A Unix Perspective on Oracle Archive Redo Log Files

Mark Bole

I am keenly interested in how organizations define and manage the respective roles of the Unix Systems Administrator (SA) and Oracle Database Administrator (DBA). Many reasonable people will maintain that it is unlikely to find one person who can perform both jobs well, while others have enjoyed great success in environments where there is shared responsibility for both the root password and the SYSTEM password. Auditing concerns, career paths, on-call schedules, and egos further cloud the issue, sometimes leading to the construction of brick walls where perhaps a picket fence or even a chalk line on the ground would be more appropriate.

Those who are not familiar with the latest version of Oracle's flagship product, Database 10g, may be surprised to find that the distinction between SA and DBA is becoming even less clear. The very first document you see is called "2 Day DBA", which is targeted to someone with no previous DBA experience, such as an SA, and which promises to cover all the basic tasks one typically performs in support of Oracle 10g for a small- or medium-sized organization. Also, Oracle now provides many features that previously may have been considered the sole province of the SA, such as:

- Logical disk volume manager (for all database files)
- Cluster file system and clusterware management that requires no third-party software
- Lempel-Ziv compression (like gzip)
- Copy any operating system file locally or between servers
- Job scheduling capabilities, including jobs completely external to the database, that vastly exceed those of cron
- POSIX-compliant regular expression handling

Nevertheless, one specific feature of Oracle has been in place from the beginning — the archived redo log file, which is arguably the single most important component of a production application for the SA and DBA to jointly manage and protect. In this article, I will review archived redo log files (ARLFs) from an SA perspective, without the need to log in to the database or use SQL statements. I will cover where archived ARLFs come from, where they go, what they contain, how their use has evolved over time, and how they can make or break your system. A summary checklist of issues and questions will be provided that the SA and DBA can review together to help ensure improved support and coverage.

Note that full documentation for all Oracle features mentioned is freely available at the following site:

```
http://www.oracle.com/technology/documentation/database10g.html
```

Redo — Another Word for Journaling

If you understand the concept of a journaling file system, such as ext3 under Linux, then you also understand most of the concept behind an Oracle redo log. Every Oracle database has at least two, and usually three or more, online redo log groups that exist as separate files in the file system (or on a raw partition), typically between 10 and 100 MB in size. They are written to by the Oracle logwriter process in a round-robin fashion and, in fact, are the first place on disk to which any changes to the database are written, even before the datafiles themselves. If the database crashes, say, due to an unintended shutdown of the server, the data in these files is used to help perform what Oracle refers to as "instance recovery" upon the next startup, ensuring that any database transactions that

were committed before the crash remain intact, even if they were not yet written to the permanent datafiles.

As updates happen in the database over time, the logwriter fills up each online redo log and moves on to the next one (a log switch), eventually coming back to the first one. At that point, it will overwrite the previous contents, which means the amount of data saved for recovery is limited by the total size of all the online log groups. If there is a crash involving the physical loss of a datafile due to disk failure or accidental erasure, for example, then the complete sequence of redo log contents, going back to the last backup of the datafile, will be needed to perform what Oracle refers to as "media recovery" and ensure that no data was lost. To accomplish this, most databases run in ARCHIVELOG mode, which causes the logwriter not to overwrite any online redo log until its contents have been archived (copied) to some other disk-based location.

In the beginning, this was the sole purpose of ARLFs — to provide for restoration of a database backup up to the time of the most recent ARLF, in the event of disk failure or even a complete site-wide failure. The main concern of the SA was two-fold:

1) To make sure that all ARLFs were reliably backed up together with the core database files, usually to tape or another off-server location.
2) To monitor the directory where ARLFs were stored so it didn't run out of space.

If, during normal operation, the filesystem where the ARLFs are stored fills up, the logwriter cannot overwrite the oldest unarchived online redo log file, bringing the entire database to a screeching halt until the space problem was addressed.

Here is a sample file listing (abbreviated `ls -lt` output) showing an example of the kind of behavior that ARLFs can exhibit. Note that in this two-hour window, the rate of disk consumption has suddenly gone from roughly half a megabyte per hour to nearly 100 MB per minute — a 12,000-fold increase!

```
20968960 Jan 30 18:10 binc02_0001_0000000097.dbf
20970496 Jan 30 18:10 binc02_0001_0000000096.dbf
20967936 Jan 30 18:10 binc02_0001_0000000095.dbf
20969984 Jan 30 18:10 binc02_0001_0000000094.dbf
20971008 Jan 30 18:10 binc02_0001_0000000093.dbf
20970496 Jan 30 18:09 binc02_0001_0000000092.dbf
20970496 Jan 30 18:09 binc02_0001_0000000091.dbf
20971008 Jan 30 18:09 binc02_0001_0000000090.dbf
20971008 Jan 30 18:09 binc02_0001_0000000089.dbf
  628736 Jan 30 17:10 binc02_0001_0000000088.dbf
  330240 Jan 30 16:10 binc02_0001_0000000087.dbf
```

What's more, this could be either normal and expected, or a symptom of something gone terribly wrong, such as a software bug or someone making a data alteration they should not be making, at least not at this time of day.

Finding the Way to Oracle Home

To stay on top of ARLF generation, you need to know where they are. Let's suppose that you have taken over a running database system and that there is no one to ask about how it was configured. The first thing you need to find is the Oracle "home", the unique base directory used for each installation of Oracle software on the server. To support concurrent installation of different versions and different products, there can be many Oracle homes on the same

box. The best place to look is in the oratab file, which is created as part of the Oracle software installation process. The file will be located as follows:

Solaris: `/var/opt/oracle/oratab`
AIX, HP, Linux and Tru64: `/etc/oratab`

The entries in this plain-text file consist of three colon-separated fields as follows:

```
ORACLE_SID:ORACLE_HOME:{Y|N }
```

For each database instance configured on the server, the entry in oratab shows its name, Oracle home location, and whether it should be started and stopped by the Oracle-supplied dbstart and dbshut scripts whenever they are run. Another, less reliable way to discover the candidate locations for ORACLE_HOME is to examine the shell startup scripts (.profile, .bashrc, etc.) for the "oracle" user account, which typically will have at least one setting for an ORACLE_HOME environment variable.

Once you have a list of ORACLE_HOME locations for the box, set your $ORACLE_HOME shell environment variable to each value in turn and look in each location for a subdirectory named dbs. In this subdirectory, along with a few other default files, you will typically find files of the form initSID.ora, where SID is the specific database instance name corresponding to the first field in the oratab file.

For example, if the oratab entry is binc02:/u01/app/oracle/product/9.2.0:Y, then the SID is "binc02", and the file you are looking for will be named "initbinc02.ora". This parameter file (PFILE) is essential for starting up any instance of Oracle. Of particular relevance is the parameter that indicates where ARLFs are to be generated. In the simplest case, you will be able to examine the PFILE for each instance and find one or more entries that look like this:

```
log_archive_dest_1 = \
  'LOCATION=/u01/app/oracle/admin/binc02/arch MANDATORY'
```

Up to 10 log archive destinations can be defined. If you find no current files in the directory specified by LOCATION=, then probably the database instance has not been started in a long time, or it is in NOARCHIVELOG mode.

Let's examine three common exceptions to this simple process for finding the ARLF location, especially those due to newer versions of Oracle. In addition to a standard plain-text PFILE, starting with version 9i, a more flexible binary version (known as a server parameter file, or SPFILE) can be used instead of a PFILE. Or, the PFILE can have a single entry pointing to a SPFILE, such as:

```
SPFILE='/opt/oracle/oradata/bin1r/spfilebin1r.ora'
```

If the dbs directory contains a PFILE pointing to an SPFILE as shown here, or if it contains an SPFILE directly, with a name matching spfileSID.ora or spfile.ora, then you need to examine the SPFILE for the log_archive_dest_n parameter. Fortunately, although this file is binary, you can easily examine its contents without logging in to the database by using the `strings` command, as this example using a SID of bin1r shows:

```
% strings spfilebin1r.ora | grep log_archive_dest_1
*.log_archive_dest_1='LOCATION=/opt/oracle/admin/bin1r/arch'
```

A second exception has to do with the dbs directory itself. If you find an ORACLE_HOME without one, then it represents an installation of a product that doesn't support a database, such as a client-software-only installation, or a cluster-ready services (CRS) installation. A third exception shows up when you discover an instance name or ARLF destination that begins with a plus sign "+" as in these two examples:

```
oratab: +ASM:/u01/app/oracle/product/10.1.0:Y

initbinc03.ora: SPFILE=+BINC_LAKE/BINC03/PARAMETERFILE/spfile.262.1
```

In this case, we are encountering a new feature available under Oracle 10g, Automated Storage Management (ASM). This is a disk volume manager that provides striping and mirroring for essentially all Oracle database files, using disk partitions, third-party volumes, or space on certified NAS devices. Unfortunately, examining and manipulating the contents of an ASM volume requires logging into the database, so it will remain outside the scope of this article.

There are two other much less common exceptions to note as well — non-standard (S)PFILE locations used when starting the database, and parameter changes made in the running database that aren't reflected in the (S)PFILE.

Redo Log File Transport Using Oracle Net

Beginning with Oracle version 8, a new type of database known as a standby database was provided. A physical standby database is basically a "live" backup, usually kept current with transactions no more than a few hours behind the primary database and ready to go live in a recovery situation in a matter of minutes. In Oracle 9i, a logical standby type was added, and both types are now known by the name Data Guard. The two hallmarks of a typical Data Guard database are that it is on a physically separate server from the primary, and transactions are copied into the standby from the primary database solely by way of reading the contents of the ARLFs generated on the primary (which has the benefit of not adding any processing load to the primary database, as other types of data replication typically would).

Although ARLFs can be copied and applied at the standby server under the control of Unix-level scripts, use of Oracle's automatic log transport mechanism is recommended. The SA can determine whether and how this is configured by looking in the (S)PFILE for a `log_archive_dest` parameter that references a SERVICE instead of a LOCATION, for example:

```
log_archive_dest_3 = 'SERVICE=binc02dg'
```

The service is an Oracle Net service, and the details can usually be determined by using the Oracle `tnsping` utility (found in $ORACLE_HOME/bin directory):

```
% tnsping binc02dg | grep DESCRIPTION
Attempting to contact (DESCRIPTION = (ADDRESS_LIST = \
  (ADDRESS = (PROTOCOL = TCP)(HOST = tree)(PORT = 1521))) \
  (CONNECT_DATA = (SERVICE_NAME = binc02dg)))
```

The output shows that the remote destination for ARLFs defined by this parameter is at a network address named "tree", where the Oracle Net Listener is using the standard network port 1521. On the remote host "tree", you can look (using the techniques described above) for an ORACLE_HOME and a SID of "binc02dg"; in the (S)PFILE for this instance, you should find the following setting:

```
standby_archive_dest = '/u01/app/oracle/admin/binc02dg/stbyarch'
```

This indicates that the standby database "binc02dg" on host "tree" will store the copies of ARLFs it receives from the primary to the directory shown. From there, the Data Guard update process will apply the transactions to the standby database. Note that if a logical standby is in ARCHIVELOG mode, it will in turn generate its own local ARLFs — a good reason to have the parameters `log_archive_dest_n` and `standby_archive_dest` set to different directories in the standby instance. Also note that there is not a one-to-one correspondence between the two sets of ARLFs — it is not unusual for the local ones to consume much more space than the received copies.

To avoid data loss due to the primary database being a single point of failure, Oracle provides a Data Guard configuration where transactions won't complete at the primary database until they are first written to the standby ARLF. Obviously, you will want an extremely reliable network and server configuration before putting this restriction on your primary database! In a more typical scenario, you will accept slightly increased risk by simply making sure that ARLFs are generated frequently, say every 15 or 30 minutes, which will become the maximum possible time period of lost data in the event of an unrecoverable failure of online redo logs at the primary site. ARLFs are normally generated only when the online redo log fills up and a log switch must occur; however, the following parameter can be used on the primary database to automatically ensure that a log switch happens at set intervals whether needed or not:

```
archive_lag_target = 900    # value is in seconds
```

Security, Retention, and Performance Issues for Archived Redo Log Files

When disaster recovery was the only purpose of ARLFs, you could delete and forget all that were older than the last successful database backup. In current versions of Oracle's product, it is now possible to use Log Miner to extract every single database update directly from the ARLFs, without any continued access to the original database or its passwords. Even transactions that did not complete, or were rolled back, at the primary are available by mining the ARLFs. For this reason, all ARLFs and copies should be maintained as securely as the primary database. Because they are relatively small and typically copied to several locations, it is easy to lose track of them. If appropriate, your organization might consider encrypting them both in transport and in storage for added protection.

The ability to mine the contents of ARLFs also makes them useful for auditing or troubleshooting activities, and you may decide to keep them around for a year or more for this reason. For example, one time I noticed a pattern over several weeks of a huge surge in ARLF generation at a certain time early in the morning. Yet there was no immediate evidence of any users performing any unusual transactions. By mining the available ARLFs for the relevant time period, I discovered the ultimate problem — an application bug, coupled with Oracle's sophisticated handling of transaction isolation levels, was causing an attempted update to a table to be partially rolled back and re-attempted many hundreds of times. Without the ARLFs, it would have been nearly impossible to identify the exact user-initiated steps and database tables that led to uncovering the bug.

The "huge surge" in ARLF generation just described should always prompt further investigation by the SA and DBA. Even if the cause is known and expected, be aware that rapid online redo log switches and generation of ARLFs consumes a large amount of system resource and can noticeably impact response time for other users of the system.

A classic example that all SAs for Oracle systems should understand goes like this — suppose the database appears to be "hung" because of some kind of runaway transaction that is generating ARLFs at a high rate and is under pressure from users; the decision is made to bounce it immediately. The SA may end up killing the Oracle processes at the Unix level or even rebooting the server to ensure a clean restart. This is perfectly acceptable to the Oracle database, and no committed data will be lost due to automatic instance recovery. When the database starts up again, the "hung" transaction, which may have been, say, two-thirds complete, must be rolled back to guarantee database integrity. However, each database block that is rolled back is in turn logged, resulting in another huge surge of ARLF generation. This activity can actually be several times longer and consume several times more disk space than the original interrupted transaction, so in this scenario the users are actually out of service longer than if nothing had been done.

Oracle's Newer Automated Management Features

Recognizing that many aspects of managing ARLFs have historically been addressed at the Unix level — external to the database — Oracle has implemented several new features to relieve the SA of this burden. Even if you are not yet ready to migrate to Oracle 10g, knowing what is available can help you be better prepared when the time comes.

Traditionally, the names of the ARLFs are generated according to a format specification in the (S)PFILE, such as the following:

```
log_archive_format = binc02_%T_%S.arc
```

where %T is the instance thread number, and %S is the log sequence number. This can cause a conflict when a new incarnation of the database is created, which resets the sequence number back to 1. Using the wrong ARLF, because it has the same name as an older one from a previous incarnation, can be disastrous during a recovery effort. In version 10g, an additional format variable (%r) is now required to be part of the name to ensure unique file names across multiple incarnations of the database.

Also in 10g, the Flash Recovery Area (FRA) feature can be used to automate most disk-based backup and recovery functions. Within specified space and retention limits, Oracle will automatically manage the placement and eventual deletion of ARLFs. The FRA is also integrated with the Oracle Managed Files (OMF) feature, which will automatically generate unique names for most types of database files, including ARLFs. If the following parameter is set in the (S)PFILE, then ARLFs will be generated automatically in the FRA location shown:

```
db_recovery_file_dest='/opt/oracle/oradata/flash'
```

and the generated ARLF names will typically look like this:

```
o1_mf_1_104_0zxl4krz_.arc
```

Since files in the FRA are subject to automatic deletion, you may want to ensure an extra copy is maintained outside of the FRA, using parameters as shown below, where destination 1 is outside the FRA and destination 2 is in the FRA:

```
*.log_archive_dest_1='LOCATION=/opt/oracle/admin/bin1r/arch'
*.log_archive_dest_2='LOCATION=USE_DB_RECOVERY_FILE_DEST'
```

The Oracle backup and recovery tool, RMAN, is also integrated with the FRA and can take full advantage of automated management of ARLFs when performing its tasks.

Clusters and the Other Kind of Log File

Two additional topics related to ARLFs merit a brief mention. In a Real Application Cluster (RAC) configuration, two or more Oracle instances running on different servers share a common database. Each instance generates its own sequence of ARLFs, so it is important to identify all the destinations involved from the SPFILE for the cluster. Typically, the ARLFs will be copied to at least one common, shared location, using ASM or a cluster file system. Each instance uses a different thread number, so the source of each file will be normally distinguishable by the thread number embedded in its name.

When all else fails, good old-fashioned Oracle runtime log files (the human-readable ones) are an invaluable source of information for both normal operation and troubleshooting. To find the primary log file for an instance, look for the entry in the (S)PFILE similar to this:

```
background_dump_dest = /u01/app/oracle/admin/binc02/bdump
```

Among the other trace files in this location, a special trace file called the "alert" log, containing all the core messages for the instance, will be found with a name of alert_SID.log. Among other things, this file will show you all non-default parameters from the (S)PFILE that was used to start the instance. It also logs the creation time and success of each ARLF. For example, if Oracle is shipping ARLFs to a standby database that happens to be out of service at the moment, you will see error messages such as the following in the primary database alert log:

```
Wed Feb  2 16:49:33 2005
Errors in file /u01/app/oracle/admin/binc02/bdump/binc02_arc1_1726.trc:
ORA-12514: TNS:listener could not resolve SERVICE_NAME given in
  connect descriptor
```

By examining the referenced trace file, binc02_arc1_1726.trc, you find the following:

```
Error 12514 attaching to destination LOG_ARCHIVE_DEST_3 standby
host 'binc02dg'
Heartbeat failed to connect to standby 'binc02dg'. Error is 12514.
```

The messages found in the alert log and other trace files will be your primary source of information when working with other support personnel.

Conclusion

Whether you work closely with — or worlds apart from — your DBA counterpart, it behooves the SA supporting an Oracle database server to understand the impacts of the database on the rest of the system and know where to find key information about how

things are configured, especially under the newest version of Oracle. You may find the following brief checklist handy as a starting point for further discussion with the DBA in your shop:

1. Are oratab files up-to-date and in the standard location for all instances?
2. Are all parameter files (PFILE/SPFILE) in the standard location, backed up, and contents documented?
3. Are there any user-managed scripts at the Unix level for copying ARLFs to remote locations?
4. If using Flash Recovery, are additional copies of ARLFs stored outside the Flash Recovery Area?
5. If using ASM, are additional copies of ARLFs generated outside of ASM storage?
6. Do all ARLFs need to be kept secure (encrypted)?
7. How long do ARLFs need to be saved?
8. Are ARLFs generated frequently enough (minimum every 15-30 minutes), or do you implement maximum protection mode for Data Guard standby databases?
9. Are log file names standardized and forward compatible with 10g?
10. Are logical standby local ARLFs stored in a different directory from the ARLFs received from the primary database, or alternatively, can the logical standby run in NOARCHIVELOG mode?
11. Are database maintenance and update activities involving large transactions fully tested with respect to the quantity of ARLF generated before going into production?
12. Are Oracle alert log files regularly monitored for error messages?

Author — Please provide 2-3 sentence bio for publication. Thank you!