximum system capacity levels. In many cases, setting artificial limits can result in degradation of service that has no basis in true system capacity.

A good way to be sure that availability problems will not be caused by artificially low settings on initialization parameters is to comb thoroughly through `v$resource_limit`, seeking parameters that are near their limits. Monitoring should be deployed to check `v$resource_limit` periodically, and to notify if any resources approach their limit.

maxextents unlimited

If good monitoring practices are maintained, such as those outlined above, it does not make sense to set `MAXEXTENTS` on tables in application schemas to anything other than `UNLIMITED`. Even if a DBA wants to keep extents below a certain number, it makes more sense to monitor extent allocations, and take corrective actions when it is least disruptive. To cause degradation or loss of service as a result of reaching `MAXEXTENTS` during peak usage periods would be a senseless outage. It is useful to note that tables and indexes can be composed of many thousands of extents without any significant performance or manageability problems. The classic problem of extent deallocation on drop or truncate of segments has been almost entirely alleviated through the use of locally-managed tablespaces.

Rollback and temporary segments, on the contrary, should **not** be set to grow with `MAXEXTENTS UNLIMITED`. Unbounded growth of a single rollback or temporary segment can cause denial of service for other sessions, and benefits only the session causing the growth. On high-throughput OLTP systems, no single session should ever need massive amounts of temporary or rollback space. Transactions on such systems should be small and commit quickly, and sorting should be tuned to a minimum for best application performance.

Initialization parameters

Several initialization parameters can impose needless limits on availability. Because the default values of many parameters are derived from the specified values of other parameters, some limits can exist without being expressly specified in the initialization file. A good example is the `SESSIONS` parameter, which has a default value of $(1.1 \times \text{PROCESSES} + 5)$. This derived value makes sense as long as sessions grow at roughly the same rate as processes. However, at a site using Shared Server (formerly MTS), sessions can

grow at a much greater rate than processes, because Shared Server consolidates processes. When using Shared Server as a scalability measure, care must be taken to account for the large number of sessions, transactions, DML locks and other resources that can be supported on a relatively small number of processes.

Practical Limits

Some resource limits are not covered by `v$resource_limit`, which only includes initialization parameters whose values might be exceeded. There are numerous other practical limits that can be exceeded, and should be individually monitored. Some of these include:

- Queue depths for AQ
- Multimaster replication queue depths/lag
- Streams queue depths
- Materialized view log depth for fast refresh MVs
- Limits for Shared Server such as dispatchers, shared servers and circuits.

A good example of this is the shared pool:

Shared/large pool monitoring: ORA-04031

By monitoring `v$sgastat`, a DBA can make a simple determination ahead of time if an instance is going to run out of shared or large pool space. In systems with diverse and heavy workloads, the shared pool can easily become “fragmented.” In this situation, there appears to be sufficient free space according to `v$sgastat`, but when you look at `x$ksmsp`, the chunks of free space that are available are too small to use. A good shared pool monitor will not only monitor percent free space, but will also compare the sizes of common objects in the shared pool with the number and size of free space chunks.

Corrective actions

Many monitoring scripts can be designed to take corrective actions on their own, rather than immediately paging a DBA.

- Scripts monitoring the job queue can try restarting a job
- Jobs that detect blocking locks held by long-idle sessions can terminate the session
- Upon finding that the shared pool is badly fragmented, selected objects can be unpinned, and the shared pool flushed

Oracle’s user profile feature allows SMON to perform many resource limiting actions along certain parameters, but more fine-grained control can be exercised by writing simple scripts to do such tasks.

Redundancy

Redundancy is one of management’s favorite buzz words when high availability is discussed. It is a word that brings to mind very expensive solutions, such as complete remote replicas of all critical systems, triple disk mirrors, and redundant diesel generators. Such solutions may help guarantee availability for mission-critical systems in the direst of circumstances, but may be beyond the means of the average organization, or even many established companies.

For the greatest benefit for the least capital expenditure, redundancy may be applied not only to whole installations, such as hosts and datacenters, but also to small pieces of architecture, such as redologs and listeners. Many individual components in the Oracle architecture can be made redundant within the scope of

a single host. At little or no additional expense, greater availability and peace of mind can be achieved with little effort.

Redundant Storage

One of the most basic measures you can take to improve availability is to invest in fault tolerant storage. This means disks should be mirrored, or protected with a fault-tolerant RAID level. Disk controllers can be made redundant as well, and disk arrays can be split across multiple controllers. Files that Oracle mirrors such as controlfiles and redologs can be mirrored on different controllers. Although volume management software products offer the capability to perform software striping and mirroring of storage, it is best to use disk controllers for this purpose. Disk controllers operate as independent CPUs, and can perform mirror resyncs and other I/O and CPU intensive operations without consuming host resources, as volume manager mirroring does.

RAID levels

Different RAID levels provide different levels of fault tolerance and speed. Choosing a RAID level is dependent on the type of access patterns and the availability requirements of the system.

RAID Level	RAID0	RAID1	RAID0+1	RAID3	RAID5
Description	Data is striped across a set of disks	Data is mirrored among a set of disks	Data is striped and mirrored	Data is striped across a set of disks, with one disk reserved for parity information for data recovery	Data and parity information for data recovery are striped across a set of disks
Advantages	Access is distributed across many disks, reducing contention	Data is duplicated in case of a disk failure	Data is duplicated in case of disk failure and evenly distributed for improved access speed	Fewer disks required for data distribution and fault tolerance	Fewest disks required for data distribution and fault tolerance
Disadvantages	No fault tolerance	Data is not evenly distributed; writing to two drives takes extra time	Data must be written to two drives	Parity drive is a bottleneck; many I/O operations are necessary for each write	Many I/O operations are necessary for each write
Speed	Fast read Fast write	Fast read Slower write	Fast read Normal write	Fast read Slow write	Fast read Slow write

SAME

Oracle recommends that customers “stripe and mirror everything” (SAME). This recommendation that everyone use RAID0+1 is fine for people who don’t want to think at all about their specific needs, but others may choose to put more thought into their storage configuration. For example, some instances may have a low rate of writes. Their best RAID level might be RAID5, since they can save money on equipment and space.

Precious logfiles

Other customers might have an extremely high write rate to their online redologs, or don’t want online logs to become a bottleneck during periods of heavy transactional activity. Because redologs are sequentially written, those customers might prefer to assign dedicated physical disks to redologs, and to alternate such

that the archiver reads from a disk other than the ones containing the current active logfiles. If log multiplexing is to be used, then the disk mirroring would be superfluous. Sustained heavy write activity on redologs can also render the RAID's write cache superfluous for those writes.

There is some disagreement over whether it is necessary or advisable to mirror logs at both the hardware and Oracle levels. Complicating this question is the aforementioned common practice of placing redologs on dedicated disks. A level of fault tolerance higher than just storing logs on a RAID 0+1 array can be achieved by using Oracle log multiplexing to maintain logfile members on two or more separate disk subsystems with independent controllers. Consider that although redundant disk controllers can be used to maintain a single mirror set, any controller-based corruptions that occur in such a configuration will be applied to all copies of a log. If a separate controller is responsible for each copy of a logfile, then one controller's corruptions can only damage one copy of any particular log, and never both. Also, by using multiplexed redologs, a system can be protected against many filesystem or volume manager-based faults. This desire not to rely on a single disk array for all redologs drives many organizations in the direction of multiplexing over hardware mirroring for logs.

Using dedicated redo drives also assures that heavily accessed database blocks will not be accidentally placed on the same disks as redologs, causing I/O contention.

Redundant Listeners

Redundancy at the Net8 listener level is another critical element for high availability. Although the days of Oracle listeners hanging are largely gone, you can still use multiple listeners to achieve redundancy across network interfaces, or to allow shutdown of one listener at a time for maintenance or troubleshooting.

To achieve redundancy with two or more listeners, configure client tnsnames.ora entries with the failover=on parameter and multiple addresses. When a client is establishing a connection, it will try the first address, then the next and so on. If one listener fails, connection attempts will automatically try to use the other listener. In this way, one listener at a time can even be intentionally stopped and restarted without any effect on client connections.

When running in a multiple listener configuration, connections can be randomized among the listeners by specifying the load_balance=on parameter in the client tnsnames.ora. This allows you to realize the scaling benefits of using both listeners.

Address List example with failover and load balancing:

```
orcl.world=
  (description=
    (failover=on)
    (load_balance=on)
    (address=(protocol=tcp)(host=207.60.86.67)(port=1521))
    (address=(protocol=tcp)(host=207.60.86.67)(port=1522))
  )
  (connect_data=(sid=orcl))
```

Cost vs. Rule

Most DBAs think of the choice of optimizers as a tuning issue, not a high-availability issue. But ask anyone who has run in a large environment with the CBO, and they will tell you that they have had more than their share of downtime related to the CBO. Even if the CBO can't crash the database, it can render unusable a critical query performed by a critical application. In many cases, this means that the service is technically down until the query is fixed.x

A popular approach has been to ignore the CBO and use rule exclusively. Oracle Applications were dependent solely on the RBO through 11*i*. Soon, however, there will be no choice between cost and rule when it comes to choosing an optimizer mode. Oracle's stated direction for future versions is to gradually phase out use of the rule-based optimizer. This is good news, because it means they will necessarily have to improve the stability of the CBO.

Because of the infamous instability of the CBO, it goes against the instinct of many DBAs that overall greater stability could be achieved through its consistent and standardized use. After all, using rule guarantees that query plans will not change wildly from one day to the next for the same SQL statement, which is a hallmark of the CBO.

The advantage of the CBO is that it is dynamic and adaptable. With the CBO, if tables are consistently analyzed on a regular basis, and histograms maintained on skewed columns, then rapid changes in data size or distribution of a particular table can be handled with the most efficient query plan for that data. With the RBO, changes in data distribution in a table will result in no change in query plan, so no additional efficiencies will be introduced to handle such changes.

Oracle's improvement of the CBO since Oracle v.7 has been a slow process, but with 10*g* it is reaching some kind of maturity. Increasingly, the task of analyzing tables has been integrated into everyday tasks that read the data, such as index rebuilds. In 10*g*, statistics are maintained automatically – after initial collection with `DBMS_STATS` – via the normal reading of data by server processes. In 9*i*, this was also possible with the `MONITORING` segment attribute.

The use of segment monitoring should mitigate some of the instability associated with the CBO. Previously, it was necessary to collect statistics periodically. Suddenly analyzing tables that had never before been analyzed, or that had run without statistics for a long time often produced unexpected results.

A variety of features have been developed over the life of the CBO to improve stability. In Oracle 8*i*, query plan stability was introduced, allowing desirable query plans to be frozen and preserved. Optimizer statistics for a segment can also be moved around starting in 8*i* using `DBMS_STATS.EXPORT...`. This allows DBAs to save prior statistics when analyzing tables. It also allows the application of production statistics to a smaller development database for more realistic testing of applications and query plans prior to rolling out new or changed applications into production.

Nevertheless, be prepared to closely monitor query plans on any instance that undergoes an Oracle version upgrade. So many changes go into each new revision of the CBO that query plans can change dramatically from version to version.

Sometimes, despite your best efforts, a statement that performs well under the RBO will perform poorly with the CBO. If tables are properly analyzed, and no cause for inefficient CBO behavior can be determined, then users should contact Oracle Support to report and thoroughly document the issue. Oracle cannot improve a product if bugs go unreported.

Filesystem maintenance and cleanup

If you have ever experienced a few core dumps (ORA-7445) over a short period, you know that Oracle's core files can be large, because they dump the whole SGA. An Oracle process that dumps core should not take down a whole system, but in many environments it easily fills all available free space in a filesystem. This can cause a variety of availability problems including the inability of the listener to write to its logfile, which actually causes the listener to hang in some versions. A full filesystem could also prevent the archiver from working, which would eventually case an instance to stop accepting transactions.

One good solution is to place the `user_dump_dest` and `background_dump_dest` on their own filesystems separate from the database files, listener log and archived redologs. Alternately, some Unixes allow quotas to be imposed on certain directories so that they cannot exceed a specified size. This approach is

preferable to setting `max_dump_file_size`, which can truncate trace files that are needed for debugging a problem.

To prevent filesystems from filling up in the first place, jobs should be scheduled to run periodically to clean up dumps and archivelog directories. Such scripts should be smart enough not to delete needed files. For trace files, a good script will delete from oldest to newest, bringing the filesystem utilization down to a threshold percentage, such as 50%.

For archivelogs, a cleanup script should be smart enough never to delete logs that have not been backed up. For RMAN backups, determining which logs have been backed up is a matter of an easy lookup in `v$backup_redolog`. For your own non-RMAN archivelog backups, some kind of separate state file or table has to be maintained by the backup jobs in order to track log sequences that have been backed up.

In addition to cleaning up trace, core and archive files, filesystem maintenance scripts must also be able to trim listener logs and the alert file. The alert file is unproblematic to manage, because Oracle's logging functions used to write to it do not hold the file open. If the file is renamed, Oracle will just start a new alert file named `alert_<sid>.log`.

The listener logs are not so dynamic. The listener holds its log file open, and if the file is moved, it will continue writing to the renamed file. This makes operating the Net8 listener in a high availability environment a challenge. One way to truncate the listener log in Unix is to cat the null device into the listener log, as in the following example:

```
$ cat /dev/null > listener.log
```

This clears the log file, and the listener continues writing to it. You may prefer to retain the data from listener logs for capacity planning and problem resolution. If so, the listener log should be copied periodically to a dated file, and the log truncated. This operation should be conducted during a low usage period, because you will lose any entries made in the log between the copy and the truncation.

Redo-related pauses

When an Oracle instance runs out of available online redologs, it will pause, and no new operations or logins will be permitted until the situation has cleared. That constitutes an availability problem, even if it is brief. Even very brief downtimes detract from the overall experience of users.

When Oracle fills an online redolog, it cannot use that logfile again until it has been archived (assuming the database is in ARCHIVELOG mode), and the database has been checkpointed. If the instance proceeds through all redologs before archival or a checkpoint has occurred, Oracle will hang because it does not have any logs available in which to record transactions.

If the archiver cannot keep up with the rate of redo generation, multiple archiver processes may be used to archive faster. However, steps should be taken to determine the cause of and reduce the rate of redo generation if possible. The best way to determine the source of heavy redo generation is to look in `v$sesstat`, where the redo size of every session is recorded by the timed statistics feature. The following query displays redo generation (per unit time logged in) by all currently logged-in sessions, ordered by volume:

```
select module, osuser, sql_hash_value, value / (sysdate - logon_time) redo
from v$session s, v$sesstat ss, v$statname sn
where s.sid = ss.sid
and ss.statistic# = sn.statistic#
and name = 'redo size'
order by redo;
```

Often the archiver only falls behind during periods of heavy transactional activity associated with large data loads or index rebuilds. The UNRECOVERABLE option is a good way to mitigate excessive redo generation for repeatable data loads and for index builds. You need to understand your own recovery scenarios and path before using UNRECOVERABLE, but for repeatable activities, it can be an important tool for achieving scalability.

If Oracle cannot complete a checkpoint (“checkpoint not complete” in the alert file) before using all available online redologs, then there may be any of a number of issues at work. While the boilerplate solution for this problem is to add more redologs, this may not be the correct solution. The most basic reason for checkpoints not completing is that the I/O subsystem is too slow to keep up with the demand in writes from the database. Many Unix filesystems do not allow asynchronous I/O, so every write operation attempted by the database writer must complete before it can go on the next write. Furthermore, some Unix filesystems serialize write operations to a single file, defeating any attempts to improve I/O performance by using multiple database writer I/O slaves. This problem actually causes one I/O slave to block others for writes to a particular datafile, ultimately slowing down overall operation. Asynchronous I/O allows write operations to be issued much more rapidly than synchronous operations, dramatically decreasing the overall time required to complete a checkpoint. Most platforms support asynchronous I/O on raw device datafiles, and some filesystems also have their own software implementations of asynchronous I/O. Many checkpoint slowness problems can be overcome by switching to asynchronous I/O.

If asynchronous I/O is not a possibility, the database writer can be set to checkpoint more aggressively. From 8i forward, you can switch to fast-start checkpointing. This will increase the frequency of checkpoints, but will ultimately allow checkpoints to complete in a timelier manner. In 10g, the aggressiveness of the checkpointing activity is automatically adjusted. If you provide no log_checkpoint... parameters, 10g will use fast-start checkpointing by default and automatically adjust the fast_start_io_target.

Checkpoint currency and recovery

In the event Oracle must perform instance (crash) recovery, the speed with which it completes depends largely upon how recently the last checkpoint occurred. When Oracle performs instance recovery, it must apply all redo entries from the online redologs from the time the database last checkpointed each datafile, forward to the time the database went down. Therefore, to minimize downtime in the event of an instance failure, the database should be kept checkpointed to as recent a point in time as possible, preferably with fast-start checkpointing.

If a database host has multiple processors, instance recovery can be performed in parallel to minimize the time required to open the database. By appropriately setting the values of parallel_max_servers and recovery_parallelism, any recovery will be performed in parallel, and should complete faster than a single-threaded recovery.

When deciding on values for these settings, some experimentation should be performed in order to determine the best recovery times for any particular architecture. Some papers have stated that no benefit from parallel recovery can be realized with less than eight parallel processes, but since architectures vary widely, so will any given result.

Recovery from user oopses

Even with strict access policies and privilege controls, it is still likely that someone will accidentally drop a table, update rows or delete critical data. This may be data that a critical application needs just to run. Such user oopses can result in extended downtimes for an application or service.

One of the best new features in Oracle 10g for mitigating the impact of a DML-related oops is flashback query. Instances that use server-managed undo instead of rollback segments can perform consistent reads of data as it appeared in the past. Using the same multiversion consistent read mechanism that allows Oracle’s writers never to block readers, undo is applied to data to provide output as of a time in the past.

Flashback query is simple to use, and allows previous versions of data to be queried from a select statement, using the “versions between” clause. After a user oops, it is critical to perform the flashback query as quickly as possible to get the data back, or the undo retention time may expire. To minimize the time spent writing the results, you can “create table ... nologging as select ... versions between...”

The flashback drop feature which is supposed to protect from accidental table drops, is less useful, and is only related by name to flashback query. When you drop a table in a database with flashback drop enabled, it actually just renames it, until the db_flashback_retention_target expires, at which point it is dropped.

Avoiding reliance on DNS

When setting up listener.ora and tnsnames.ora addresses, either the IP address or DNS name of the target machine must be specified. Some prefer to use the DNS name rather than the IP address for reasons of transparency. The problem with using DNS names for Net8 addresses is that it causes you to rely on name resolution via DNS (prone to outage) or a local hosts file (prone to corruption).

In order to eliminate the hosts file and the DNS as points of possible failure for an Oracle Net8 implementation, IP addresses should be used in listener.ora and tnsnames.ora entries for critical services. Although the transparency benefits of using DNS are lost, mistakes in hosts file or DNS configuration cannot negatively impact the ability of clients to establish connections via a Net8 listener.

A larger lesson can be gleaned from this tip. When considering configurations for any critical service, make sure you rely on as few points of failure as possible. The more external services you make a system rely on, the more points of potential failure you create.

TIPS FOR HUMANS

Although availability seems like a technical issue, it is also largely a human issue. A great deal of downtime can be traced not to technical failures, but rather to human failures. Just a few of these are:

- Lack of diligence or followup on previous problems
- Panicking under pressure of an outage
- Laziness
- Failure to grasp the financial impact of the DBA's role in a company
- Lack of planning
- Failure to focus on the most probable problems

My last few tips therefore address the DBAs themselves, and how they can prepare themselves for the inevitable major failure over which they will one day preside.

Remote access / effective communications

Essential to any 24x7 operation is a dependable remote access capability. DBAs and other key support staff need to be able to quickly log in from home or the road in order to resolve problems. In many IT organizations, several different people may be called on to assist in problem solving, so remote access modem pools or VPNs should be large and dependable enough to handle many connected users. Depending on the criticality of operations, some larger organizations may decide to keep DBA and systems staff on site at all times.

I often hear that DBAs are not provided with company-funded pagers or mobile phones. Any company that expects their DBAs to respond to emergencies must provide a means to do so. It is up to the DBA to insist on this. A couple unattended major outages will be all that it takes to clear up this misunderstanding between DBAs and management.

On-site operators

Even if all staff is not on site, a staff of one or two systems operators located in the same place as the physical systems can be indispensable in an emergency. They can page and call people, maintain a conference call, and visually check systems. DBAs and sysadmins should be able to devote 100% of their time to fixing the problem in an emergency. If they have to waste time paging and calling people, it just extends the downtime. System operators who perform this support role, such as those at Amazon.com, are the true heroes of high availability.

Bridge line

There is no substitute for a commercial conference system that can bridge together 20 to 50 people on a single call. When all the minds come together, ideas for resolution are more likely than with one person pecking away in the dark.

Application failure tolerance

Despite the best efforts of any staff, outages will take place from time to time, and software must be prepared to handle that eventuality gracefully. By designing applications in such a way that they display a friendly message, such as "This service is temporarily unavailable - Please try again later," the impact of outages on users can be lessened. The advantage of presenting a clear, friendly failure message to users is that it makes the outage less cryptic and confusing to users. Without such functionality, users see error messages that mean nothing to them, and give no indication that anyone is aware of a problem. With a "service unavailable" message, users can be told that the problem is known, and that people are working to correct it. Furthermore, the DBA rushing to solve the problem does not have to answer as many calls asking if she is aware of the problem.

Dealing with new and unfamiliar problems

It is a good idea to develop a game plan for unexpected outages. In most cases, the original root cause of a problem will require extensive analysis to uncover. For this reason, decide ahead of time on how a severe problem will be approached and managed. A diagnostic plan should be developed and made well known to all DBA staff.

Often the corrective action for a particular problem will have the side effect of eliminating any evidence of the cause of the problem. For example, a problem with the SGA will be completely cleared by restarting the instance. DBAs should therefore be able to rapidly perform some cursory diagnostic actions, then proceed to concerted efforts toward the task of restoring service.

Taking this approach is especially effective upon encountering the first occurrence of a particular problem. After restoring service, ample time can be spent on thinking through the root cause, and developing strategies for better diagnosing or correcting the problem if it ever happens again. The DBA's ability to perform post-event root cause analysis is completely dependent on the quality of data recorded during the event.

Chief among the tools for obtaining such data tools is the system state dump. Logged in as a user with DBA privileges, a system state dump must be performed when connected to the database through a dedicated server process.

The SQL statement for generating a system state dump is:

```
SQL> alter session set events 'immediate trace name systemstate level 10';
```

A system state dump will create a trace file in the instance's `user_dump_dest`, containing the wait states and resource dependency hierarchies of all sessions logged in. Systemstate dumps are primarily for hanging situations and resource pileups. Oracle has written a useful awk script that parses systemstate dumps and provides a clear, readable output. Ask your Oracle Support analyst for a copy of `ass.awk` (I'm not kidding). Many other diagnostics can be easily and quickly dumped, depending on the type of problem that a system is encountering. The information contained in the resulting trace files can be valuable in diagnosing a wide variety of database problems, and can also be helpful when working with Oracle Support.

In addition to database diagnostics, it is also important to check host resources, such as memory, swap and disk space. Many of these emergency diagnostic techniques can also be run routinely at periodic intervals, as part of a meticulous monitoring regime. If particular areas cause problems, additional diagnostic queries and tests can be added to monitoring scripts to provide early warning of an impending problem or better post-outage diagnostics.

These guidelines are limited in that they cannot take into account the particular nature or circumstances of any particular critical problem. The best diagnostic methods are specific to a problem, and successful root cause analysis depends on the ability of the DBA to investigate. In a nutshell, postpone as much analysis as possible for after the system is back up. Get your diagnostics fast and return the system to service as best you can. Remember, availability means having the host up as much as possible, not necessarily understanding why it last went down.

CONCLUSION

Much of the availability revolution consists of a shift in attitude. Those who have previously regarded downtime as inevitable, or as a necessary evil, will adapt to increasing availability requirements, or fall by the wayside. Over time, more and more DBAs will lose their scheduled weekly or monthly outage windows as their companies conclude that any and all downtime is unacceptable. As the volume of business transacted on the Internet grows, so will the amount of revenue lost per second of downtime.

Very visible public outages can be damaging to a company's credibility. They not only have the potential to be expensive in terms of lost revenue, but they also diminish customer confidence, projecting an incompetent image to the public. Certain sites have experienced notorious outages that people remember and discuss long after the event has passed. Because lengthy downtimes are so preventable, an attitude must be adopted among IT management and staff that treats availability issues very seriously. Adopting a *laissez-faire* attitude toward availability leaves a company's success up to luck, and outages resulting from such an attitude may be deserving of the incompetent image it may paint of a company.

Oracle's focus on availability has increased with each successive version of the RDBMS, reflecting user demand for such features. While some sites will not need all of the new availability features, and will not need to employ every technique outlined here, other sites will find them indispensable, and all would be wise to consider them.