Ein Dateisystem innerhalb der Datenbank – Oracle SecureFiles/DBFS in der Praxis

Thomas Krahn, PLATH GmbH

Oracle SecureFiles/DBFS ist ein hochverfügbares, skalierbares Dateisystem innerhalb der Datenbank mit einer Performance vergleichbar mit normalen Dateisystemen. Dieser Artikel gibt einen Überblick und deckt Hürden und Tücken bei dessen Benutzung auf.

Mit der Version 11 der Datenbank bietet Oracle das integrierte Feature „Oracle SecureFiles/DBFS". Dieses stellt mittels der DBFS_CONTENT API ein Dateisystem bereit, dessen Inhalt komplett in der Datenbank abgelegt ist. Dateien und Verzeichnisse können durch das Kommandozeilen-Tool dbfs_client oder das Dateisystem auf Linuxsystemen als Verzeichnis gemountet werden. Die übliche Client-Server-Trennung mittels SQLNet-Session verbindet Datenbankserver und Clients. Zusätzliche Features wie „Advanced Compression", „Advanced Security" und „Partitioning" ermöglichen es, Dateien zu komprimieren, zu deduplizieren oder zu verschlüsseln sowie das Dateisystem zu partitionieren.

Installation und Konfiguration

Die Installation könnte einfacher nicht sein. Serverseitig ist lediglich der Oracle Database Server (Version 11*g*) zu installieren. Der DBA muss nur einen separaten Tablespace zur Speicherung des Dateisystems sowie einen Datenbank-Nutzer mit den nötigen Rechten zum Arbeiten mit DBFS anlegen. Das eigentliche Erstellen eines DBFS geschieht über die DBFS_CONTENT API. Oracle liefert dafür die SQL-Skripte dbfs_create_filesystem.sql sowie dbfs_create_filesystem_advanced.sql mit, die in $ORACLE_BASE/rdbms/admin stehen. Doch Vorsicht! Wer sich an die Oracle-Dokumentation zu DBFS hält und mit dbfs_create_filesystem.sql ein DBFS erstellt, kann schnell ein Lizenzproblem bekommen. Das Skript erstellt von Hause aus ein partitioniertes Dateisystem. Selbst in der SE-Version wurde diese Implementierung nicht geändert, sodass der Versuch des Erstellens sogar mit einem „ORA-00439 Feature not enabled" quittiert wird. Auch die Verwendung von Komprimierung oder Verschlüsselung der Daten sowie der Deduplizierung der Dateien ist mit diesem Skript nicht möglich. Mit dem Skript dbfs_create_filesystem_advanced.sql hingegen hat man die freie Wahl, diese Optionen zu nutzen. Vor Verwendung der Features sollte man jedoch die Lizenzen prüfen. So bedarf die Nutzung von Kompression und Deduplizierung einer Advanced-Compression-, die Verschlüsselung einer Advanced-Security-Lizenz. Nicht zu vergessen die Partitioning-Lizenz, wenn das Dateisystem partitioniert werden soll. Ein einfaches Dateisystem, das keine zusätzlichen Lizenzen benötigt, kann wie folgt erstellt werden:

```
# sqlplus <dbfs_user>/<dbfs_
user_pass> @$ORACLE_HOME/rdbms/
admin/dbfs_create_filesystem_
advanced.sql TESTFS DBFS_TBS
nocompress nodeduplicate noen-
crypt
non-partition
```

Das Skript erstellt unter dem Benutzer „dbfs_user" ein Dateisystem TESTFS im Tablespace DBFS_TBS, welches die Daten nicht komprimiert (nocompress), Dateien nicht dedupliziert (nodeduplicate), die Daten nicht verschlüsselt (noencrypt) und die Tabelle, die das DBFS enthält, nicht partitioniert (non-partition). Wer seine Daten komprimieren möchte, hat drei weitere Möglichkeiten: „compress-low", „compress-medium" und „compress-high". Der Grad der Komprimierung ist wie bei jeder Komprimierung stark von den zu komprimierenden Dateien abhängig. Wer viele identische Dateien in seinen DBFS speichert, kann von der Deduplizierungs-Option (deduplicate) Gebrauch machen. Hierbei wird nur noch die erste der Dateien im DBFS hinterlegt – alle folgenden identischen Dateien werden mit einer Referenz auf diese versehen. Dies kann je nach Dateigröße und Häufigkeit des Auftretens viel Plattenplatz sparen.

Sensible Dateien lassen sich mittels der Verschlüsselungs-Option (encrypt) verschlüsselt hinterlegen. Bei dieser Option sollte man die CPU-Nutzung des Datenbank-Servers genau im Auge behalten, da sie zusätzliche Ressourcen benötigt. Bei der Partitionierung eines DBFS lässt Oracle dem DBA leider nicht genügend Freiraum. Aktuell sind lediglich der Dateiname, der Verzeichnisname oder die GUID einer Datei / eines Verzeichnisses als Partitionierungsschlüssel möglich. Erstellt werden grundsätzlich 16 Partitionen. Über den Hash-Wert des jeweiligen Partitionierungs-Kriteriums wird dann die zu verwendende Partition ermittelt. Die Möglichkeit, das Partitionierungs-Kriterium frei zu wählen, sucht man vergeblich. Wer noch mehr Optionen braucht, kann das DBFS auch direkt über die DBFS API erstellen. Diese erlaubt zusätzlich zu den bisher genannten Optionen auch die Angabe des zu verwendenden LOB-Tablespace sowie die Nutzung von BasicFiles statt SecureFiles.

Wallets zur sicheren und einfachen Benutzung

Da sowohl für das Benutzen von „dbfs_client" als auch zum Mounten eines DBFS die Authentifizierungsdaten des Datenbank-Benutzers, der das DBFS bereitstellt, notwendig sind, wird an dieser Stelle kurz auf die Verwendung eines Wallets eingegangen. Dahinter

verbirgt sich normalerweise eine durch den DBA erstellte, passwortgeschützte Datei, die Authentifizierungsdaten eines oder mehrerer Datenbankbenutzer enthält. Sie wird mit folgenden Befehlen erstellt:

```
# Wallet erstellen
$ mkstore -wrl /home/oracle/
wallet --create
# Authentifizierung an DB ORCL
hinzufügen
$ mkstore -wrl /home/oracle/
wallet --createCredential ORCL
<dbfs_user> <pass>
```

Das so erstellte Wallet-File kann nun auf alle Clients verteilt und bei der Verwendung von DBFS eingesetzt werden. Somit sind Passwörter für Datenbank-Zugriffe nicht mehr in Konfigurationsdateien oder Quelltexten zu integrieren. Zudem ist der Zugriff auf den Inhalt des Wallets durch ein Passwort geschützt. Bevor ein solcher Wallet benutzt werden kann, sind zusätzliche Anpassungen an der „sqlnet.ora“ nötig:

```
WALLET_LOCATION = (SOURCE =
(METHOD = FILE)
(METHOD_DATA = (DIRECTORY =
$HOME/oracle/wallet) ) )
SQLNET.WALLET_OVERRIDE = TRUE
```

Wer DBFS auch am Datenbank-Server nutzen will, sollte sich zuvor mit dem Bug 7258404 auseinandersetzen. Dieser beinhaltet, dass durch den Eintrag „SQLNET.WALLET_OVERRIDE = TRUE“ die Datenbank nicht mehr in einer SQLPlus-Session gestoppt und wieder gestartet wird. Man umgeht durch den von Oracle publizierten Workaround SQLNET.WALLET_OVERRIDE (wieder auf FALSE setzen) zwar das Problem, verhindert dadurch aber die Nutzung von Wallets.

Wenig Spielraum mit DBFS_Client

Das Kommandozeilen-Tool „dbfs_client“ steht nicht auf Windows-Clients zur Verfügung, dafür jedoch unter Solaris, HP-UP, AIX und Linux. Mit ihm sind die einfachsten Operationen wie Dateien kopieren und löschen, Verzeichnisse erstellen und löschen sowie das Auflisten von Verzeichnisinhalten möglich. Das Umbenennen von Dateien innerhalb eines DBFS ist aktuell nicht implementiert. Das Tool stellt mittels OCI eine Verbindung zur Datenbank her und arbeitet den entgegengenommenen Befehl ab. Nachfolgend die Benutzung des Tools unter Verwendung eines Wallet-Files (/@ORCL) statt des klassischen User/Pass-Verfahrens:

```
# Verzeichnis im DBFS erstellen
$ dbfs_client /@ORCL --command
mkdir test dbfs:/TESTFS
# Datei ins DBFS kopieren
$ dbfs_client /@ORCL --command
cp /tmp/test.dat dbfs:/TESTFS/
test
# Verzeichnisinhalt auflisten
$ dbfs_client /@ORCL --command
ls dbfs:/TESTFS
# Datei aus DBFS kopieren
$ dbfs_client /@ORCL --command
cp dbfs:/TESTFS/test/test.dat /
tmp
# Datei im DBFS löschen
$ dbfs_client /@ORCL --command
rm dbfs:/TESTFS/test.dat
# Verzeichnis im DBFS löschen
$ dbfs_client /@ORCL --command
rmdir dbfs:/TESTFS/test
```

Wer häufig große Dateien mit „dbfs_client“ übertragen muss, ist mit der Option „direct_io“ gut beraten. Diese übergeht den normalen Datei-Cache und hilft, besonders große Dateien schneller zu übertragen:

```
$ dbfs_client /@ORCL -o direct_
io --command cp /tmp/bigfile.
dat dbfs:/TESTFS
```

Maximaler Spielraum beim Mounten

Wer clientseitig Linux verwendet, kann ein DBFS auch wie ein normales Dateisystem in einen beliebigen Verzeichnisbaum einhängen (mounten). Dann stehen alle bekannten Befehle zur Dateimanipulation zur Verfügung. Während bei der Benutzung des „dbfs_client“-Tools für einzelne Dateien jedes Mal eine neue Session aufgebaut wird, entsteht beim Mounten nur eine einzige Session, was den Overhead in der Datenbank in Grenzen hält. Dazu muss jedoch das FUSE-Package der entsprechenden Distribution installiert sein. Weil damit jedoch zusätzliche Schichten zwischen dem eigentlichen Befehl und der Datenbank liegen, ist klar, dass diese Operationen nicht ganz so schnell abgearbeitet werden wie mit dem „dbfs_client“-Tool. Zum Einbinden benutzt man nicht den klassischen Mount-Befehl, sondern „dbfs_client“:

```
# DBFS per User/Pass mounten
$ dbfs_client <user>@<db> -o
direct_io /mnt/dbfs

# DBFS per Wallet mounten
$ dbfs_client /@ORCL -o
wallet,direct_io /mnt/dbfs
```

Wenn man das DBFS mit dem User/Pass-Verfahren einhängt, bleibt die Shell an dieser Stelle gesperrt, bis das DBFS mittels „fusermount –u“ wieder ausgehängt wird. Es besteht auch die Möglichkeit, das DBFS per Eintrag in „/etc/fstab“ zu mounten. Dann sollte allerdings sichergestellt sein, dass die Datenbank beim Start des Clients verfügbar ist. Wer nach dem Mounten ein unbrauchbares Verzeichnis in seinem Mount-Verzeichnisbaum vorfindet, hat sehr wahrscheinlich die falschen Authentifizierungsdaten verwendet. Leider weist bisher keine Fehlermeldung darauf hin. Neben dem Parameter „direct_io“ stehen noch weitere zur Verfügung, etwa zum Einrichten eines Failover-Mechanismus oder um den Zugriff auf das DBFS auch anderen Anwendern zu erlauben. Das eingebundene DBFS ist standardmäßig nur für den Benutzer zugänglich, der dieses auch gemountet hat. Um den Zugang auch anderen Nutzern zu ermöglich, müssen der Parameter „allow_other“ beim

Aktion	DBFS Time	Ext3 Time
Entpacken	1 h 52 min	23 min
Packen	51 min	58 s
Löschen	47 min	2 s

Abbildung 1: Entpacken und Packen des Linux Kernel im Vergleich zwischen Ext3 und DBFS

Mounten mit angegeben sowie ein zusätzlicher Eintrag in die Datei „/etc/fuse.conf“ erfolgt sein.

Performance

Laut Oracle-Dokumentation ist die Performance von DBFS vergleichbar mit herkömmlichen Dateisystemen. Zum Vergleich hat der Autor zwei Szenarien getestet. Als Testumgebung diente Oracle Enterprise Linux mit 11*g* R2 64Bit und ASM sowie separaten Datenspeichern für die zu vergleichenden Dateisysteme.

Im ersten Test wurde der Linux-Kernel in ein gemountetes DBFS entpackt und von dort wieder gepackt. Da er aus vielen 1 bis 4 KB kleinen Dateien besteht, simuliert man damit das Speichern und Laden vieler kleiner Dateien in ein DBFS. Die Zeiten wurden mit denen eines mit „Ext3“ formatierten Dateisystems verglichen. Das Quelllaufwerk des Kernels, das Ziellaufwerk des Ext3-Dateisystems sowie das Ziellaufwerk des DBFS-Tablespace waren dabei auf physikalisch getrennten, aber baugleichen Datenträgern abgelegt. Abbildung 1 zeigt die gemessenen Zeiten.

DBFS ist beim sequenziellen Schreiben und Lesen vieler kleiner Dateien in keiner Weise mit einem herkömmlichen Dateisystem zu vergleichen. Während des Tests war eine CPU-Auslastung von 65 bis 85 Prozent auf dem Client-System – bedingt durch den „dbfs_client“-Prozess – zu beobachten. Zudem stand der Database Writer unter enormer Last.

Im zweiten Szenario wurden Dateien verschiedener Größe immer wieder von einem Ext3-Laufwerk in das DBFS kopiert und von dort wieder hinaus. Die Dateien wurden zum Vergleich mit dem Linux-Befehl „cp“ auf ein gemountetes DBFS sowie mit dem „dbfs_client“-Befehl direkt in das DBFS übertragen. Abbildung 2 zeigt die gemessenen Zeiten.

Es ist deutlich zu sehen, dass ohne zusätzliches Tuning das Übertragen einer Datei etwa um den Faktor 5 langsamer ist (rote Balken). Das Übertragen mittels „dbfs_client“-Tools (DBFS_CP) ist meist schneller als das Übertragen in ein gemountetes DBFS, was durch die zusätzliche Schicht des FUSE erklärt ist. Erst durch Tuning der SQL-Session erreicht DBFS (gelbe Balken) eine ähnliche Performance wie das mit Ext3 formatierte Dateisystem. Für diesen Performance-Gewinn sind die Parameter „SEND_BUF_SIZE“ und „RECV_BUF_SIZE“ server- und clientseitig auf einen möglichst großen Wert anzupassen. Dies kann global über die „sqlnet.ora“ oder verbindungsspezifisch in „listener.ora“ und „tnsnames.ora“ geschehen. Die Default-Werte sind betriebssystemspezifisch. Für den Test wurden diese auf die 512k gesetzt. Da das „dbfs_client“-Tool das Umbenennen von Dateien aktuell nicht unterstützt, konnte dies auch nicht im Vergleich getestet werden. Wer jedoch ein nach Dateinamen oder Verzeichnis partitioniertes DBFS einsetzen möchte, sollte wissen, dass das Umbenennen einer Datei / eines Verzeichnisses über ein gemountetes DBFS länger dauern kann als das Übertragen der Daten in ein DBFS. Ändert sich der Hash-Wert des Datei- oder Verzeichnisnamens, muss die Datei aus der alten Partition ausgelesen und in die neue Partition übertragen werden, was zusätzlich Zeit kostet. Ferner ist zu beachten, dass im „ARCHIVELOG“-Modus jede Datei mit archiviert werden muss, was ebenfalls Zeit kostet. Insbesondere, wenn die zu übertragende Datei größer als die Summe der Onlinelogs ist, kann es schnell zu Engpässen kommen.

Fazit

Oracle SecureFiles/DBFS ist ein durchaus gelungenes Feature der Datenbank 11*g*. Installation und Konfiguration sind schnell erledigt. Durch die Verwendung von Verschlüsselung lassen sich auf einfachste Weise selbst sensible Dateien ins Dateisystem bringen.

Bestehende Datei-Tabellen können mittels der DBFS-API mit einem DBFS verbunden werden. Der Einsatz eines RAC macht das darin liegende Dateisystem hochverfügbar und lässt Freiräume für die Skalierung. In Verbindung mit Dataguard ist sogar das Replizieren eines Dateisystems kein Problem mehr.

Wer ein Dateisystem bisher umständlich mit „tar“ und „bzip“ gepackt und auf andere Standorte übertragen hat, kann dies nun ganz durch Transportable Tablespaces realisieren. Ein mit Flashback genutzter Tablespace für das DBFS ermöglicht zudem Blicke in die Vergangenheit von Dateien. Nicht zu vergessen ist, dass sich das DBFS mit RMAN sichern und wiederherstellen lässt.

Kontakt:
Thomas Krahn
thomas.krahn@plath.de

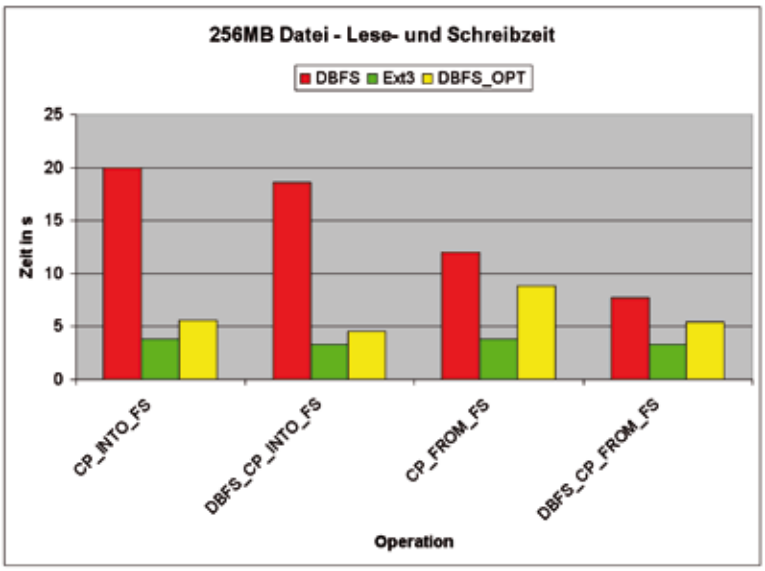

Abbildung 2: Datei-Operationen im Vergleich zwischen einem gemounteten DBFS und dbfs_client