ORACLE
FUSION MIDDLEWARE

An Oracle White Paper
February 2014

Oracle Data Integrator Performance Guide

ORACLE

Executive Overview 2
INTRODUCTION 3
UNDERSTANDING E-LT 3
ORACLE DATA INTEGRATOR ARCHITECTURE AT RUN-TIME 4
Sources, Targets, Staging Area 4
An E-LT Process 5
OPTIMIZING MAPPINGS 6
Reducing Data Movement 6
Strategies for Loading Data from Sources to the Staging Area 7
Data Flowing through the Agent 7
Data Flowing through Loaders 8
Specific Methods 8
Choosing the Strategy for the Data Flows 8
Locating Joins Optimally 9
Strategies for Integrating Data and Checking Integrity 9
Customizing the Strategies 9
RUN-TIME OPTIMIZATION 10
Agent Location 10
Load Balancing 11
CONCLUSION 12
Appendix A: Mapping Loading Strategies 12
Appendix B: IKM Strategies with Staging on Target 12
Appendix C: IKM Strategies with Staging Different from Target 12

Executive Overview

Solving a data integration problem with Oracle Data Integrator (ODI) will invariably raise questions about how to best achieve performance and scalability. This document is intended to answer these performance and scalability questions, and describe optimization solutions that are appropriate for Oracle Data Integrator 12c.

The most common question Oracle receives about Oracle Data Integrator's performance is, "how fast is it?" Clearly, this is a subjective question that varies considerably based on environment, complexity, optimizations, etc. To best answer this question simply, Oracle Data Integrator is usually the fastest ETL choice a customer could make. This is largely due to the E-LT architecture of Oracle Data Integrator, but also because Oracle Data Integrator is capable of leveraging native technology optimizations for load, unload and complex set-based transformations.

Throughout this document we will continue to illustrate performance and optimization through customer use cases and Oracle Data Integrator case studies.

Like any powerful enterprise software platform, Oracle Data Integrator can be tuned to meet many specific requirements. A substantial portion of this document is dedicated to explaining when to tune, how to tune, and where you would want to tune Oracle Data Integrator to achieve specific results. We will also describe the advantages of ODI's E-LT approach and provide detailed guidance on various loading strategies.

INTRODUCTION

Oracle Data Integrator has a number of features for optimizing performance and scalability. First and foremost, the E-LT architecture itself is the key feature. Unlike the "Pushdown" features introduced by IBM or Informatica, the Oracle Data Integrator E-LT architecture makes use of native optimized code for any technology it supports – this means that Oracle Data Integrator automatically optimizes databases or Big Data workloads for you, regardless of whether you are using Oracle database, Exadata, Hadoop, Teradata, Sybase, Microsoft SQL Server, or any other popular technology. Oracle Data Integrator can also make use of other native utilities like bulk loaders, un-loaders and other data manipulation APIs - no other ETL tool is as optimized for databases and Big Data platforms.

Oracle Data Integrator can be configured with Parallel Agents for load balancing. This is an effective strategy for distributing transaction volume, but not necessarily for speeding transformations. More on this topic appears later in the paper, but for now it is important to make the distinction between "engine-centric" ETL tools, like IBM DataStage and Informatica PowerCenter for example, where parallel processing is a requirement to speed transformations that are accomplished in a row-by-row manner. In contrast, Oracle Data Integrator uses the benefits of set-based transformation processing in any technology to achieve unparalleled transformation speed.

UNDERSTANDING E-LT

Simply said, E-LT means that instead of having the Oracle Data Integrator run-time agent (or any component from Oracle Data Integrator) performing the transformation work; the work is done by the engines hosting the source or target data. Oracle Data Integrator generates native high-performance code for these engines, and orchestrates its execution. As a consequence, the transformation capabilities and scalability of Oracle Data Integrator is equivalent to the engines themselves. DBMS and Big Data engines are the most heavily optimized, and reliable, data manipulation platforms in the world, and with each release, these platforms only continue to get better.

ORACLE DATA INTEGRATOR ARCHITECTURE AT RUN-TIME

The Oracle Data Integrator run-time architecture is a very simple yet powerful and flexible solution. Oracle Data Integrator consists of one or more Java agents, which can run on Source, Target, or dedicated hardware. Each Oracle Data Integrator agent uses an Oracle Data Integrator Repository, which itself can use a shared or dedicated physical database repository. Since the Oracle Data Integrator run-time agent is so lightweight, it can be deployed very near, or even on the Source and Target systems (Hadoop, Databases, Legacy, Files, etc) ensuring a very fast connection and highly optimized run-time. Since each Agent uses a Repository for metadata storage, it is typically advisable to locate the Repository in a common physical location, or a location that has high-speed network accessibility to the Oracle Data Integrator agent JVM.

More information on Oracle Data Integrator's architecture can be found in the *Oracle Data Integrator 12c Architecture Overview*, a whitepaper located on the Oracle Data Integrator Oracle Technology Network webpage.

Sources, Targets, Staging Area

When designing a mapping in Oracle Data Integrator, three data engines are to be taken into consideration: the source(s), the target(s), and the staging area.

The staging area is a separate area where Oracle Data Integrator creates its temporary objects and executes most of the transformations. This staging area is by default located in the target engine, but may be located in the source engine, or in a third (not a source or a target) engine. The default is the target engine as in many traditional use cases including Data Warehousing for example, the power lies in the target DBMS.

Reasons for changing the staging area from the default target location might include any of the following:

- The source and/or target do not have any transformation capabilities. Technologies such as File or JMS Queues and Topics or LDAP do not have dedicated engines and cannot support a staging area.
- The source and target do not have sufficient transformation capabilities. For example, if you want to benefit from transformations available only on Oracle while moving data from Microsoft SQL Server to a Flat File, you would move the staging area to the Oracle database.
- The source(s) and/or target cannot scale for transformation, for example, an OLTP database that would not allow extra overhead from an ETL process. The staging area should be moved in that case on the source or another engine.

- The location of the source and target cannot be changed. The staging area is critical to transformation capabilities. It is also key for performance when designing mappings.

For best overall performance, general guidance should be to perform staging, joins and transformations on the server which has the largest number of CPUs available.

An E-LT Process

A typical E-LT process runs in the following order:

1. Source Data Sets are loaded from the source tables into temporary loading tables created in the staging area. Transformations such as joins or filters running on the source are executed when reading data from the source tables, for example using a SELECT statement when the source is an RDBMS.
2. Data is merged within the staging area from the loading tables into an integration table. Transformations running in the staging area occur at the time when source data is read from the temporary loading tables. The integration table is similar in structure to the target.
3. Data Integrity is checked on this integration table and specific strategies (update, slowly changing dimensions) involving both the target and the integration table occur at that time.
4. Data is integrated from the integration table to the target table.

The steps for integration vary from one mapping to another. For example, if source and target data is located in the same server, step 1 never needs to occur, and data is directly merged from the source tables to the integration table.

As the process above indicates, having the staging area on the target makes the 3 last steps execute within the same target engine, resulting in better performance and fewer stopping and failure points for this data flow.

Since a mapping typically consists of moving data from several source servers to a target server, and then performing transformations within those servers, two key environmental parameters must be taken into account when designing an optimized mapping:

1. CPU availability on the servers
2. Network transfer capabilities

Optimal mapping design is guided by a compromise between the limitations in terms of server capabilities, CPU and network availability.

OPTIMIZING MAPPINGS

There are two key facts to understand E-LT performance:

One of the most time-consuming operations in Data Integration is moving data between servers/machines/applications (ie: network latency). The second most time consuming operation is joining data sets (ie: computational overhead). Therefore, the goals for optimal performance in Oracle Data Integrator design are:

1. Reduce the data flows between servers
    a. Select the best strategy for reduced network latency
2. Locate large joins optimally
    a. Select the appropriate integration/data integrity strategy

By bearing in mind these key considerations while designing mappings, load plans and packages within Oracle Data Integrator, the tooling will enable you to dramatically improve performance relative to engine-based ETL tool capabilities.

The following sections explore these options at a deeper level.

Reducing Data Movement

Reducing the movement of data can be achieved using several methods:

Location of the staging area: Having the staging area located on the target server is usually the most optimal situation as the data volume is typically smaller. However, there are some cases where moving the staging area to one of the source servers drastically reduces the volumes of data to transfer. For example, if a mapping aggregates a large amount of source data to generate a small data flow to a remote target, then you should consider locating the staging area on the source.

Execution location of transformations: You have the choice to execute any of the transformations on the source, target, or staging area. You should select the execution location for the transformations according to the engine's functional capabilities and CPU speed, but also in order to reduce the volume of data flow.

For example, consider the following:

- When filtering source data, it is recommended to execute the filters on the source servers to reduce the data flow from the source to the staging area.
- When joining source tables, if the expected source set resulting from the join is smaller than the sum of the two sources, the join should be performed on the source. If the expected source set after the join is larger than the sum of the two sources, the join should be performed on the staging area, for example in the case of a cross join.

Changed Data Capture is another technique that can be used to reduce dramatically the volume of source data by reducing the overall incoming flow of data to the changed data only. This technique also enables more real-time extraction and consumption of the data.

More information on Oracle Data Integrator's Changed Data Capture framework and integration with Oracle GoldenGate can be found on the Oracle Data Integrator Oracle Technology Network webpage. Various blogs are also available on the subject.

Strategies for Loading Data from Sources to the Staging Area

The methods used to move a data set from one server to another are determined by the loading Knowledge Module selected for the source data set.

Oracle Data Integrator comes with a large number of Knowledge Modules. Three main methods are used to transfer data from one server to another.

1. Data flowing through the agent
2. Loaders
3. Specific methods

These methods can be combined within the same mapping for extracting data from different sources. In addition, Oracle Data Integrator 12c can perform extract operations in parallel.

Data Flowing through the Agent

When data flows through the agent, it is usually read on a source connection (typically via JDBC) by batches of records, and written to the target by batches. The following parameters can help you tune this data flow:

- *Array Fetch* in the source data server definition. This parameter defines the size of the batches of records read from the source at a time and stored within the agent.
- *Batch Update* in the target data server definition. This parameter defines the size of the batches of records written to the target.

Big Array Fetch/Batch Update values have the following consequences:
The same number of rows will be read in fewer batches, thus reducing the network overhead caused by the client/server communication. This is useful when the client/server protocols are very verbose. Another consequence of this approach is that the agent stores a bigger amount of data, and requires more RAM.

The Batch Update/Array Fetch configuration is a compromise between network and agent overhead. With a highly available network, you can keep low values (<30, for example). With a poor network, you can use larger values (100 and more), or you can consider using another method, for example using un-loaders/loaders and low bandwidth protocol such as FTP. Array Fetch/Batch Update typical values are between 30 and 500.

Data Flowing through Loaders

When loaders are used to transfer data, the agent delegates the work to external loading programs such as Oracle's SQL*Loader, Oracle Loader for Hadoop, SQL Server's BCP, etc.

Typically, data is extracted by the source system native loader to a file, and then this file is copied to the target machine and then loaded using the target native loader.

Such calls require the loader to be installed on the machine running the agent. It is also possible to use an ODI built-in un-loader command called OdiSqlUnload.

Loaders usually provide the best performance to extract from/insert to a database with large, even huge volumes of data. The knowledge modules using these loaders usually implement loader specific features for optimal speed.

Specific Methods

The methods include engine-specific loading methods, including:

- Server to Server communication through technologies similar to Oracle's Database links or SQL Server's Linked Servers.
- File to Table loading specific methods such as Oracle External Tables.
- Creating AS/400 DDM Files and loading data through a CPYF command.
- Using an Unloader/Loader method through UNIX pipes instead of files for very large volumes.

Choosing the Strategy for the Data Flows

The choice of a loading strategy is usually restricted by the technical limitations:

- Can you create a temporary file on the disk?
- Do you have access to the loaders on the machine running the agent?
- Will the DBA allow creation of DB Links?
- What is the bandwidth?
- How much CPU/RAM can be reasonably used by the agent?

Usually, the recommendation to achieve optimal performance is to use the Knowledge Modules that leverage most of the specific native features of both the source and technologies. For example, using a SQL to SQL (data flowing through the agent) strategy from Oracle to Oracle is not ideal, since there are methods such as DB Links or External Tables that are several times faster.

Appendix A provides an overview of the performance of the different loading strategies used within a Mapping. Each strategy is rated from the less efficient (*) to the most efficient (*****).

Locating Joins Optimally

Joins are the second most time consuming activity in data integration. Join capabilities vary from one engine to another. It is critical to locate the joins on the server with optimal capabilities and the most of the CPU available.

Having the staging area on a server with the most available CPU, and joins running on this staging area is usually the best choice.

Because Oracle Data Integrator generates native optimized code for each of the joins involved in a transformation, analyzing the underlying engine execution plans can help tune a mapping for better performance.

Strategies for Integrating Data and Checking Integrity

The number of operations required for integration strategies (IKM) is directly related to the complexity of the strategies. Similarly, the number of constraints checked in a data integrity process (CKM) adds more operations and therefore slows down the overall integration process. These are all processes and strategies orchestrated by the Knowledge Modules.

It is recommended to avoid using useless checks when working with large volumes, and to avoid using unnecessarily complex integration knowledge modules.
Common examples of unnecessary operations:

- Checking 3 times in an integration chain the same data that has not changed.
- Using an *Update* knowledge module with DELETE_ALL and INSERT options selected. Using a *control append* knowledge module instead would have the same result with better performance.

Appendix B and Appendix C provide an overview of the performance of the different integration strategies used within a Mapping. Each strategy is rated from the less efficient (*) to the most efficient (*****).

Customizing the Strategies

Oracle Data Integrator's Knowledge Module framework allows users to customize existing knowledge modules, as well as to create new knowledge modules for additional flexibility to one's environment.

When a Knowledge Module does not provide optimal performances for a given situation, it is important to be able to alter the Knowledge Module to optimize the appropriate strategy.

Such an optimization should occur in three steps:

1. Identify the non-performing task
2. Determine the required performance tuning improvement
3. Implement and test the improvement

Identify a non performing task in an integration process is very straightforward. Oracle Data Integrator's Operator navigator provides detailed information on the sessions' execution time. This information can be drilled down to each task. Similarly, this information can be viewed through Oracle Enterprise Manager Cloud Control if that is being leveraged.

Operator also provides a view of the generated and executed code for each task. When a task is identified with performance issues, it is necessary to review the code of this task, and determine what is needed to improve this code execution. Typical cases of improvements are: altering the command itself, creating automatically optimization objects (indexes, keys, etc) or using an alternate strategy. These improvements will be implemented by altering existing tasks, or by creating new tasks.

Implementing the optimization change is done in the Knowledge Modules. This implementation is made in a generic way using Oracle Data Integrator's *Substitution Methods*. These methods prevent the use of hard-coded value (column names, table names, etc) to represent metadata that will vary from one Mapping to another (for example, the name of the target table).

More information on Knowledge Module customization is available in the *Knowledge Module Developer's Guide* in Oracle Data Integrator's documentation.

Once this implementation is finished, run the mapping to test the changes and reassess the performance with the altered Knowledge Module.

RUN-TIME OPTIMIZATION

Having optimal execution time can be tuned through agent configuration, load balancing and appropriate location.

Agent Location

The agent acts, in an integration process, as the conductor. It issues the commands to the source, target and staging area server; calls bulk loader utilities and accesses temporary files.

As a consequence, its position is constrained by the need to access resources located on specific machines: loader, applications, and files.

The following considerations should be made when selecting its location:

Limit network overhead: When reading a large amount of data through JDBC, it is better to have the agent located on the server containing the data, instead of having it connecting to this server remotely.

Limit machine overhead: If the agent needs to have data flowing through it, it should not be installed on a machine with no spare resources. In Oracle Data Integrator 11g, you can adjust the *ODI_INIT_HEAP* and *ODI_MAX_HEAP* parameters in the *odiparams* configuration file to define the agents JVM initial and maximum heap size. In Oracle Data Integrator 12c the ODI_INIT_HEAP and ODI_MAX_HEAP parameters are located in the setODIDomainEnv script file within an ODI domain bin directory.

The Batch Update and Array Fetch parameters mentioned earlier also provide a way to tune the agent's performance and resource consumption.

A typical case where agent location is important is when it uses the OdiSqlUnload tool on the source to generate a temporary data file which is later loaded with a native loader to the target. In that case, it is relevant to have the agent running on the source server. If it is located on the target server, it would transfer all of the data through the network using JDBC.

Load Balancing

Load Balancing is a feature to provide optimal performance by distributing the execution of session over several agents. When an agent has linked agents defined, it automatically distributes the execution of its incoming sessions over the agents. Session distribution is made proportionally to the value specified in the *Maximum Number of Sessions* supported by an agent. This value reflects the agent's availability and processing capabilities and is relative to the other agents. If Agent A has a value twice that of Agent B, A will receive twice the number of sessions B receives.

Note that the parent agent still takes charge of the creation of the session in the repository. Load balancing separates the creation and execution of a session. If you want the parent agent to take charge of some of the sessions and not just distribute the sessions, then it must be linked to itself.

When using load balancing, it is possible that an agent could reach the maximum number of sessions allowed. In this situation, the incoming sessions for this agent will be put in a *Queued* status. The *Use pre-emptive load balancing* Agent property allows for redistribution of queued sessions to agents as sessions finish. Otherwise, sessions are not redistributed.

Each ODI agent supports multiple sessions. Several sessions running within one agent run within the same agent process, each session using a different thread in this process. With load balancing, as sessions run in different agents, they run in different processes. On multi-processor machines, this can provide dramatic increase of performance, without having to define it at run-time. Load balancing is automatically handled provided that agents are correctly linked.

Additionally, the Oracle Data Integrator platform integrates in the broader Fusion Middleware platform and becomes a key component of this stack. Oracle Data Integrator provides its run-time components as Java EE applications, enhanced to fully leverage the capabilities of the Oracle WebLogic Application Server. Oracle Data Integrator components include exclusive features for Enterprise-Scale Deployments, high availability, scalability, and hardened security. High-Availability (HA) and Scalability is fully supported via clustered deployments for Java EE components.
The Oracle Data Integrator components deployed in WebLogic Server also benefit from the capabilities of the WLS cluster for scalability, including JDBC Connection Pooling and Load Balancing. In addition to the cluster-inherited HA capabilities, the run-time agent also supports a Connection Retry mechanism to transparently recover sessions running in repositories that are stored in HA-capable database engines such as Oracle RAC. The E-LT architecture mandates the use of lightweight run-agents able to run in strategic locations (for example on a Data Warehouse server as described previously) for optimal performances of the integration processes. With this architecture, every run-time component of Oracle Data Integrator guarantees high scalability, 24/7 availability, and best-in-class security.

CONCLUSION

Oracle Data Integrator can be optimized to meet the most demanding data integration projects. The core E-LT differentiation of Oracle Data Integrator, plus advanced tuning features, Knowledge Module optimizations, and Parallel Agent sessions are all ways that ODI provides superior performance compared to traditional engine-based ETL tools. As this paper demonstrates, Oracle remains committed to providing the highest performing data integration solution for Oracle and heterogeneous systems.

APPENDIX A: MAPPING LOADING STRATEGIES

Loading Strategy	Agent on Source or Target	Agent on separate machine	Sample Loading Knowledge Modules (LKMs)
Using the agent to read from source and write to the staging area. Recommended for small volumes (less than 500K records). Should be used in conjunction with Changed Data Capture on source to reduce the amount of data flowing through the agent.	**	*	SQL to SQL; File to SQL; SQL to Oracle; SQL to DB2 UDB; SQL to MSSQL
Using native bulk loaders for flat files. Recommended for large volumes when the source is a flat file.	*****	****	File to Oracle (External Table); File to Oracle (SQLLDR); File to MSSQL Bulk; File to DB2 UDB (LOAD); File to Teradata (TTU)
Using OdiSqlUnload to extract data in a staging file and a native bulk loader to load the data in the staging area. Recommended for large volumes when the source server does not provide a native unload utility. Source data is staged in a flat file on disk.	***	**	MSSQL to Oracle (BCP SQLLDR); SQL to Teradata (TTU); SQL to DB2 400 (CPYFRMIMPF); SQL to DB2 UDB (Load); SQL to MSSQL (Bulk)
Using a native bulk unload utility to extract data in a staging file and a native bulk loader to load the data in the staging area. Recommended for large volumes when the source server provides a native unload utility. Source data is staged in a flat file on disk.	****	***	Oracle to Oracle (datapump); DB2 UDB to DB2 UDB (Export_Import)
Using piped unload/load strategy. Source data is extracted using either OdiSqlUnload or a native unload utility to a pipe. Data is consumed from the pipe using a load utility and written to the staging area. Recommended for very large volumes when source data shouldn't be staging in a separate file on disk.	*****	****	SQL to Teradata (TTU)
Using RDBMS specific built-in data transfer mechanisms such as database links or linked servers. Source data is read by the staging area RDBMS engine using native protocols and written to the staging area tables. Recommended for reasonable large volumes. The location of the agenda does no impact performance.	***	***	Oracle to Oracle (Dblink); MSSQL to MSSQL (linked servers); DB2 400 to DB2 400;

APPENDIX B: INTEGRATION STRATEGIES WITH STAGING ON TARGET

Integration Strategy	Flow Control or Static Control (CKM) OFF	Flow Control or Static Control (CKM) ON	Sample Integration Knowledge Modules (IKMs)
Simple Replace or Append. Final transformed data flow is simply appended to the target data. Recommended when incoming data flow does not conflict with target data for example in the case of a Data Warehouse Fact table.	*****	**	Insert; Control Append; Hive Control Append
Incremental Update or Merge. Final transformed data flow is compared with target data using an update key. New records are inserted and existing ones are updated where needed. Recommended when incoming data overlaps with existing target data for example in the case of a Data Warehouse Dimension table.	****	***	Oracle Merge; Oracle Update; Oracle Incremental Update; SQL Incremental Update; Teradata Incremental Update; DB2 400 Incremental Update; DB2 400 Incremental Update (CPYF); DB2 UDB Incremental Update
Slowly Changing Dimensions. Final transformed data flow is compared with target data and some updates are converted into new records with a new surrogate key. Recommended for populating Data Warehouse's Type 2 Slowly Changing Dimensions.	***	**	Oracle Slowly Changing Dimension; MSSQL Slowly Changing Dimension; DB2 400 Slowly Changing Dimension; DB2 UDB Slowly Changing Dimension; Teradata Slowly Changing Dimension

APPENDIX C: INTEGRATION STRATEGIES WITH STAGING DIFFERENT FROM TARGET

Integration Strategy	Agent on Staging or Target Machine	Agent on Other Machine	Sample Integration Knowledge Modules (IKMs)
File to Server Append. Restricted to a single file as a source directly loaded to the target table using a native bulk loader. Recommended for direct load of a single very large volume file to a target table.	*****	****	File-Hive to Oracle (OLH); File to Teradata (TTU)
Server to Server, File or JMS Append using the agent. Final transformed data is read from the staging area by the agent and appended to the target table. Data flows from the staging area to the target through the agent.	***	**	SQL to SQL Control Append; SQL to SQL Incremental Update; SQL to File Append
Server to Server Append using native bulk unload/load utilities. Final transformed data is extracted from the staging area to a file or pipe using an unload utility and loaded to the target table using a bulk loader.	***	**	SQL to Teradata Control Append
Server to File Append using native bulk unload utilities. Recommended when the target is a file. Staging transformed data is exported to the target file using the bulk unload utility.	****	***	Teradata to File (TTU)

Oracle Data Integrator Performance Guide
February 2014
Author: Oracle Data Integrator Product Management

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 © 2014, 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 and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group. 0114

Hardware and Software, Engineered to Work Together