Oracle Database 10g & Multi-Terabyte Database Migration

By: Saravanan Shanmugam and James Madison, The Hartford (http://www.thehartford.com/)

Migrating large Oracle databases such as data warehouses and decision support systems, which can easily consume many terabytes, presents one of the biggest technical challenges for any organization.

Prior to Oracle Database 10g, there were no high-speed tools available to migrate in a short time, such as over a weekend. Such migrations would require extensive planning, would have to be executed over the course of many days, and would require a mix of data movement technologies such as the export/import utility, "create table as select" (CTAS), or third party migration utilities that are not supported by Oracle.

Saravanan Shanmugam is an Oracle Certified Master, lead DBA and primary database architect for the platform migration on which this paper is based.

James Madison is the project manager and primary application architect for the migration.

Oracle Database 10g Cross Platform Transportable Tablespace Technology – XTTS (1) makes it possible to move large amounts of data across platforms in a comparatively short time with fairly straightforward planning. And Oracle Recovery Manager – RMAN (2) easily and quickly takes care of endian conversion, if required.

Assume your challenge is to move 7 TB across disparate hardware and versions of UNIX in less than 24 hours (a situation very similar to that of the authors'). How can this be done? The answer is provided below.

TECHNOLOGIES FOR MIGRATION

The process for selecting the technology set to facilitate large migrations requires considering the tools available and then choosing a set of technologies most appropriate for the various parts of the migration process. The technologies considered for the migration were:

Technology	Comments
Export/import	Classic data mover, available for years
Various third-party tools	Most originated prior to Oracle developed technology. XTTS and Data Pump make third-party tools far less valuable
Data Pump (3)	Available as of Oracle Database 10g
Cross platform transportable tablespace (XTTS)	Available as of Oracle Database 10g

The deciding factor was simply speed. We were willing to accept negatives in other aspects of the tool's use: high complexity, difficult interfaces, extensive environmental setup, manual intervention, and just about any other negative as long as we hit the target of moving 7 TB in 24 hours.

Export/import was chosen as the way to move metadata. But for the bulk of the 7 TB, it was quickly dropped. As most DBA's know, export/import on 7 TB would have taken forever. Third-party tools provided a glimmer of hope early on, but we determined that XTTS was a better bet for this task for reasons that will be discussed in this paper. Further, given the size and complexity of the environment in which most major data migrations occur, having all the tools supported by a single industry-leading vendor is a desirable safety net when technical problems occur.

Unlike traditional options for large migrations, XTTS and Data Pump provided the speed we needed, and despite our willingness to accept any secondary negatives, these tools also proved fairly straightforward to setup, use, and automate.

CROSS PLATFORM TRANSPORTABLE TABLESPACE

Oracle8i introduced the transportable tablespace (TTS), which allowed for fast movement of large amounts of data between databases with the same block size. TTS was a key development for large environments because previously, no Oracle utility could handle large volumes in a reasonable time. Oracle9i made TTS even more useful by removing the block size restriction, but like the previous release, Oracle9i still required homogeneous operating systems (OS's). Oracle Database 10g removed the final barrier by introducing Cross Platform TTS (XTTS) which allows TTS to function across heterogeneous OS's.

Now, tablespaces can be transported among databases of different block sizes residing on different OS's, thus allowing the DBA to move tablespaces virtually anywhere in the database environment.

Table 1 lists the major steps in a typical XTTS process. Further details are discussed later in this article. Other variations can be used based on the needs of your specific environment. Oracle documentation covers the details of such variations. Some of the main benefits of using XTTS are:

- Reduced overall complexity of the process because XTTS is a high-level copy of data
- Reduced errors since XTTS moves objects as a unit unlike table-by-table methods that could miss objects or rows of data
- No need to create or rebuild indexes
- Movement of tablespaces between heterogeneous OS platforms
- Endian conversion, if needed, is only 3% slower than a standard OS file copy
- Statistics do not have to be recollected for tables and indexes that are moved

```
SELECT * FROM V$TRANSPORTABLE_PLATFORM

PLATFORM_ID   PLATFORM_NAME                          ENDIAN_FORMAT
-----------  --------------------------------  --------------
          1  Solaris[tm] OE (32-bit)           Big
          2  Solaris[tm] OE (64-bit)           Big
          3  HP-UX (64-bit)                    Big
          4  HP-UX IA (64-bit)                 Big
          5  HP Tru64 UNIX                     Little
          6  AIX-Based Systems (64-bit)        Big
          7  Microsoft Windows IA (32-bit)     Little
          8  Microsoft Windows IA (64-bit)     Little
          9  IBM zSeries Based Linux           Big
         10  Linux IA (32-bit)                 Little
         11  Linux IA (64-bit)                 Little
         12  Microsoft Windows 64-bit for AMD  Little
         13  Linux 64-bit for AMD              Little
         15  HP Open VMS                       Little
         16  Apple Mac OS                      Big
         17  Solaris Operating System (x86)    Little
```

Query 1: Determining endian format

BYTE ORDER; LITTLE/BIG ENDIAN

The most critical functionality of XTTS is the ability to handle differing byte orders. When a word of data is written to a file, the bytes of the word can be written either with the least significant byte first or the most significant byte first. If the least significant byte is first, the byte order is little endian. If the most significant byte is first, the byte order is big endian.

For example, consider the decimal number 1032. This is 10000001000 in binary. If we assume a 32-bit word, pad with 0's accordingly, and put in spaces every eight bits for readability, we have: 00000000 00000000 00000100 00001000. If these bytes are to be written into the four byte addresses 753 - 756, they would have the following configurations for little and big endian:

Endian	Address			
	753	754	755	756
Little	00001000	00000100	00000000	00000000
Big	00000000	00000000	00000100	00001000

Figure 1: Byte order for little and big endian

To remember how this works, either remember that we as human beings write numbers in big endian, or use the three L's: **L**ittle endian puts the **L**east significant byte in the **L**owest address.

The Oracle tool that does the actual endian conversion is Oracle Recovery Manager – RMAN, using the CONVERT command (4). A detailed RMAN example is provided and discussed later in this article. To determine the endian format of your source and target machines, run the query shown in Query 1, which shows the results from an Oracle 10.1.0.3 database with the entries pertinent to the environment being discussed here highlighted.

	Event	Comments
1	Check endian format on both source and target databases	Query V$TRANSPORTABLE_PLATFORM as discussed later
2	Fix TTS violations in source database for the tablespaces being moved	Execute dbms_tts.transport_set_check ('ts1,ts2',true);
3	Place tablespaces being moved in read only mode in source database	Alter tablespace ts1 read only;
4	Validate the source database environment	For example, ensure that ORACLE_SID, ORACLE_HOME are set as expected, etc. See Oracle documentation for more.
5	Create directory in the source database for the data pump export	Create directory exp_dir as '/tmp2/dev/export' Only needed if you want export dumps on target for import
6	Grant read for the directory to oraop	Grant read on directory exp_dir to oraop;
7	Export meta data from source database	If you plan to take a dump file and move it to the target platform
8	Drop or rename tablespaces of same name in target database	A tablespace must have a unique name within a database. If the tablespace is shared between target databases, drop from all
10	Create the user concerned in the target database	If User does not exist in the Target database
11	Run RMAN CONVERT to convert to target endian	Only needed if endian format is different
12	Validate the target database environment	For example, ensure that ORACLE_SID, ORACLE_HOME are set as expected, etc. See Oracle documentation for more.
13	Create database link to source	Only needed if you want to import meta data over a link
14	Import meta data into target	Run the import script
15	Run the import meta data in another target database	If the tablespaces are to be shared between databases
16	Make tablespaces in source and target read write	Do not attempt to make it read write if the tablespaces are shared between databases

Table 1: Major steps in a typical XTTS process

DATA PUMP

Data Pump is a utility released with Oracle Database 10g. It has functionality similar to the export and import utilities, but is much faster. Consult Oracle's documentation for the full mapping between import/export and Data Pump.

The biggest advantage of Data Pump over export/import is the ability to pull tablespace metadata to facilitate plug-in tablespaces in the target database over network links. This greatly reduces the overhead of taking an export dump of tablespace metadata, using FTP to get it to the target server, and then importing it. Other benefits of Data Pump include the DBMS_DATAPUMP package that facilitates automation, a web interface, and improved restart capability.

TESTING ENVIRONMENT SETUP

Using the above technologies, we can now dig into some of the details related to the moving of data. The details discussed here are for the testing environment used by the authors' in preparation for the production migration:

Attribute	Source Server	Target Server
Operating system	Tru64 5.1b -- A little endian OS	HP-UX 11.11 – A big endian OS
CPU's	2 at 667 MHz	2 at 1 GHz
RAM	2 GB	32 GB
Database size	15 GB	None before migration,
Database version	10.1.0.3	10.1.0.3
Database global name	DTEST	DTEST1
Database mount point(s)	/plad02/oradata/dtest, /olap003/oradata/dtest /dmex002/oradata/dtest, /plad03/oradata/dtest	/ona7/appl/oracle/oradata/dtest

For production environments, the process is identical except the sizes are larger, there are more filesystems to mount, more data files to move, etc. Thus we will discuss the 15 GB test environment in detail to keep the discussion manageable but will address the performance issues that come with production volumes later in the discussion.

The following were the OS level preparations made prior to testing the migration process. The goal here is to allow a target-side conversion to be run over NFS. The setup is shown in Figure 2 and is:

1) Establish a gigabit connection between the source server and the target server.

2) Mount via NFS over the gigabit connection the datafiles from the source server to the target server.
   a) Exports from the source for our test were:
      ```
      /plad02/oradata/dtest, /olap003/oradata/dtest,
      /dmex002/oradata/dtest, /plad03/oradata/dtest
      ```
   b) Mounts on the target for our test were:
      ```
      /pladt_plad02/oradata/dtest, /pladt_olap003/oradata/dtest,
      /pladt_dmex002/oradata/dtest, /pladt_plad03/oradata/dtest
      ```

The largest advantage of a design based on NFS is that it eliminates the need for having temporary space on the target side equal to the amount of data being moved. That is, if an ftp approach is taken, the 7 TB's of data would have to be moved via ftp to a 7 TB holding area on the target and the 7 TB would then have to be endian-converted to the permanent 7 TB location. RMAN cannot do endian conversion "in place." Most environments simply do not have an extra 7 TB ready to use as temporary space, so the ftp to temporary space is often not feasible. The other advantage is that it makes the process a single step. If a mechanism such as ftp were used, moving the files would be one step and the conversion would be the second.

The single largest negative of the NFS approach is that the NFS protocol has high overhead. Where ftp can be well optimized for network throughput using techniques like jumbo MTU and increasing the TCP window size, NFS offers less opportunity for optimization with such methods. Despite this, we recommend the use of NFS because it simplifies the overall design. NFS is a reliable technology, implementation is very straight forward, and as noted, produces a one-step process.

We even considered an approach such as doing ftp into named pipes feeding RMAN processes. This could potentially allow the use of ftp in a single-step process with no need for temporary storage, but that approach was deemed too complex. We felt that the simple and reliable nature of the NFS solution produced the design that offered the greatest chance for success, and decided to address the NFS overhead through adding network interface cards (NIC's) as detailed later in this discussion.

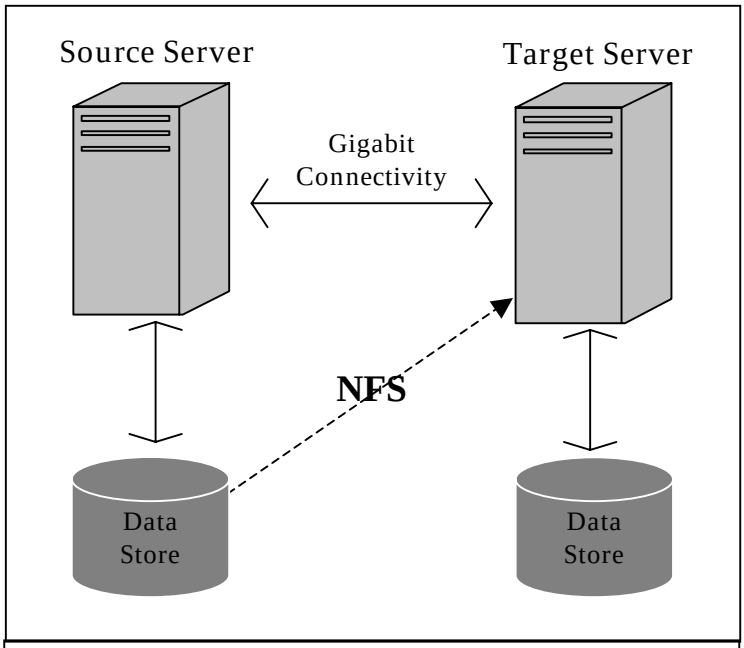

Figure 2: NFS used to present source storage to target server

DETAILED MIGRATION TEST

These are the steps required. Substitute the particulars of your environment for the particulars of the test environment being used for the current discussion. These fall into two major categories: preparation and execution.

Note: The procedure for transporting tablespaces is documented at length in Oracle Database Administrator's Guide (5).

TRANSPORT PREPARATION

1) Create the target database, called DTEST1 here, as follows:
   a) Use the same character set (US7ASCII) and national character set (AL16UTF16) as the source database.
   b) Use a dictionary managed system tablespace; a locally managed system tablespace will not allow dictionary managed user tablespaces to be plugged in or created.

2) Create a partial clone of the source database, called CLONE here, consisting of only the system, sysaux and undo tablespaces.

3) Drop entries for all tablespaces other than system, sysaux, and undo using the DROP TABLESPACE command in the CLONE database. This is done to prevent the loss of any sysaux tablespace entries such as database control metrics or database workload repository information.

4) Perform a full database export dump of the CLONE database:

```
exp system/***** full=y file=clone.dmp log=clone.log
```

5) Prepare a tablespace creation script for all tablespaces in the source database, other than system, sysaux, and undo, including temporary tablespaces. This will be used in the target database.
   a) For non-temporary tablespaces, ensure that the size is 25 MB.
   b) For temporary tablespaces, ensure that the size is the same as in the source database.
   c) The mount points used are as they appear on the target since that is where this script will be run.

6) Run the script just prepared against the target server, which is DTEST1 in our discussion. This ensures that no users were left out and that any datafiles in the target belonging to other database are not being overwritten during the import.

7) Determine the Oracle Directories and their paths in the source database, DTEST, with this query:

```
select directory_name,
       directory_path
  from dba_directories
```

8) Create symbolic links with the directory path structure pointing to existing mounts in the target server. This is done to prevent the failure of the directory creation while doing a full database import in the next step.

9) FTP the clone.dmp file created above to the target server and import it into the target database, DTEST1. This creates all users, and objects other than tables and indexes. Most PL/SQL objects will likely be invalid due to the absence of tables and indexes.

```
imp system/******
full=y file=clone.dmp
log=clone.log ignore=y
```

10) Drop the tablespaces in the target database except the sysaux, system, undo and temporary tablespaces.

TRANSPORT EXECUTION

1) Determine the endian of the source and target databases as shown in Query 2. The query results are shown here for both the source and target.

2) Check for the Transportable Tablespace violations for the tablespaces being transported, as shown here:

```
execute dbms_tts.transport_set_check(
  'INDX,USERS,INDX,TEST1,TEST2,SEMCI,ARS2,DM_AUTO,OEMREP',
  true
);
PL/SQL procedure successfully completed.

select * from transport_set_violations;
Sys owned object MIKE1 in tablespace USERS not allowed in pluggable set
```

3) Remove all violations. Some of the violations may include materialized views. These should be dropped and recreated manually after the tablespaces are transported.

4) Create any directories needed for XTTS log files:

```
create directory exp_dir
    as '/home/oracle/mig';
```

```
SELECT D.PLATFORM_NAME, ENDIAN_FORMAT
FROM V$TRANSPORTABLE_PLATFORM TP, V$DATABASE d
WHERE TP.PLATFORM_NAME = D.PLATFORM_NAME


PLATFORM_NAME                            ENDIAN_FORMAT
---------------------------------------- --------------
HP Tru64 UNIX                            Little

PLATFORM_NAME                            ENDIAN_FORMAT
---------------------------------------- --------------
HP-UX (64-bit)                           Big
```

Query 2: Determining source and target endian

5) Create a database link while connected as system which will allow metadata from the source database to be used by the import dump utility:

```
create database link 'dtest.world' connect to system
identified by test01 using 'dtest.world';
```

6) Run an RMAN on the datafiles as shown in Listing 1 to convert the endian format.
   a) This will make a copy of datafiles from the NFS mount points to the local location '/ona7/appl/oracle/oradata/dtest'.
   b) In this example, the endian conversion is performed using the RMAN CONVERT command on the target system
   c) The degree of parallelism used here was 3. Note the advantage of doing so as shown in Table 1.

7) Perform an import dump (impdp) to import the metadata for the tablespaces via the dtest1.world link as shown in Listing 2

8) Copy the objects such as external tables and bfiles to the target location under the Oracle Directory specified.

9) Run utlrp.sql script to recompile the invalid objects.

10) Run validation checks.

11) Make the tablespaces read/write for both the source and target database.

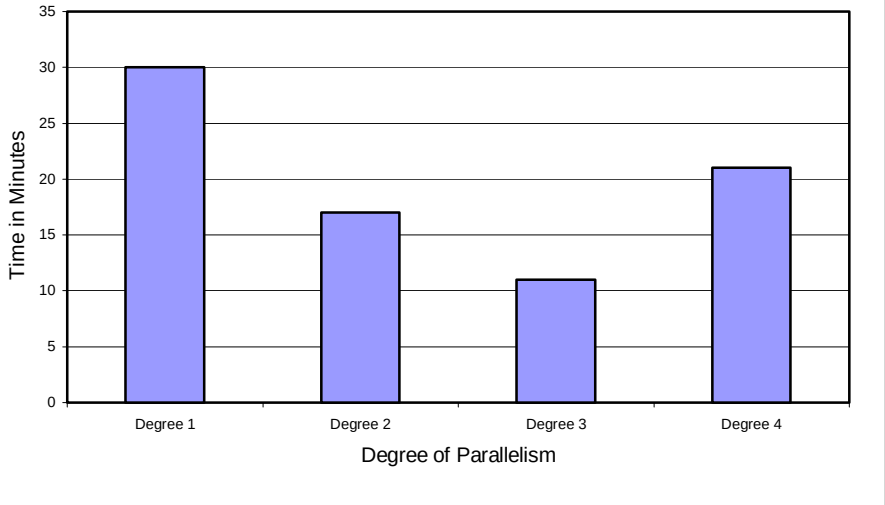

Table 1: RMAN times by degree of parallelism for a 15 GB move

```
run {
allocate channel d1 device type disk;
allocate channel d2 device type disk;
allocate channel d3 device type disk;
CONVERT DATAFILE
  '/pladt_dmex002/oradata/dtest/users01.dbf',
  '/pladt_dmex002/oradata/dtest/indx01.dbf',
  '/pladt_plad02/oradata/dtest/ss01.dbf',
  '/pladt_plad03/oradata/dtest/ss02.dbf',
  '/pladt_olap003/oradata/dtest/ss03.dbf',
  '/pladt_plad03/oradata/dtest/oemrep01.dbf',
  '/pladt_dmex002/oradata/dtest/test301.dbf',
  '/pladt_dmex002/oradata/dtest/test102.dbf',
  '/pladt_dmex002/oradata/dtest/test101.dbf',
  '/pladt_dmex002/oradata/dtest/test201.dbf',
  '/pladt_dmex002/oradata/dtest/semci001.dbf',
  '/pladt_dmex002/oradata/dtest/ars2001.dbf',
  '/pladt_dmex002/oradata/dtest/dm_auto001.dbf'
FROM PLATFORM 'HP Tru64 UNIX'
DB_FILE_NAME_CONVERT
  '/pladt_plad02',
  '/ona7/appl/oracle',
  '/pladt_plad03',
  '/ona7/appl/oracle',
  '/pladt_olap003',
  '/ona7/appl/oracle',
  '/pladt_dmex002',
  '/ona7/appl/oracle';
}
```

Listing 1: RMAN code for endian conversion

```
impdp system/test
DIRECTORY=exp_dir
NETWORK_LINK=dtest.world TRANSPORT_TABLESPACES=
  INDX,TEST1,TEST2,SEMCI,ARS2,DM_AUTO,OEMREP
TRANSPORT_FULL_CHECK=n TRANSPORT_DATAFILES=
/ona7/appl/oracle/oradata/dtest/ars2001.dbf,
/ona7/appl/oracle/oradata/dtest/oemrep01.dbf,
/ona7/appl/oracle/oradata/dtest/ss02.dbf,
/ona7/appl/oracle/oradata/dtest/test102.dbf,
/ona7/appl/oracle/oradata/dtest/users01.dbf,
/ona7/appl/oracle/oradata/dtest/dm_auto001.dbf,
/ona7/appl/oracle/oradata/dtest/semci001.dbf,
/ona7/appl/oracle/oradata/dtest/ss03.dbf,
/ona7/appl/oracle/oradata/dtest/test201.dbf,
/ona7/appl/oracle/oradata/dtest/indx01.dbf,
/ona7/appl/oracle/oradata/dtest/ss01.dbf,
/ona7/appl/oracle/oradata/dtest/test101.dbf,
/ona7/appl/oracle/oradata/dtest/test301.dbf
```

Listing 2: Importing the metadata information with import dump

RESULTS OF THE TEST RUN

The results of this test showed that we were able to migrate a 15 GB just under 11 minutes with a parallelism of degree 3. This did not include all the manual efforts beyond the actual data move itself, but such manual activities can be highly automated to minimize them during the actual migration of production systems. In determining the degree of parallelism that worked best for RMAN in our environment, degrees of 2, 3, and 4 were tested, and the best performance was achieved at degree 3 as shown in Table 1.

EXTRAPOLATING TO PRODUCTION

Based upon the test results, we determined we would need four gigabit cards for each of the source and target servers connected through four private network connections: 15 GB in 11 minutes is 7 TB in 93 hours, and with four NIC's, that would be 7 TB in 23.25 hours, as needed.

One concern was that parallelism often does not fully scale. In our environment, however, it scaled very nicely. Our source database had over 100 file systems. This allowed us to run dozens of processes with each hitting only one or a few file systems at a time. This prevented I/O contention, but more importantly we found that it actually increased throughput per NIC because all those processes running against disks with no I/O contention put maximum pressure on the 4 NIC's. Another concern raised was that the bus or drive controllers could become an issue. These turned out to be non-issues for us but should be verified on a case by case basis.

A generalized diagram of a large scale environment is shown in Figure 3. In this case, 100 source file systems and 100 target file systems are shown, along with four gigabit Ethernet cards over which the NFS traffic travels. A key point to note is that the file systems must be logically grouped to go over specific NFS mounts. This must be done at the time the NFS mounts are defined, not at the time of the script execution. For example, the 25 file systems /pl26 to /pl50 will have to be NFS mounted specifically over the same NIC card to the target server mount points /tl26 to /tl50. Once that is done, the scripts that actually do the execution simply reference /tl26 to /tl50.

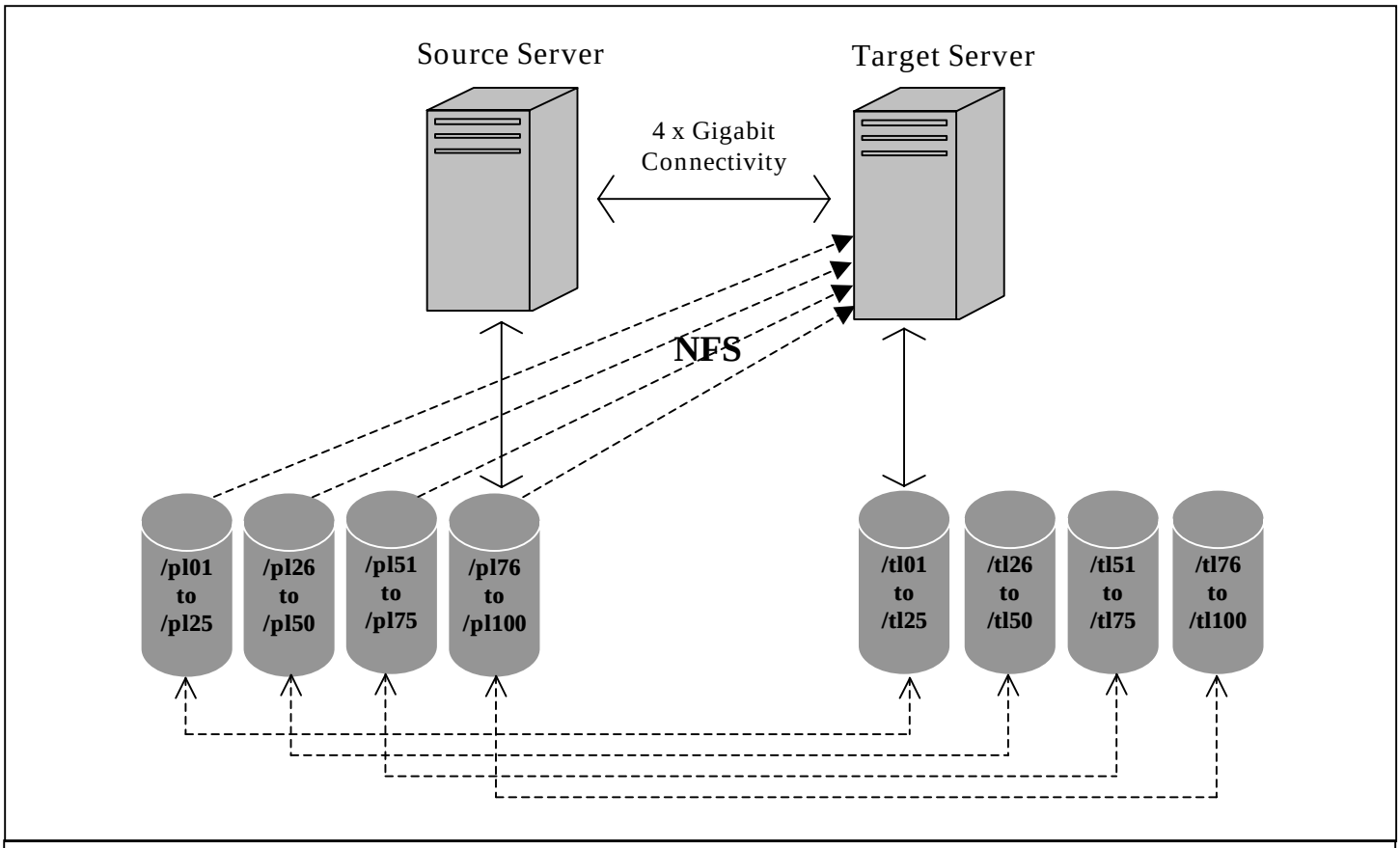

Figure 3: Extrapolation of the testing construct to the full-size production environment

NON-DATABASE CONSIDERATIONS FOR MASS MOVES

From our discussions so far, it is clear that Oracle provides the database technologies needed to move large amounts of data in a timely and error-free manner. Understanding the non-database reasons that would drive the need for such large-scale migrations is also important. That is, we have shown ***that*** it ***can*** be done, but it is important to also see ***why*** it ***must*** be done in appropriate situations. These reasons come from two major areas: the application architecture, and the value to the business.

APPLICATION ARCHITECTURE

In data warehousing environments, is it valuable to be able to relate data across systems. Examples from the authors' environment include an identifier that is unique for every load that allows the data to be clearly identified throughout the warehouse, and a set of global keys that allow data from fairly disparate systems to be related to each other. This ability to have global identification has tremendous value from both an application and business perspective, but it causes a very complex web of system coupling this is essentially impossible to take apart. Thus, finding ways to just pick up and move this web of complexity as a unit is valuable. Mass migrations using Oracle tools as described above facilitate this.

Another inevitability in a warehouse is the tendency of systems to join against each other. This stems from one of the basic requirements of a warehouse, to be the integration point for the organization rather than having information in silos (which is how it tends to exist in the organization at large). This need to integrate silos leads to many joins at the application layer between the various sources. What would happen if we attempted to move systems in pieces with all these joins occurring? Many of them would then cross between the old and new instances.

Such cross-instance joins tend to have horrible performance. If it were possible to have all the tables that participate in any join be in either the old or new instance exclusively, the problem could be solved with a driving-side hint. Trouble is, the joins instead tend to get their joins mixed between instances. The solution is simply to move the database as a unit.

Global keys and cross-instance joins are just two problems to consider, and every environment has the potential to create its own unique challenges, but once the decision is made to move the database as a unit, one other major problem often comes up: most shops run their back-end systems on the same server as the database. That is, from the inception of most batch-intensive shops like data warehouses, a default assumption has been that the system would be on the same server as the database and thus the systems can do such things as easily access the database's utl_file_dir location, bypass the listener, etc. Trouble is, if your shop has run for years on this assumption and built many systems on this assumption, suddenly moving the database can cause a problem.

The immediate thought may be to move all the systems at the same time as the database. If that is possible, do it! Such a "lift and load" approach is great if it can be done, but it is frequently not viable because moving systems is generally vastly more difficult than moving the database. Essentially, Oracle has now provided "silver bullet" tools for massive data moves, but no silver-bullet solutions exist to move dozens or hundreds of systems with complexities like those discussed above. The solution to this is to move the database first then move the systems over time. To facilitate this process of moving the database without systems, only a few minor changes are needed. These are:

1) Utilize the TWO_TASK environmental variable to allow systems to find remote databases.
2) If you use OS authentication, enable remote_os_authent so that the database will allow the systems to connect. Use a database trigger if enabling remote authentication creates too big a security hole.
3) Create a writable NFS mount back to the old server for all directories listed in the utl_file_dir database parameter.
4) Create a writable NFS mount back to the old server for all directories used for external file operations.

Once these are done, the database can be moved without the systems, and the systems can be migrated over time.

VALUE TO THE BUSINESS

Large database moves generally occur in a migration context that includes the purchase of new hardware. In a data warehouse such purchases can easily run into the millions of dollars. Every day that this new hardware goes unutilized costs the company thousands of dollars in depreciation and maintenance costs while yielding no value to the business. To minimize this, the goal is to get as many of the systems or as much of each system off the old hardware and onto the new as soon as possible.

As discussed, application architecture is often complex and very intertwined, so moving all the systems will likely be a painful, time-consuming process. However, as we have elaborated in detail here, moving large databases as a unit, even while leaving the systems in place, is quite feasible. If such an approach it taken, it gets the databases off the old hardware and on the new quickly. This means much of the old storage can be shut down, ending the maintenance costs there, and the new hardware is being utilized, thus providing ROI to the business on their major hardware purchase.

This approach reduces schedule risk by breaking the two hardest parts of the project into two mostly-separate pieces so that the project can have more of a phased approach. The first phase is to move the databases without the systems. The next phase is to move the systems now that the databases are safely on the new hardware. Phasing projects has a long history of being more successful than any approach that lumps things together, and good project managers will likely jump on this opportunity to take the two hardest parts of their project and separate them into discrete units of work.

Another advantage from a business perspective is that of minimized risk. A major migration by its nature is a very risky endeavor, and managers at all levels will be looking for ways to mitigate risks at any point. The approach of moving databases as a unit using Oracle-native tools helps do this. The business should be made aware that while moving the database as a unit might sound risky, it is far less problematic than moving it in many pieces over many months because while each small piece clearly has less risk than moving the whole database, the cumulative effect of moving many pieces over a large span of time produces a larger total body of risk in the end. Once the single-unit approach is accepted, making it clear that all Oracle-native tools will be used will also help sell the approach because using tools from a single vendor prevents integration issues when setting things up and finger-pointing if vendor tools do not work together as planned.

CONCLUSION

Moving large databases as a unit has the positive effect of closing a major line item early in the project. As scary as it might sound to try to move a 7 TB database across platforms with an endian reversal in a weekend--the fact is that it can be done. Give yourself a few days early in the week after the upgrade weekend to deal with fallout, but know that by about Thursday--it's done! The handful of fallout issues will have been addressed, the project manager will have marked the task 100% complete, the new hardware will be getting well utilized, and as you're eating lunch on Friday with a grateful manager paying the bill, you'll be thinking..."That really wasn't so hard after all!"

REFERENCES

1. Oracle Cross Platform Transportable Tablespace
http://www.oracle.com/technology/deploy/availability/htdocs/xtts.htm

2. Oracle Recovery Manager – RMAN
http://www.oracle.com/technology/deploy/availability/htdocs/rman_overview.htm

3. Data Pump
http://www.oracle.com/technology/products/database/utilities/htdocs/data_pump_overview.html

4. RMAN CONVERT command
http://download-west.oracle.com/docs/cd/B14117_01/server.101/b10770/rcmsynta18.htm#RCMRF200

5. Using Transportable Tablespace
http://download-west.oracle.com/docs/cd/B14117_01/server.101/b10739/tspaces.htm - ADMIN01101