

An Oracle White Paper
February 2009

Data Pump in Oracle[®] Database 11g: Foundation for Ultra High-Speed Data Movement Utilities

Introduction	1
Data Pump Overview	2
Data Pump Architecture	3
Master Table	3
Process Structure	3
Data Movement	5
Metadata Movement	6
Interprocess Communication	6
File Management	6
Directory Management	6
Major Features	7
Performance	7
Restart	9
Fine-grained Object Selection	9
Monitoring and Estimates	11
Data Pump Clients: expdp and impdp	11
Other Useful Features	12
Original exp and imp	14
Differences Between Data Pump and Original exp/imp	14
Conclusion	15

Introduction

This paper provides an overview of the Data Pump architecture, followed by a description of the main features of Data Pump, some discussion of best practices, and finally a brief comparison of the Data Pump Export and Import utilities to the original Export and Import utilities.

Data Pump Overview

First available in Oracle Database 10g, Data Pump is a fully integrated feature of Oracle Database that enables very high-speed loading and unloading of data and metadata to and from the database. It automatically manages and schedules multiple, parallel streams of loading or unloading for maximum throughput. Data Pump infrastructure is callable through the PL/SQL package `DBMS_DATAPUMP`. Thus, custom data movement utilities can be built using Data Pump. Oracle Database includes three such client utilities:

- Command-line export (`expdp`)
- Command-line import (`impdp`)
- Web-based Oracle Enterprise Manager export/import interface

Data Pump is also the foundation for several other key features in Oracle Database: Automated Workload Repository (AWR), Streams-Based Replication, Logical Standby, Grid, and Transportable Tablespaces. For Streams-Based Replication and Logical Standby, Data Pump quickly generates the initial configuration at the replicated site using Flashback technology to provide a consistent starting point. Initial Grid instantiation is based on Transportable Tablespaces, which relies on Data Pump to move and *hook up* the metadata for the objects defined in the transported tablespace set.

When gathering requirements for Data Pump from large customers, we repeatedly heard, “Time is money. If you do nothing else, make export and import much faster for large amounts of data.” We took that to heart: Data Pump decreases the elapsed time for large export/import operations by *two orders of magnitude* in some data-intensive cases.

In addition to increased performance, our customers enumerated many other requirements. As a result, Data Pump-based export and import clients (`expdp` and `impdp`) support all the features of the original export and import clients (`exp` and `imp`), as well as many new features, such as dump file encryption and compression, checkpoint restart, job size estimation, very flexible, fine-grained object selection, direct loading of one instance from another, detailed job monitoring, and the ability to move individual table partitions using transportable tablespaces.

Data Pump is an integral feature of Oracle Database and therefore is available in all configurations. However, some features of Data Pump such as parallelism are available only in the Enterprise Edition. Dump file encryption is available as part of the Advanced Security Option, and dump file data compression is included in the Advanced Compression Option.

Data Pump Architecture

Oracle Data Pump was written from the ground up with an architecture designed to produce high performance with maximum flexibility. Understanding the architecture of Data Pump will help you take advantage of its speed and features.

Master Table

At the heart of every Data Pump operation is the master table. This is a table created in the schema of the user running a Data Pump job. It is a directory that maintains all details about the job: the current state of every object being exported or imported, the locations of those objects in the dumpfile set, the user-supplied parameters for the job, the status of every worker process, the current set of dump files, restart information, and so on.

During a file-based export job, the master table is built during execution and written to the dumpfile set as the last step. Conversely, loading the master table into the current user's schema is the first step of a file-based import operation, so that the master table can be used to sequence the creation of all objects imported.

The use of the master table is the key to the ability of Data Pump to restart a job in the event of a planned or unplanned job stoppage. Because it maintains the status of every object to be processed by the job, Data Pump knows which objects were currently being worked on, and whether or not those objects were successfully completed.

Process Structure

A Data Pump job comprises several processes. These processes are described in the order of their creation.

Client Process – This is the process that makes calls to the Data Pump API. As mentioned earlier, Oracle Database ships four client utilities of this API. This paper will discuss only the new export/import clients, `expdp` and `impdp`. These have a very similar look and feel to the original `exp` and `imp` clients, but have many more capabilities, as will be described later. Because Data Pump is integrated into Oracle Database, a client is not required once a job is underway. Multiple clients may attach and detach from a job as necessary for monitoring and control.

Shadow Process – This is the standard Oracle shadow (or foreground) process created when a client logs in to Oracle Database. The shadow services Data Pump API requests.¹ Upon receipt of a `DBMS_DATAPUMP.OPEN` request, the shadow process creates the job, which consists primarily of creating the master table, the AdvancedQueuing (AQ) queues used for communication among the various processes, and the master control process. Once a job is running, the main task of the shadow process consists of servicing `GET_STATUS` requests from the client. If the client detaches, the shadow process also goes away.

Master Control Process (MCP) – As the name implies, the MCP controls the execution and sequencing of a Data Pump job. There is one MCP per Data Pump job, maintaining the job state, job description, restart, and dumpfile information in the master table. A job is divided into various phases of metadata and data unloading or loading, and the MCP hands out work requests to the worker processes appropriate for the current phase. The bulk of MCP processing is performed in this work dispatch loop. The MCP also performs central file management duties, maintaining the active dumpfile list and handing out file pieces as requested by processes unloading data or metadata. An MCP has a process name of the form: `<instance>_DMnn_<pid>`.

Worker Process – Upon receipt of a `START_JOB` request, the MCP creates worker processes as needed, according to the value of the `PARALLEL` parameter. The worker processes perform the tasks requested by the MCP (primarily unloading and loading of metadata and data), and maintain the object rows that make up the bulk of the master table. As database objects are unloaded or loaded, these rows are written and updated with the current status of these objects: pending, completed, failed, and so on. The worker processes also maintain *type completion rows*, which describe the type of object currently being worked on: tables, indexes, views, and so on. These types completion rows are used during restart. A worker process has a name of the form: `"*DWnn*"`.

Parallel Query (PQ) Process – If the External Tables data access method is chosen for loading or unloading a table or partition, some parallel query processes are created by the worker process that was given the load or unload assignment, and the worker process then acts as the query coordinator. These are standard parallel execution slaves that exploit the parallel execution architecture of Oracle Database, and enable intra-partition loading and unloading. In Real Application Clusters (RAC), parallel query processes may be created on an instance other

¹ The Data Pump public API is embodied in the PL/SQL package `DBMS_DATAPUMP`. It will not be described in detail here. Full documentation may be found in *Oracle Database Utilities and PL/SQL Packages and Types Reference*.

than that on which the Data Pump job was initiated. All other processes described thus far are created on that initial instance.

Data Movement

Data Pump supports four methods of data movement, each of which has different performance and functional characteristics. In descending order of speed, these four methods are:

- Data File Copying (transportable tablespaces)
- Direct Path load and unload
- External Tables
- Conventional Path

Data Pump will choose the best data movement method for a particular operation. It is also possible for the user to specify an access method using command line parameters.

The fastest method of moving data is by copying the database data files that contain the data without interpretation or altering of the data. This is the method used to move data when transportable mode is specified at export time. There are some restrictions on the use of data file copying. Some types of data, some types of tables, and some types of table organization cannot be moved with this method. For example, tables with encrypted columns cannot be moved using this access method. In addition, the character sets must be identical on both the source and target databases in order to use data file copying.

Direct path and external tables are the two main data access methods provided by Oracle Database 11g. The direct path access method is the faster of the two, but does not support intra-partition parallelism. The external tables access method does support this function, and therefore may be chosen to load or unload a very large table or partition. Each access method also has certain restrictions regarding the use of the other. For example, a table being loaded with active referential constraints or global indexes cannot be loaded using the direct path access method. A table with a column of data type LONG cannot be loaded with the external tables access method. In most cases, you need not be concerned about choosing an access method; the Data Pump job will make the correct choice based on an array of job characteristics. Both methods write to the dumpfile set in a compact, binary stream format that is approximately 15 percent smaller than the original exp data representation.

When neither direct path nor external tables can handle the data to be imported, Data Pump uses a method called conventional path. For example, a table that contains an encrypted column and a LONG column would be imported using conventional path because direct path cannot be used to import encrypted columns and external tables cannot be used to import LONG columns. Data loading with the conventional access method is much slower than the direct path and external tables methods. So, the Data Pump uses this method only when it has no other choice.

Metadata Movement

The Metadata API (DBMS_METADATA) is used by worker processes for all metadata unloading and loading. Unlike the original `exp` function (which stored object definitions as SQL DDL), the Metadata API extracts object definitions from the database, and writes them to the dumpfile set as XML documents. This allows great flexibility to apply XML Stylesheet Language Transformations (XSLTs) when creating the DDL at import time. For example, an object's ownership, storage characteristics, and tablespace residence can be changed easily during import. This robust XML might take up more dumpfile space than the old style SQL DDL, but it provides more flexibility and features. In addition, the `COMPRESSION` parameter can be used to decrease the size of metadata written during a Data Pump export job.

Interprocess Communication

AdvancedQueuing (AQ) is used for communicating among the various Data Pump processes. Each Data Pump job has two queues:

- Command and control queue: All processes (except clients) subscribe to this queue. All API commands, work requests and responses, file requests, and log messages are processed on this queue.
- Status queue: Only shadow processes subscribe to read from this queue. It is used to receive work-in-progress and error messages queued by the MCP. The MCP is the only writer to this queue.

These queues have names of the form: `KUPC${C|S}_<job-unique timestamp>`.

File Management

The file manager is distributed across several parts of the Data Pump job. As mentioned earlier, the actual creation of new files and allocation of file segments is handled centrally within the MCP. However, each worker and parallel query process makes local process requests to the file manager to allocate space, read a file chunk, write to a buffer, or update progress statistics. The local file manager determines if the request can be handled locally and if not, forwards it to the MCP using the command and control queue. Reading file chunks and updating file statistics in the master table are handled locally. Writing to a buffer is typically handled locally, but may result in a request to the MCP for more file space.

Directory Management

Because Oracle background server processes handle all dumpfile set I/O, the operating system *persona* doing the I/O is *oracle*, not the user running the job. This presents a security dilemma because *oracle* is typically a privileged account. Therefore, all directory specifications are made using Oracle directory objects with read/write grants established by the DBA.

For example, the DBA may set up a directory as follows:

```
Create directory dmpdir1 as '/private1/data/dumps';
```

```
Grant read, write on directory dmpdir1 to scott;
```

Then `scott` can specify a dump file on the `expdp` command line as:

```
expdp scott/tiger dumpfile=dmpdir1:scott.dmp
```

If the file size is limited for manageability by the `FILESIZE` parameter, then potentially many dump files could be created. The file manager automatically maintains dumpfile set coherency using a globally unique identifier and other information written into the file headers. An import or SQL file job cannot start until all members of the job's dumpfile set are present in `DUMPFILE` parameter specifications.

Major Features

This section will briefly describe some of the major new features in Data Pump.

Performance

Data Pump export/import operations (`expdp` and `impdp`) are typically much faster than their original `exp` and `imp` counterparts. A single thread of Data Pump's direct path data unload is about twice as fast as the original `exp`. A single thread of Data Pump data load is *15 to 45 times faster* than the original `imp`. In addition, Data Pump operations can be specified with parallel threads of execution². Also note that parallel threads can be dynamically added to and removed from running jobs to tailor the job to the changing execution environment.

During export, when there are two or more worker processes, data and metadata unloading proceed in parallel. Data Pump will also automatically build each index in parallel up to the degree of parallelism of the job. The permanent parallel degree of the index itself remains unchanged from its source value.

I/O bandwidth is most important factor

It is important to make sure there is sufficient I/O bandwidth to handle the number of parallel threads specified. Otherwise, performance can actually degrade with additional parallel threads. Care should be taken to make sure the dumpfile set is located on spindles other than those

² With Oracle Database Enterprise Edition

holding the data files for the instance. Wildcard file support makes it easy to spread the I/O load over multiple spindles. For example, a specification such as the following will create files named `full101.dmp`, `full201.dmp`, `full301.dmp`, `full401.dmp`, `full102.dmp`, `full202.dmp`, `full302.dmp`, and so on, in a round-robin fashion across the four directories pointed to by the four directory objects:

```
Dumpfile=dmpdir1:full1%u.dmp,dmpdir2:full2%u.dmp
```

```
Dumpfile=dmpdir3:full3%u.dmp,dmpdir4:full4%u.dmp
```

Initialization Parameters

Essentially no tuning is required to achieve maximum Data Pump performance. Initialization parameters should be sufficient upon installation. Note the following:

- Make sure the `disk_asynch_io` value remains `TRUE`. It has no effect on those platforms whose file systems already support asynchronous I/O, but a value of `FALSE` can have a significant adverse impact on those platforms that do not.
- Make sure the `db_block_checksum` default value is `FALSE`, but if an integrity issue is being investigated requiring this to be set to `TRUE`, its impact on data loading and unloading would be minimal (less than 5 percent).
- Data Pump's AQ-based communication and the metadata API both require some amount of SGA. Make sure the `streams_pool_size` value is sufficient.

Both the metadata API during export and the worker process during import may execute some fairly long-running queries that have the potential for exhausting rollback segments. This is mostly an issue in jobs affecting many objects. Make sure these are configured sufficiently large. For example, export/import of a database containing 400,000 objects required two rollback segments, each 750 MB in size.

Metadata API

Metadata performance in the Data Pump is about the same as the original `exp` and `imp` clients, but the XML format used by Data Pump makes metadata operations much more flexible and extensible. Small examples where metadata movement is a significant part of the job may thus not show much performance improvement. However, because data movement dominates most real world production environments, most operations see a dramatic overall improvement.

Network Mode

Data Pump supports the ability to load one instance directly from another (network import) and unload a remote instance (network export). Rather than using network pipes, which are not supported on all platforms, network mode uses DB links.

During network import, the Metadata API executes on the remote node, extracting object definitions and sending them to the local instance for creation where the Data Pump job is executing. Data is fetched and loaded using `insert as select` statements such as:

```
Insert into foo (a,b,c,...) select (a,b,c,...) from foo@remote_service_name
```

These statements incorporate hints on both sides to access the direct path engine for maximum performance.

Network export provides the ability to export read-only databases. Data Pump `expdp` cannot run locally on a read-only instance because maintaining the master table, writing messages to queues, and creating external tables all require write operations on the instance. Network export creates the dumpfile set on the instance where the Data Pump job is running and extracts the metadata and data from the remote instance, just as network import does. Data movement in network export is done exclusively by External Tables because `create as select@service` style DML statements are required.

With either network mode operation, network bandwidth is expected to become the bottleneck. Be careful that the parallel setting does not saturate the network. We discovered that many of our customers would implement a sort of ‘network mode’ with the original `exp` and `imp` by exporting into a network pipe and importing out the other end. This overlapped the export and import operations, thus improving elapsed time. Given the dramatic improvement in the performance of file-based Data Pump operations, it is unclear whether or not network mode can provide a significant reduction in elapsed time for instance initialization as it did with the original `exp` and `imp`.

Restart

Most stopped Data Pump jobs can be restarted without loss of data as long as the master table and dumpfile set remain undisturbed while the job is stopped. It does not matter whether the job was stopped voluntarily with a `STOP_JOB` command on the client, or involuntarily due to a system failure, power outage, or other unexpected event. Sufficient context is maintained in the master table to know where to start. A client can attach to a stopped job with the `ATTACH=<job name>` parameter, and then start it with the interactive `START` command.

There can sometimes be an unforeseen, repeating problem with a particular object during import that prevents further progress. The `START=SKIP_CURRENT` command will skip the current object and continue with the next, thus allowing progress to be made.

Fine-grained Object Selection

With the original `exp` and `imp`, the only choices for the user to include or ignore were indexes, triggers, grants, or constraints. A Data Pump job can include or exclude virtually any

type of object and any subset of objects within a type, using the various client parameters available.

Exclude

The **EXCLUDE** parameter allows any database object type to be excluded from an export or import operation. The optional name qualifier allows you even finer selectivity within each object type specified. For example, the following three lines in a parameter file would exclude all functions and procedures, as well as packages with names starting with **PAYROLL**, from the job:

```
Exclude=function
```

```
Exclude=procedure
```

```
Exclude=package:"like 'PAYROLL%'"
```

Include

The **INCLUDE** parameter includes *only* the specified object types and objects in an operation. For example, if the preceding three specifications were **INCLUDE** parameters in a full database export, then only functions, procedures, and packages with names starting with **PAYROLL** would be written to the dumpfile set.

Content

The **CONTENT** parameter allows the user to request that the operation include only metadata, only data, or both. The original **exp** parameter **ROWS=N** was equivalent to specifying **content=metadata_only** in Data Pump, but there was no equivalent in the original **exp** for Data Pump's ability to specify **content=data_only**.

Query

The **QUERY** parameter operates much as it did in the original **exp**, but with two significant enhancements:

It can be qualified with a table name such that it applies only to a specific table.

It can be used during import as well as export.

Sample

The **SAMPLE** parameter is used to specify a subset of data to be exported. The sample percentage specified with this parameter indicates the probability that a block of rows will be included in the export sample. Sampling in this manner will not guarantee referential

integrity, but can be a very useful way to populate a test or development system using a subset of data from a larger database.

Monitoring and Estimates

Another requirement we heard from our customers was to provide better, more detailed monitoring capabilities. In addition to the standard progress and error messages printed by the client and into the log file, the new client interactive command `STATUS` will show detailed job information including overall percent done, the status of each worker process, the current objects being worked on, and the percent job completion for each one. You can also specify a time interval in seconds for an automatic update of detailed status.

The start of every Data Pump export job includes an estimation phase where the approximate amount of all data to be unloaded is determined. The default method for determining this is to estimate the size of a partition by counting the number of blocks currently allocated to it. If tables have been analyzed, statistics can also be used that should provide a more accurate estimate. This serves two purposes:

- You get an idea of how much dumpfile space will be consumed.
- All the information needed to start unloading tables is retrieved and ordered by descending size. This allows the MCP to schedule the unloading of metadata and data in parallel.

The objects retrieved during the estimation phase are called *table data objects* with each representing a partition (or the entire table, if the table is not partitioned).

Because *0 to n* clients may be attached to a running job, you can start a long-running job at work, detach from it, go home, re-attach, and then monitor it throughout the evening.

Data Pump Clients: expdp and impdp

Although the new `expdp` and `impdp` clients retain a similar look and feel to the original `exp` and `imp` clients, 100 percent parameter compatibility was not a goal. Where a concept makes sense for both an export and import operation, we made sure the parameter is the same for both. The new clients also support far greater capabilities:

- Interactive command mode: typing control-C (^C) will invoke the interactive command mode and an `export>` or `import>` prompt will appear. From this prompt, you can request help, obtain detailed job status, change monitoring parameters, dynamically add files (including wildcard specifications) to the job's dumpfile set, stop the job but leave it able to be restarted, kill the job leaving it not able to be restarted, change the degree of parallelism for the job, return to logging mode to continue receiving progress messages, or exit the client and leave the job running.

- All modes of operation are supported: the command-line clients can be used when the `MODE` is full, schema, table, tablespace, or transportable tablespace. Data Pump and its new `expdp` and `impdp` clients are a complete superset of the original `exp` and `imp` functions.
- Flashback technology is supported for exports and imports as of a certain time.
- `WHERE` clause predicates may be applied to individual tables by both `expdp` and `impdp`.
- Privileged users (those with either the `EXP_FULL_DATABASE` or `IMP_FULL_DATABASE` roles) may attach to and control jobs initiated by other users, even if the job is stopped.

Other Useful Features

Data Pump includes several useful new features that extend its power and flexibility far beyond the functionality provided by the original export/import utility.

REMAP_DATA – One common use of Data Pump is to populate test or development systems based on data from a production system. In these cases, it is often useful, and indeed sometimes legally required, that personally identifiable data be obscured so that it is not exposed to those who should not have access to such data.

The `REMAP_DATA` parameter allows you to specify a remap function that takes as a source the original value of the designated column and returns a remapped value that will replace the original value in the dump file. For example, a column of sensitive customer data such as credit card numbers could be replaced with numbers generated by a `REMAP_DATA` function. This would allow the data to retain its essential formatting and processing characteristics without exposing private data to unauthorized personnel.

DDL Transformations – Because object metadata is stored as XML in the dumpfile set, it is easy to apply transformations when DDL is being formed (using XSLT) during import. `impdp` supports several transformations:

- `PARTITION_OPTIONS` specifies how table partitions will be handled in an import operation. It is possible to export one or more partitions of a table. Then, on import, the user can choose to load these partitions exactly as they existed in the original database, merge these partitions into a single table, or promote each partition to be an individual, separate table.
- `REMAP_TABLE` allows the user to specify a table name for a partition that is to be promoted to its own table as part of a transportable partition import operation. This must be used in conjunction with `PARTITION_OPTIONS=departition`.
- `REMAP_TABLESPACE` will change the tablespaces in which object segments are located. It also changes the tablespace name in the tablespace definition.

- **REMAP_SCHEMA** provides the old FROMUSER/TOUSER objects to be moved from one schema to another. This gives the capability to change object ownership.
- **REMAP_DATAFILE** is useful when moving databases across platforms that have different file system semantics. You can also specify through the **TRANSFORM** parameter that storage clauses should not be generated in the DDL. This is useful if the storage characteristics of the target instance are very different from those of the source.

SQL File - `impdp` can also perform a *SQL file* operation. Rather than creating database objects, this merely writes the equivalent DDL to a file in the form of a SQL script almost ready for execution. Only the embedded connect statements are commented out.

TABLE_EXISTS_ACTION – The original `imp` would allow rows to be appended to existing tables if `IGNORE=Y` was specified. The **TABLE_EXISTS_ACTION** parameter for Data Pump `impdp` provides four options:

1. **SKIP** is the default: A table is skipped if it already exists.
2. **APPEND** will append rows if the target table's geometry is compatible. This is the default when the user specifies `CONTENT=DATA_ONLY`.
3. **TRUNCATE** will truncate the table, then load rows from the source if the geometries are compatible and truncation is possible. For example, it is not possible to truncate a table if it is the target of referential constraints.
4. **REPLACE** will drop the existing table, then create and load it from the source.

CONTENT – This parameter, applicable to both clients, allows the movement of `DATA_ONLY`, `METADATA_ONLY`, or `BOTH` (the default).

VERSION – `expdp` supports the **VERSION** parameter, which tells the server-based Data Pump to generate a dumpfile set compatible with the specified version. This will be used to perform downgrades in the future. There is no need (as there was with the original `exp`) to run older versions of Data Pump clients.

Oracle Enterprise Manager – Oracle Enterprise Manager supports a fully functional interface to Data Pump.

Data Pump Views – Data Pump maintains a number of user- and DBA-accessible views to monitor the progress of jobs:

- **DBA_DATAPUMP_JOBS**: This shows a summary of all active Data Pump jobs on the system.
- **USER_DATAPUMP_JOBS**: This shows a summary of the current user's active Data Pump jobs.

- **DBA_DATAPUMP_SESSIONS**: This shows all sessions currently attached to Data Pump jobs.
- **V\$SESSION_LONGOPS**: A row is maintained in the view showing progress on each active Data Pump job. The **OPNAME** column displays the Data Pump job name.

Original exp and imp

Original **exp** is no longer supported for general use with Oracle Database 11g. Original **imp** still ships with Oracle Database 11g in order to support import of legacy dump files.

- The original **imp** will be supported forever and will provide the means to import dump files from earlier releases (release 5.0 and later) that were created with the original **exp**. Original and Data Pump-based dump files are not compatible: Neither client can read dump files created by the other.
- The original **exp** was deprecated in Oracle Database 10g Release 2, and is no longer supported for general use as of Oracle Database 11g. Data Pump **expdp** will be the sole supported means of export, moving forward. New features in Oracle Database 10g and later releases are not supported in the original **exp**. Older versions of **exp** may be used with Oracle Database 11g for downgrade purposes. Beyond Oracle Database 10g, the **expdp VERSION** parameter should be used for downgrade.

Differences Between Data Pump and Original exp/imp

This section highlights some of the main differences you will notice when first running Data Pump-based export and import.

Data Pump is designed for *big* jobs with *lots* of data. This has a number of implications:

- Startup time is longer. All the process and communication infrastructure must be initialized before a job can get underway. This could take about 10 seconds. Also, export start time includes retrieval and ordering of all table data objects so the MCP can start immediately scheduling table unloads.
- Data Pump **expdp** must write the master table to the dumpfile set at the end of a job. Data Pump **impdp** must locate and load the master table, then build its indexes. This should also be about 10 seconds, but if the master table is very large, the index builds at import start time could take longer. The Direct Path access method is used to unload and load the master table.
- Importing a subset of a dumpfile set updates nonpertinent rows from the master table. If the subset is very small compared to the export set, time to perform the required updates could be noticeable.

- Performance of metadata extraction and creation is about equal to the original `exp` and `imp`. It is very difficult to make DDL go faster. The greatest performance improvement in Data Pump occurs when unloading and loading data.
- Uncompressed XML metadata in dump files is about 7 times bigger than the original `exp` DDL, and Data Pump's data stream format is about 15 percent smaller than the original `exp` row/column format. The `COMPRESSION` parameter helps reduce the size of Data Pump metadata, and metadata compression is on by default starting with Oracle Database 10g Release 2
- Data Pump is as resource-intensive as you wish. Reducing elapsed time per job is first and foremost in the design. Data Pump will consume as much CPU, memory, I/O bandwidth, and network bandwidth (in network mode) as your setting of `PARALLEL` will allow.

The user running Data Pump must have sufficient tablespace quota to create the master table.

Be cognizant of rollback configuration in jobs containing many objects.

The progress messages displayed by the clients are different than those displayed by the original `exp` and `imp` clients, but still reflect the current object type. Also note that 'Already exists' errors are flagged as such and included in the total errors count issued at the end of the job.

A log file is generated by default with a name `export.log` or `import.log`. This can be overridden with the `LOGFILE` parameter. The original `exp` and `imp` do not generate a log file by default.

Conclusion

Data Pump is a callable feature in Oracle Database 11g that provides very high-speed loading and unloading of data and metadata. Command line export and import clients, `expdp` and `impdp`, that fully exploit the Data Pump infrastructure, are also provided with Oracle Database. They are implemented as complete supersets of the original `exp` and `imp`, and will replace these legacy clients going forward.



Data Pump in Oracle® Database 11g:
Foundation for Ultra High-Speed Data
Movement Utilities
February 2009
Author: Roy F. Swonger
Contributing Authors: George Claborn, William
Fisher, Carol Palmer, Jim Stenoish

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com



Oracle is committed to developing practices and products that help protect the environment

Copyright © 2009, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.