DBFS
TECHNICAL
BRIEF

Database File System (DBFS) Best Practices

Introduction

Relational database systems have traditionally not been very good at storing unstructured or file based data. Processing unstructured data in a relational database involves being able to stream large amounts of sequential data and the performance, and storage requirements, have not been on a par with traditional file systems.

The desire to do this has traditionally been associated with content management type systems where the unstructured data is combined with metadata about the content, and keeping the two in sync is difficult to control when the unstructured data resides in a file system and the metadata resides in a database. Having all data in a database, like the Oracle database, also provides a consistent application-programming interface along with the protections of the Oracle database for security, recoverability and consistency.

Oracle SecureFiles provides file system like performance for unstructured data along with compression, deduplication and encryption capabilities. Oracle Database File System (DBFS) provides a file system interface on top of Oracle SecureFiles, making it practical to store unstructured file based data in the database alongside metadata about that file based data.

About Oracle Database File System (DBFS)

The Oracle Database File System (DBFS) creates a standard file system interface on top of files and directories that are stored in database tables. DBFS is similar to NFS in that it provides a shared network file system that looks like a local file system. Like NFS, there is a server component and a client component. In DBFS, the server is the Oracle Database. Files are stored as SecureFiles LOBs in a database table.

A set of PL/SQL procedures implement the file system access primitives such as create, open, read, write and list directory. The implementation of the file system in the database is called the DBFS Content Store. The DBFS Content Store allows each database user to create one or more file systems that can be mounted by clients. Each file system has its own dedicated tables that hold the file system content.

Using DBFS (Configuration, Usage and More…)

Please see the: **Oracle Database SecureFiles and Large Objects Developer's Guide**

RELATED DBFS MOS NOTES

- **1999798.1** – How to setup up 11.2 DBFS files systems using the dbfs_client API
- **193842.1** – How to setup 12c DBFS file system
- **1591515.1** – How to setup/configure 11.2.0.X DBFS file system on RAC configurations CRS Managed (Non-Exadata) demo
- **1150157.1** - List of critical patches required for Oracle 11.2 DBFS and DBFS Client
- **1320683.1** - How to trace DBFS when any failure happens

RELATED ORACLE WHITE PAPERS

For details (and examples) regarding configuring DBFS for use with Oracle RAC and Oracle GoldenGate, and for more information about Oracle SecureFiles, please see these Oracle White Papers:

Full Stack Role Transition: Oracle DBFS and Oracle Data Guard

SecureFiles Technical White Paper

Migrating to SecureFiles White Paper

DBFS Best Practices

DBFS Client Failover Configuration Best Practice for a RAC / HA Environment

Oracle Documentation Link:

For a description of DBFS and its benefits, please see the *Oracle Database SecureFiles and Large Objects Developers Guide*: https://docs.oracle.com/database/122/ADLOB/DBFS-securefiles-store.htm#ADLOB4594

The Oracle DBFS Client (dbfs_client) documentation recommends modifying service to include failover parameters, but these parameters are typically general RAC parameters for failover – this *DBFS Best Practice Technical Brief* includes recommendation(s) specific for RAC/HA configurations.

A single OS file system command comprises a sequence of DBFS client requests. Each of the DBFS client requests, associated with a single file system command, is by default directed to any node in the RAC cluster. When using DBFS with RAC/HA, a dbfs_client should always mount with the *-o failover* option, since failover not only handles failover scenarios, but also allows for seamless execution of these requests across different RAC nodes. If "-o failover" is not set, some sub-steps associated with a file system command may fail, since a DBFS request associated with a sub-step may be directed to a different RAC node from the previous step.

An Oracle best practice recommendation for RAC/HA environments, when using the *failover* mount option, is the use of a *service_name*. A service_name is used for client applications, housed on a separate server, that are already using dbfs_client connecting remotely to the DBFS database. Using this recommended service configuration will then permit dbfs_client to automatically connect to the correct DBFS instance in the event of a node failure. This also could be used, in this scenario, to reduce possible data loss that could occur between the dbfs_client cache and the database when a failure occurs.

The dbfs_client program can failover to one of the other existing database instances if one of the database instances in an Oracle RAC cluster fails. For dbfs_client failover to work correctly, **you must modify the Oracle database service and specify failover parameters**.

Run the DBMS_SERVICE.MODIFY_SERVICE procedure to modify the service as shown the example below:

```
exec DBMS_SERVICE.MODIFY_SERVICE(service_name => 'service_name',
        aq_ha_notifications => true,
        failover_method => 'BASIC',
        failover_type => 'SELECT',
        failover_retries => 180,
        failover_delay => 1);
```

DBFS and Exadata:

Please see this Oracle White paper for more information about using DBFS with Exadata.

DBFS use case performance on Exadata configurations