Oracle Maximum
Availability Architecture

Oracle GoldenGate Performance Best Practices

ORACLE WHITE PAPER | MAY 2017

Introduction

The strategic integration of Oracle Exadata Database Machine and Oracle Maximum Availability Architecture (MAA) best practices (Exadata MAA) provides the best and most comprehensive Oracle Database availability solution. Oracle GoldenGate is a key component of MAA, providing a logical replication solution for fast platform migration and a near zero downtime solution for application and database upgrades. It complements the rest of Oracle's MAA solution that tolerates failures and enables online maintenance and rolling upgrade through Oracle Real Application Clusters (Oracle RAC), Oracle Automatic Storage Management (Oracle ASM), and Oracle Active Data Guard.

This white paper describes best practices for configuring Oracle GoldenGate for the best performance, simple manageability, and stability for Oracle databases. Non-Oracle databases are not covered in this paper.

Refer to *"*Oracle GoldenGate with Oracle Real Application Clusters Configuration*"* MAA white paper at the link below for the initial configuration of Oracle GoldenGate, including installation, Oracle Database File System (DBFS) configuration for shared Oracle GoldenGate files, and Oracle Real Application Clusters (Oracle RAC) services configuration.

http://www.oracle.com/technetwork/database/features/availability/maa-goldengate-rac-2007111.pdf

Oracle Software

Use Oracle GoldenGate Release 12.2.0 or later to take advantage of increased functionality and enhanced performance features.

Starting with Oracle GoldenGate Release 12.1.2, Replicat can operate in integrated mode for improved scalability within Oracle target environments. The apply processing functionality within the Oracle database is leveraged to automatically handle referential integrity and data description language (DDL) so that the operations are applied in the correct order.

Extract can also be used in integrated capture mode with an Oracle database, introduced with Oracle GoldenGate Release 11.2.1. Extract integrates with an Oracle database log mining server to receive change data from the database in the form of logical change records (LCRs). Extract can be configured to capture from a local or downstream mining database. Because integrated capture mode is fully integrated with the database, no additional setup is required to work with Oracle RAC, Oracle ASM, Transparent Data Encryption (TDE), and data compression, which greatly simplifies setup without sacrificing performance.

The latest release of Oracle GoldenGate can be downloaded from My Oracle Support, Patches and Updates.

It is recommended that you use at minimum Oracle Database 11g Release 2 (11.2.0.4) to take advantage of both integrated Extract and integrated Replicat GoldenGate features. Refer to Latest GoldenGate/Database (OGG/RDBMS) Patch recommendations (Doc ID 2193391.1).

Database Configuration

This section describes the configuration best practices for the source and target databases used in an Oracle GoldenGate replicated environment. It is assumed that the Extract and Data Pump processes are both running on the source environment, and one or more Replicat processes are running on the target database. In an active-active bi-directional Oracle GoldenGate environment, or when the target database may be converted to a source database, combine both target and source database configuration steps.

Configuring the Source Database

Do the following tasks/steps to configure the source database

1. Run the database in ARCHIVELOG mode.

Oracle GoldenGate Extract mines the Oracle redo for data that can be replicated. The database must be running in `ARCHIVELOG` mode. When using Extract in integrated capture mode, the LogMiner server can seamlessly mine redo from the log buffer, online, and archive log files.

2. Enable force logging mode.

In order to ensure that the required redo information is contained in the Oracle redo logs for segments being replicated, it is important to override any `NOLOGGING` operations which would prevent the required redo information from being generated. If you are replicating the entire database, enable database force logging mode.

Check the existing force logging status by executing the following command:

```
SQL> SELECT FORCE_LOGGING_MODE FROM V$DATABASE;
```

If the database is currently not in force logging mode, enable force logging by executing the following commands:

```
SQL> ALTER DATABASE FORCE LOGGING;
SQL> ALTER SYSTEM SWITCH LOGFILE;
```

There are cases when you do not want to replicate some application data that are loaded with `NOLOGGING` operations. In those cases, isolate the tables and indexes into separate tablespaces, then enable and disable logging according to your requirements. You must first disable database force logging mode by executing the following commands:

```
SQL> ALTER DATABASE NO FORCE LOGGING;
SQL> ALTER TABLESPACE <tablespaces_replicated> FORCE LOGGING;
SQL> ALTER TABLESPACE <tablespaces_not_replicated> NOLOGGING;
```

It is important to test the effects of force logging mode on database performance before configuring Oracle GoldenGate.

3. Enable supplemental logging.

Oracle GoldenGate requires minimal supplemental logging enabled at the database level along with additional key column values to be logged into redo to allow the same updated or deleted rows manipulated on the source database to be found on the target database.

1. Perform the following steps to verify and enable database minimal supplemental logging:

```
SQL> SELECT supplemental_log_data FROM v$database;

SQL> ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;

SQL> SELECT supplemental_log_data FROM v$database;
```

2. Add supplemental logging at the schema level using the Oracle GoldenGate command `ADD SCHEMATRANDATA`.

For more information about creating supplemental log groups, refer to *Oracle GoldenGate Installing and Configuring Oracle GoldenGate for Oracle Database* at

http://docs.oracle.com/goldengate/c1221/gg-winux/GWURF/GUID-5DA7C3DC-5D87-4A8B-AD23-6EF587A5CF41.htm#GWURF265

It is recommended to test and monitor any overheads added to the database by supplemental logging using a production workload, before enabling supplemental logging in your production environment.

4. Configure the Streams pool.

When using Extract in integrated capture mode, an area of Oracle memory called the Streams pool must be configured in the System Global Area (SGA) of the database.

The size requirement of the Streams pool for Extract in integrated capture mode is based on the number of integrated Extracts and the integrated capture mode parameter, MAX_SGA_SIZE, which controls the amount of shared memory used by the LogMiner server. The default value is 1GB which is adequate in almost all cases. This is not the same as the database initialization parameter, SGA_MAX_SIZE.

Set the STREAMS_POOL_SIZE initialization parameter for the database to the following value:

```
(MAX_SGA_SIZE * # of integrated Extracts) + 25% head room
```

For example, using the default values for the MAX_SGA_SIZE with two integrated Extracts:

```
( 1GB * 2 ) * 1.25  = 2.50GB
STREAMS_POOL_SIZE = 2560M
```

5. Configure database parameters for redo log read performance.

Set the following database initialization parameters:

```
_log_read_buffers = 64
_log_read_buffer_size = 128
```

These two database initialization parameters can improve performance of reading the redo log files by the LogMiner server by increasing the number and size of read buffers. These parameters also affect the same read buffers that are used during database media recovery, of which the performance may also improve.

6. Install the UTL_SPADV package.

The UTL_SPADV PL/SQL package provides subprograms to collect and analyze statistics for the LogMiner server processes. The statistics help identify any current areas of contention such as CPU or I/O. To install the UTL_SPADV package, as the Oracle GoldenGate administrator user on the source database, run the following SQL script:

```
SQL> @$ORACLE_HOME/rdbms/admin/utlspadv.sql
```

Later in this white paper, there is an example using the UTL_SPADV package to monitor the LogMiner server performance in real time. For more information about the UTL_SPADV package, refer to *Oracle Database PL/SQL Packages and Types Reference* at

http://docs.oracle.com/database/122/ARPLS/UTL_SPADV.htm#ARPLS883

For additional database configuration requirements, refer to *Installing and Configuring Oracle GoldenGate for Oracle Database* at

http://docs.oracle.com/goldengate/c1221/gg-winux/GIORA/GUID-CD8ABF2B-6ED2-48EE-8FD7-6062CE7F532E.htm#GIORA121

Configuring the Target Database

Do the following tasks/steps to configure the target database.

1. Run the database in ARCHIVELOG mode.

Although Oracle GoldenGate does not require that the target database run in `ARCHIVELOG` mode, Oracle recommends doing so for high availability and recoverability. If the target database is configured to fail over or switch over to a source database, `ARCHIVELOG` mode is required. The target database should also be involved in a backup strategy to match the recovery options on the source database. In the event of a failure on the source environment, and if an incomplete recovery is carried out, the target database also needs recovery to make sure that the replicated objects are not from a point in time ahead of the source.

2. Enable force logging.

When replicating bi-directionally, or if the source and target database need to switch roles, force logging should be enabled to prevent missing redo data required by Oracle GoldenGate Extract.

For instructions about how to enable force logging mode, refer to "Source Database" on page 5.

3. Configure the Streams pool.

When using integrated Replicat the Streams pool must be configured. If using non-integrated Replicat the Streams pool is not necessary.

The size requirement of the Streams pool for integrated Replicat is based on a single parameter, `MAX_SGA_SIZE`. The `MAX_SGA_SIZE` parameter defaults to `INFINITE` which allows the Replicat process to use as much of the Streams pool as possible. Oracle does not recommend setting the `MAX_SGA_SIZE` parameter.

Set the `STREAMS_POOL_SIZE` initialization parameter for the database to the following value:

```
(1GB * # of integrated Replicats) + 25% head room
```

For example, on a system with one integrated Replicat process the calculation would be as follows:

```
(1GB * 1) * 1.25 = 1.25GB
STREAMS_POOL_SIZE = 1280M
```

4. Configure the target SGA parameters.

The database parameters controlling the size of the shared memory components in the System Global Area (SGA) must be configured similarly to the source database of the data being replicated. This ensures that no unexpected drop in performance is seen due to incorrectly sized memory. For example, if the source database is configured with an 11GB buffer cache, the same performance cannot be expected with the same workload using a 2GB buffer cache.

If replicating a subset of the source database the target SGA may be sized smaller. If there is additional database work carried out on the target database, such as increased reporting applications, the SGA should be increased accordingly.

For additional database configuration requirements, refer to *Installing and Configuring Oracle GoldenGate for Oracle Database* at

http://docs.oracle.com/goldengate/c1221/gg-winux/GIORA/GUID-CD8ABF2B-6ED2-48EE-8FD7-6062CE7F532E.htm#GIORA121

Oracle GoldenGate Configuration

This section describes the configuration best practices for Oracle GoldenGate components Extract, Data Pump, and Replicat.

Extract Configuration

Oracle recommends using Oracle GoldenGate Release 12.2.0.1 or later with the integrated capture mode Extract to take advantage of the integration with the LogMiner server. Integrated capture enables seamless extraction of more data types than with classic mode Extract, such as compressed data (Basic, OLTP, and Exadata Hybrid Columnar Compression). There is no additional configuration required for Extract to read log files stored on Oracle ASM. RMAN's fast recovery area policies ensure that archive logs cannot be removed until Extract no longer needs them.

When using integrated capture the default settings are appropriate for most environments. Be sure to set the `STREAMS_POOL_SIZE` initialization parameter correctly, as explained in "Database Configuration" on page 5. It is also recommended that you set the integrated Extract parameter `_LOGMINER_READ_BUFFERS`, which controls the number of buffers (64KB each) used by LogMiner to read redo. These buffers are allocated from the streams pool allotted to LogMiner and default to 64. It is recommended that you set the `_LOGMINER_READ_BUFFERS` to a value of 256.

```
TRANLOGOPTIONS INTEGRATEDPARAMS (_LOGMINER_READ_BUFFERS 256)
```

Data Pump Configuration

The primary function of the Data Pump is to read the trail files created by Extract and route data to the target host. Use the `PASSTHRU` parameter in the Data Pump parameter file to increase Data Pump performance and reduce CPU usage when table names and table structures are not altered or data is being filtered. This prevents the Data Pump from looking up table definitions from either the database or from a data definitions file. The `PASSTHRU` parameter is table-specific and can be configured with a wildcard character to apply to multiple tables.

If there are tables that require mapping or data conversions, use the `NOPASSTHRU` parameter. Tables listed with the `NOPASSTHRU` parameter must be specified after the `PASSTHRU` parameter, as shown in the example below.

In the following example, the `PASSTHRU` parameter instructs the Data Pump to pass through all tables belonging to the `SOESMALL` schema, but the `SOEADMIN.OPS` table is processed normally.

```
EXTRACT dpump_1a
USERID soeadmin, PASSWORD ****
RMTHOST ggdb02, MGRPORT 8901
RMTTRAIL /home/oracle/goldengate/latest/dirdat_os/aa
```

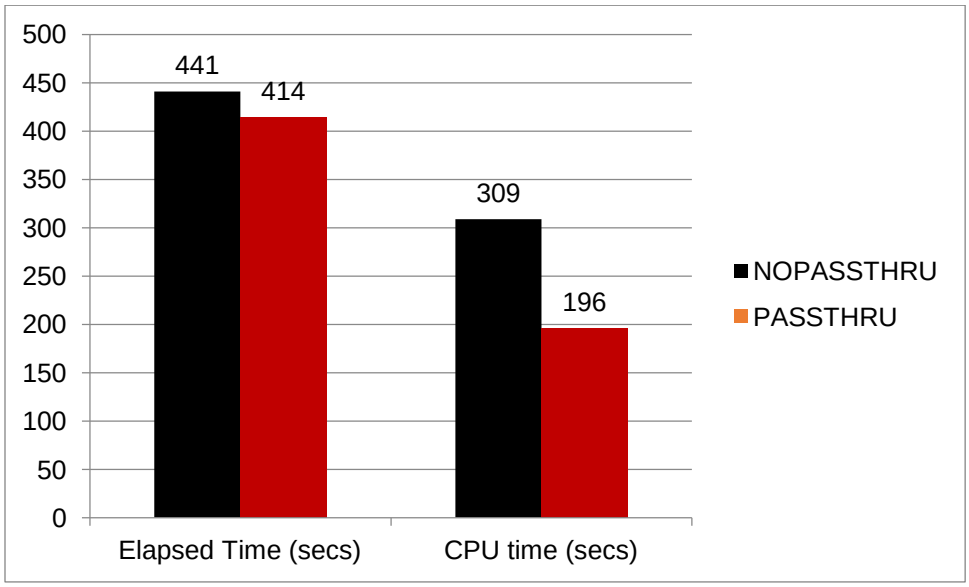
```
REPORTCOUNT EVERY 15 MINUTES, RATE

PASSTHRU
TABLE SOESMALL.*;

NOPASSTHRU
TABLE SOEADMIN.OPS, WHERE (OPNO < 10);
```

A performance comparison test was performed to show the difference in elapsed time and CPU time with Data Pump using the `PASSTHRU` and `NOPASSTHRU` (the default) parameters. The workload was a Swingbench online transaction processing (OLTP) type workload with approximately 10 DML statements per transaction affecting three tables (5 inserts, 5 updates). A total of 34 trail files were generated totaling 16GB in size.

The following graph shows the difference in elapsed time and CPU seconds used for the two tests.



There is a 6% reduction in elapsed time when using the `PASSTHRU` parameter, but the bigger difference is the reduction in CPU time which is 37% less. This improvement can vary depending on the amount of data being replicated that requires conversion or mapping by the Data Pump.

When replicating across a Wide Area Network (WAN), follow these best practices:

1. Tune TCPBUFSIZE and TCPFLUSHBYTES parameters.

The two `RMTHOST` parameters, `TCPBUFSIZE` and `TCPFLUSHBYTES`, are very useful for increasing the buffer sizes and network packets sent by Data Pump over the network from the source to the target system. This is especially beneficial for high latency networks.

It is recommended that you set these parameters to an initial value of 1MB (1,048,576 bytes) or the calculated value, whichever is larger.

To determine a suitable value, perform the following steps:

a. Use the `ping` command to obtain the average round trip time (RTT).

For example:

```
% ping ggsoftware.com

Pinging ggsoftware.com [192.168.116.171] with 32 bytes of data:
Reply from 192.168.116.171: bytes=32 time=31ms TTL=56
Reply from 192.168.116.171: bytes=32 time=61ms TTL=56
Reply from 192.168.116.171: bytes=32 time=32ms TTL=56
Reply from 192.168.116.171: bytes=32 time=34ms TTL=56

Ping statistics for 192.168.116.171:
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
Minimum = 31ms, Maximum = 61ms, Average = 39ms
```

b. Calculate the bandwidth-delay product (BDP).

For example, if the network between the source and target databases is 155 megabits per second (Mbits) and the latency is 39ms the calculation would be as follows:

```
BDP = (155,000,000 / 8) * 0.039 = 755,625bytes
```

c. Multiply the result by 3 to determine 3xBDP.

For example:

```
3xBDP = 755,625 x 3 = 2,266,875
```

In this example, because the result is more than 1MB, set the values of `TCPBUFSIZE` and `TCPFLUSHBYTES` to `2,266,875`.

The parameters are set in the Data Pump parameter file. For example:

```
RMTHOST target, MGRPORT 7809, TCPBUFSIZE 2266875, TCPFLUSHBYTES 2266875
```

The maximum socket buffer size for non-Windows systems is usually limited by default.

Ask your system administrator to increase the maximum socket buffer size on the source and target systems so that Oracle GoldenGate can increase the buffer size configured with the TCPBUFSIZE parameter.

2. Use Data Pump compression if network bandwidth is constrained and when CPU headroom is available.

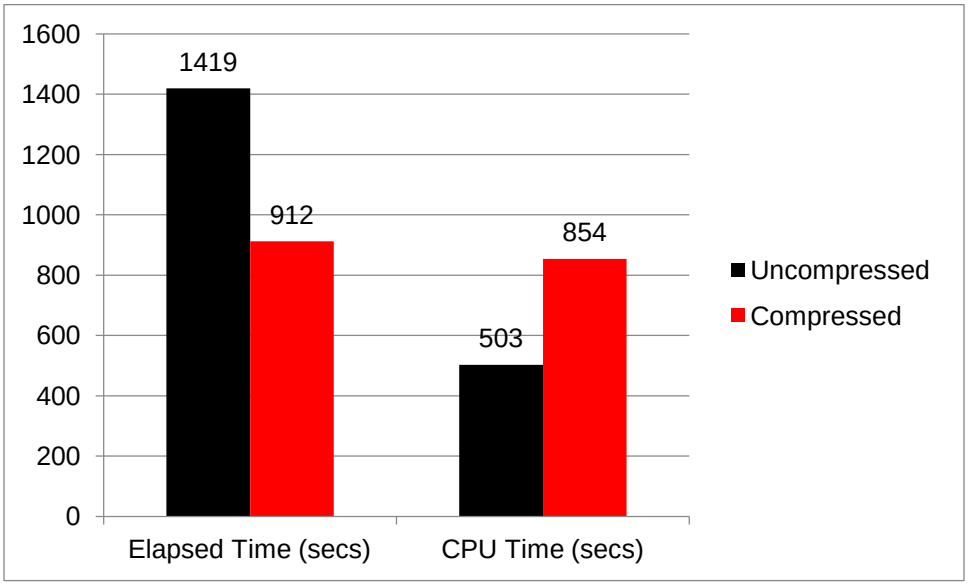
Use Data Pump compression only if network bandwidth is insufficient or network latency is high. Before the Data Pump sends the trail file data to the Collector process on the target database, data is compressed on the source. The Collector process then uncompresses that data upon receipt, and the uncompressed data is written to the target database trail files.

The compression ratio is dependent on the type of data contained in the trail files. For example, scalar data types like `VARCHAR2`, `DATE`, and `NUMBER` compress better than compressed LOB column data. Enabling compression uses more CPU for each Data Pump process on the source and for the Collection server process on the target

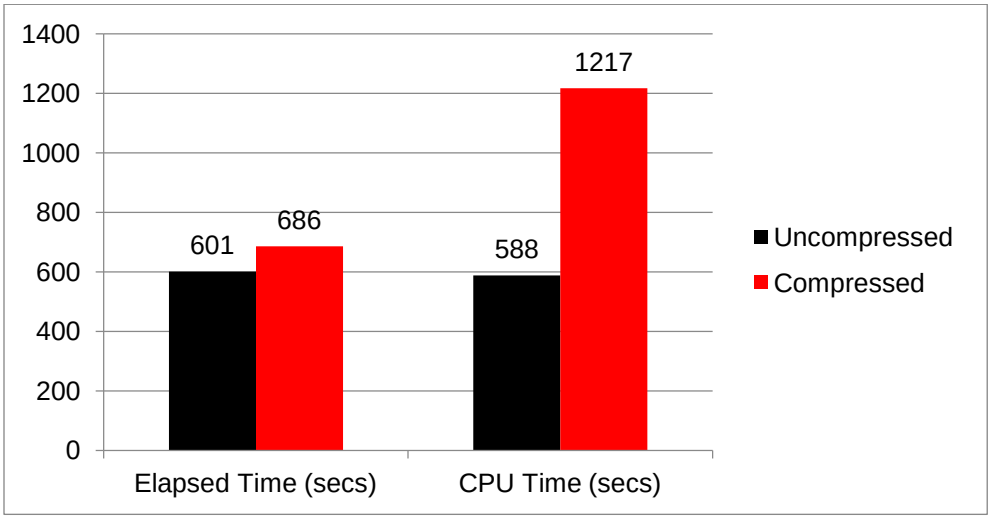
system. Compression is enabled in the Data Pump parameter file using the `RMTHOST COMPRESSION` parameter.

As an example of the Data Pump performance benefits, with a 90ms network latency using a 10 Gigabit network connection between two servers, and trail file data containing just scalar data types, the effects of adding compression is shown in the following graph.

The graph shows a 64% reduction in time when compressions is enabled, with a 70% increase in CPU consumed by the source Data Pump and the target Collector processes combined.



With a lower network latency of 15ms, there was a performance degradation of 14%, with a 2X increase in CPU consumed by the source Data Pump and the target Collector processes combined when Data Pump compression was enabled, as shown in the following graph.



It is recommended that you test Data Pump compression in your current environment before deciding if it will be of benefit.

For more information about the `RMTHOST COMPRESSION` parameter for Data Pump, refer to *Reference Guide for Oracle GoldenGate for Windows and UNIX* at

http://docs.oracle.com/goldengate/c1221/gg-winux/GWURF/GUID-4675F779-D83C-4AF6-9B0A-C811CC715F15.htm#GWURF631

Replicat Configuration

The Oracle GoldenGate Replicat processes running on the target database read the trail files and apply the data using SQL DML statements on the replicated objects.

The following are recommendations to optimize Replicat performance.

1. Configure integrated Replicat.

Oracle recommends using integrated Replicat, introduced in Oracle GoldenGate Release 12.1 and Oracle Database 11*g* Release 2 (11.2.0.4). Integrated Replicat leverages the database apply process functionality. Referential integrity is maintained and DDL operations are automatically applied. This alleviates the database administrator from having to understand how to partition tables between Replicat processes based on foreign key constraints, or from having to ensure that the correct Replicat handles the DDL for tables.

Integrated Replicat offers automatic parallelism which automatically increases or decreases the number of based on the current workload and database performance. Management and tuning of Replicat performance is simplified because you do not have to manually configure multiple Replicat processes to distribute the tables between them. Integrated Replicat automatically enables the asynchronous commit feature so processing can continue immediately after each `COMMIT` command is issued.

For more information about integrated Replicat configuration, refer to *Installing and Configuring Oracle GoldenGate for Oracle Database* at

http://docs.oracle.com/goldengate/c1221/gg-winux/GIORA/GUID-8AC69DA1-1117-47DE-BD8C-DB3BC51A1DA1.htm#GUID-8AC69DA1-1117-47DE-BD8C-DB3BC51A1DA1

2. Set BATCHSQL in the Replicat parameter file.

By default, integrated Replicat tries to reorder and group DML statements of the same type against the same object within each transaction, and applies the transaction DML as a batch instead of applying each DML statement individually. Using batches can reduce the CPU and execution time of DML statements.

To increase the Replicat apply performance further, enable `BATCHSQL`, which groups multiple transactions into fewer, larger transactions, batching the same DML types together. This is enabled by adding the `BATCHSQL` parameter to the Replicat parameter file.

The following is an example from a target database AWR report, using integrated Replicat without BATCHSQL enabled. There are 3-4 rows per execution on two of the tables.

SQL ordered by Executions

Executions	Rows Processed	Rows per Exec	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Module	SQL Text
20,468,975	20,469,219	1.00	2,508.66	53.4	.6	bu28rqwk7y6rb	GoldenGate	UPDATE /*+ restrict_all_ref_c...
20,466,464	20,466,811	1.00	2,912.26	20.9	5.8	5q8xtx2bhquzj	GoldenGate	INSERT /*+ restrict_all_ref_c...
19,606,938	72,709,149	3.71	2,381.43	37.5	6.7	f4vq0jufrmsx0	GoldenGate	INSERT /*+ restrict_all_ref_c...
19,510,351	65,663,725	3.37	2,478.54	64.8	.1	7kmvq4xrdsuk7	GoldenGate	UPDATE /*+ restrict_all_ref_c...

When BATCHSQL is enabled, the rows per execution increases and the elapsed time decreases as shown below.

SQL ordered by Executions

Executions	Rows Processed	Rows per Exec	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Module	SQL Text
4,855,893	20,477,534	4.22	1,941.08	98.3	.2	bu28rqwk7y6rb	GoldenGate	UPDATE /*+ restrict_all_ref_c...
4,855,639	20,476,399	4.22	1,670.95	39.3	7.8	5q8xtx2bhquzj	GoldenGate	INSERT /*+ restrict_all_ref_c...
4,835,457	72,743,061	15.04	1,453.91	70.4	7.3	f4vq0jufrmsx0	GoldenGate	INSERT /*+ restrict_all_ref_c...
4,832,499	65,837,321	13.62	1,991.06	89.1	.1	7kmvq4xrdsuk7	GoldenGate	UPDATE /*+ restrict_all_ref_c...

By adding the BATCHSQL parameter to the Replicat parameter file, which enables BATCHSQL, this small OLTP workload example improved performance by approximately 31%.

The maximum size of each statement batch is controlled by the BATCHSQL BATCHTRANSOPS parameter. The default size of 50 for integrated Replicat is adequate in most cases, but changing the batch size may result in performance gains.

Setting the batch size too low or too high may result in performance degradation. Integrated Replicat applies transactions in parallel, so setting BATCHTRANSOPS too high can result in increased dependencies between transactions, which results in slower performance. When changing the BATCHTRANSOPS size, do so in a controlled manner so performance with the old and new settings can be accurately compared.

For more information about the BATCHSQL parameter refer to *Reference for Oracle GoldenGate Windows and UNIX* at

http://docs.oracle.com/goldengate/c1221/gg-winux/GWURF/GUID-2ED88418-6ACB-484D-B140-364232EC419A.htm#GWURF404

Configure the GoldenGate Heartbeat Table

Introduced in Oracle GoldenGate release 12.2.0.1 is the built-in heartbeat table feature that provides end-to-end replication lag views without having to manually implement your own heartbeat table. After creating the heartbeat table using the GGSCI command ADD HEARTBEAT, it is possible to see the end-to-end replication latency by looking at the GG_LAG database view.

To create and enable the GoldenGate heartbeat table:

1. Add the following parameter to the GLOBALS file on both the source and target GoldenGate installations to set the heartbeat table database schema name and table naming convention.

```
HEARTBEATTABLE SOE.GG_HEARTBEAT
```

It is recommended that you use a schema name (SOE in this example) that is already being replicated by GoldenGate.

2. Create the heartbeat table and the scheduler jobs to update the heartbeat table using GGSCI.

```
GGSCI> DBLOGIN USERID SOE, password SOE
GGSCI> ADD HEARTBEATTABLE, FREQUENCY 5
```

This must be carried out on both the source and target databases.

3. Confirm the scheduler job was created in the source database.

```
SQL> select JOB_NAME, START_DATE, LAST_START_DATE, NEXT_RUN_DATE
from dba_scheduler_jobs where job_name ='GG_UPDATE_HEARTBEATS';
```

Example output:

```
JOB_NAME
--------------------------------------------------------------------------------
START_DATE
---------------------------------------------------------------------------
LAST_START_DATE
---------------------------------------------------------------------------
NEXT_RUN_DATE
---------------------------------------------------------------------------
GG_UPDATE_HEARTBEATS
16-NOV-16 03.15.44.030278 PM AMERICA/LOS_ANGELES
15-DEC-16 11.41.21.188995 AM AMERICA/LOS_ANGELES
15-DEC-16 11.41.24.000000 AM AMERICA/LOS_ANGELES
```

If the LAST_START_DATE and NEXT_RUN_DATE columns are not updating, manually execute the scheduler job, connected to the database as the heartbeat table owner as shown here.

```
SQL> connect soe/soe
SQL> exec dbms_scheduler.run_job('GG_UPDATE_HEARTBEATS');
```

4. Monitor the replication on the target database.

```
col Lag(secs) format 999.9
col "Seconds since heartbeat" format 999.9
col "GG Path" format a32
col TARGET format a12
col SOURCE format a12
set lines 140
SELECT remote_database "SOURCE", local_database "TARGET", incoming_path "GG
Path", incoming_lag "Lag(secs)", incoming_heartbeat_age "Seconds since heartbeat"
FROM ggadmin.gg_lag;
```

For more information about creating the heartbeat table, refer to *Reference for Oracle GoldenGate on Windows and UNIX* at

http://docs.oracle.com/goldengate/c1221/gg-winux/GWURF/GUID-126E30A2-DC7A-4C93-93EC-0EB8BA7C13CB.htm#GWURF1238

Database File System (DBFS) Configuration

When running Oracle GoldenGate on Oracle Exadata Database Machine, with Oracle RAC or Oracle Data Guard configurations, Oracle recommends placing the shared Oracle GoldenGate files (trail files, checkpoint files, bounded recovery files, and parameter files) on Database File System (DBFS) file systems. Using DBFS provides integration with Cluster Ready Services (CRS) which automates DBFS file systems mounting on a surviving node in the Oracle RAC cluster. This allows Oracle GoldenGate processes to automatically start after the required file systems have been mounted.

For details about how to configure DBFS for optimal performance and availability, refer to the "Oracle GoldenGate on Exadata Database Machine Configuration" MAA white paper at

http://www.oracle.com/technetwork/database/features/availability/maa-wp-gg-oracledbm-128760.pdf

Using DBFS with Oracle Data Guard and Oracle GoldenGate provides synchronization between the source and target databases with the external files used by Oracle GoldenGate. This is important during role transitions, especially for automatic restart of Oracle GoldenGate processes after a failover. For information about the configuration of such an environment refer to the "Transparent Zero Data-Loss Role Transition with Oracle Data Guard and Oracle GoldenGate" MAA white paper at

http://www.oracle.com/technetwork/database/availability/ogg-adg-zdl-2219106.pdf

Data Gathering for Oracle GoldenGate Performance

To troubleshoot Oracle GoldenGate performance there are several key pieces of information that must be gathered and analyzed. You normally start tuning when you first encounter an unacceptable lag or latency (the time taken to extract or apply the data from the time it was created on the source database) and the throughput decreases. Because of the decoupled architecture of Oracle GoldenGate it is important to gather performance data on both the source and target environments for the same time period.

The following pieces of information are necessary for Oracle GoldenGate performance analysis:

- Latency or lag time for each Oracle GoldenGate process.
- Oracle GoldenGate process report files and the ggserr.log error log.
- Automatic Workload Repository (AWR) and Active Session History (ASH) database reports.
- CPU and I/O data.
- Oracle Streams Performance Advisor (SPADV) report.
- Integrated Capture and Integrated Replicat healthcheck report.

It is recommended to use the script provided in MOS note 2262988.1 to gather all of the above information covering the same period of time:

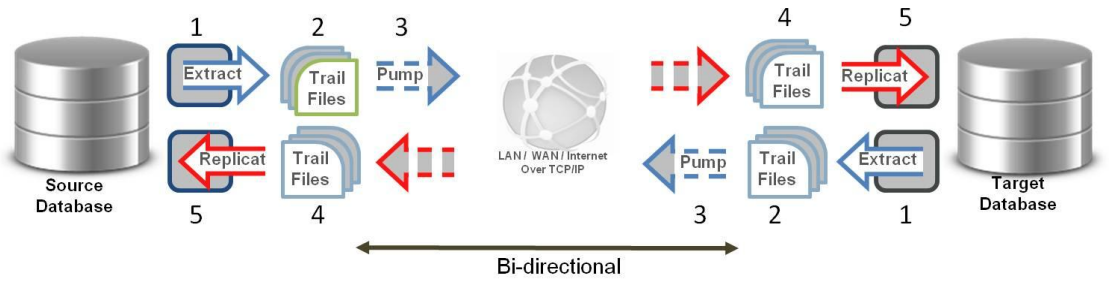
https://support.oracle.com/oip/faces/secure/km/DocumentDisplay.jspx?id=2262988.1&h=Y

The script is run on each server where Oracle GoldenGate processes are running.

Refer to Appendix A for more detailed instructions on how to manually gather these key pieces of information.

Oracle GoldenGate Performance Tuning Methodology

Before you try to diagnose slow performance in an Oracle GoldenGate environment, it is important to first understand the flow of data between the source and target databases. The following figure shows the flow of data between the source and target databases.



The following components are capable of contributing to a performance bottleneck:

1. Oracle log files are read by the Extract process capturing any required data for replication.
2. Extract carries out any mapping and conversion to the data and then writes it out to the trail files.
3. Data Pump reads the trail files and carries out any mapping and conversion required to the data.
4. Data Pump transfers the trail files from the source system to the target system where it is written by the Collector process to the remote trail files.
5. Replicat reads the trail file, applies any mapping and conversions, and applies the data to the target database using SQL statements.

The following workflow demonstrates how to determine and resolve where replication latency is introduced in Oracle GoldenGate and, consequently, where the performance bottleneck is present. Performance tuning is an iterative process. Once something has been changed in the environment, lag needs to be monitored and then the tuning process repeated.

1. Locate where the latency is first reported, moving from Extract to Replicat.

Move from the source to target side, using the previously recommended method to gather data to view the Oracle GoldenGate process latency (database queries, `ggserr.log`, `ggsci INFO *`, `LAG EXTRACT`, or `LAG REPLICAT`).

Once the process with lag has been determined, continue to the next step.

The following example shows output from the `ggserr.log` file for an Extract and Data Pump process with increasing lag.

```
2014-11-01 00:47:30  INFO    OGG-01026  Oracle GoldenGate Capture for Oracle,
dpump 1a.prm: Rolling over remote file /goldengate/latest/dirdat os/aa000031.
2014-11-01 00:47:37  WARNING OGG-00947  Oracle GoldenGate Manager for Oracle,
mgr.prm:  Lag for EXTRACT DPUMP_1A is 00:00:04 (checkpoint updated 00:00:02 ago).
2014-11-01 00:47:37  WARNING OGG-00947  Oracle GoldenGate Manager for Oracle,
mgr.prm:  Lag for EXTRACT EXT_1A is 00:00:21 (checkpoint updated 00:00:08 ago).
2014-11-01 00:48:37  WARNING OGG-00947  Oracle GoldenGate Manager for Oracle,
mgr.prm:  Lag for EXTRACT DPUMP_1A is 00:00:21 (checkpoint updated 00:00:08 ago).
2014-11-01 00:48:37  WARNING OGG-00947  Oracle GoldenGate Manager for Oracle,
mgr.prm:  Lag for EXTRACT EXT_1A is 00:00:27 (checkpoint updated 00:00:01 ago).
```

It is interesting to note that when Extract lag increases, so does Data Pump lag. This implies that if you resolve the Extract lag, the Data Pump lag also decreases. Depending on when the last checkpoint of each process occurred, the lag values may differ.

2. Latency is reported for an Extract process (not a Data Pump Extract).

a. If the Extract process (classic or integrated capture mode) is reaching maximum CPU (90-100%) as shown in `top`, create an additional Extract process and partition the work to be extracted between them. When dividing workload between Extract processes you must also create additional Data Pump and Replicat processes.

For example:

```
top - 18:22:41 up 184 days,  3:52,  4 users,  load average: 1.00, 0.66, 0.37
Cpu(s): 7.8%us,  1.3%sy,  0.0%ni, 90.5%id,  0.5%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  99060552k total, 61840724k used, 37219828k free,  3399436k buffers
Swap: 25165816k total,        0k used, 25165816k free, 24251384k cached

   PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 79023 oracle    20   0  697m  61m  31m R 98.8  0.1   5:17.75
/goldengate/latest/extract PARAMFILE /goldengate/latest/dirprm/ext_1a.prm
 79034 oracle    20   0  697m  61m  31m S 52.1  0.1   2:46.17
/goldengate/latest/extract PARAMFILE /goldengate/latest/dirprm/ext_1a.prm
 78929 oracle  20   0  232m  24m  14m R 38.8  0.0   2:04.78
/goldengate/latest/extract PARAMFILE /goldengate/latest/dirprm/ext_1a.prm
```

To ensure data integrity, parent and child tables with referential integrity relationships should be processed by the same Extract process.

b. If one or more of the LogMiner preparer processes are reaching maximum CPU (90-100%), the LogMiner Reader (LMR) has a high percentage of flow control, and if there is available idle time for the LogMiner builder (LMB) process, increase the Extract `PARALLELISM` parameter (described earlier). This shows up in `top` and SPADV.

The following is an example of SPADV output.

```
PATH 2 RUN_ID 42 RUN_TIME 2013-MAR-21 15:16:16 CCA Y
|<C> OGG$CAP_EXT_1A 125182 125147 94239 LMR 0% 80% 20% "CPU + Wait for CPU" LMP
(1) 0% 0% 100% "CPU + Wait for CPU" LMB 60% 0% 40% "CPU + Wait for CPU" CAP 60%
0% 40% "CPU + Wait for CPU" |<Q> "STREAMSADMIN"."OGG$Q_EXT_1A" 125126 0.01 564
|<A> OGG$EXT_1A 0.01 0.01 0 |<B> NO BOTTLENECK IDENTIFIED
```

The following is an example of `top` output.

```
top - 15:16:09 up 71 days, 45 min,  4 users,  load average: 2.13, 1.39, 0.95
Cpu(s): 13.8%us, 1.3%sy, 0.0%ni, 84.7%id,  0.2%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  99060552k total, 58834488k used, 40226064k free,  2644064k buffers
Swap: 25165816k total,        0k used, 25165816k free, 21705628k cached

   PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 86969 oracle    20   0 18.0g 520m 509m R 99.7  0.5   8:03.73 ora_ms02_GGS1
 86955 oracle    20   0 18.1g 543m 520m R 55.7  0.6   4:33.26 ora_cp03_GGS1
 87064 oracle    20   0  446m  75m  18m R 48.9  0.1   3:58.58
/goldengate/latest/extract PARAMFILE /goldengate/latest/dirprm/ext_1a.prm
```

The following query can be used to identify the LMP process identifiers.

```
SELECT c.capture_name, lp.spid
  FROM V$LOGMNR_PROCESS lp, DBA_CAPTURE c
  WHERE lp.session_id=c.logminer_id
  AND lp.role='preparer';
```

c. If there are I/O waits in the source database for log file reads (for example, if AWR shows 'log file sequential read' > 20ms), place the log files on a faster I/O system.

d. If there are I/O wait times on the Oracle GoldenGate trail file location, move the trail files to a higher performing file system.

When trail files are located on non-DBFS storage, use `iostat` to quickly identify the issue. For example:

```
$ iostat -x 30

Time: 12:35:00 PM
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          17.03    0.00    3.47    7.83    0.00   71.68
Device:         wrqm/s   r/s   w/s      wkB/s  avgrq-sz avgqu-sz   await  svctm  %util
sda2            8361.40  0.00 445.60  38630.40   173.39    88.49  192.18   1.38  61.48

Time: 12:35:30 PM
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          14.76    0.00    2.31    8.81    0.00   74.12
Device:         wrqm/s   r/s   w/s    wkB/s    avgrq-sz avgqu-sz   await  svctm  %util
sda2            18551.60  0.00 994.80 77213.60   155.23   187.03  175.34   1.01 100.00
```

If trail files are located on a DBFS file system, a combination of `iostat` and Automatic Workload Repository (AWR) reports from the DBFS instance can similarly identify any I/O contention.

e. If you are using integrated Extract and if there are high background waits (>25%) in the source database AWR report for 'LogMiner preparer: memory' or 'LogMiner reader: buffer', increase the `MAX_SGA_SIZE` Extract parameter by 25%. Make sure the `STREAMS_POOL_SIZE` initialization parameter is sized large enough. Memory sizing was discussed earlier in this white paper.

The following is an example from an AWR report.

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn	% bg time
LogMiner preparer: memory	512,085	44	6,375	12	28.83	39.86
LogMiner reader: buffer	1,015,017	16	3,564	4	57.15	22.28

3. Latency is reported for a Data Pump process.

a. If the Data Pump process is reaching maximum CPU (90-100%) as shown in `top`, using the `PASSTHRU` parameter helps decrease CPU consumption (detailed previously in this white paper). If this does not help due to data mappings, create an additional Data Pump process and partition the work between them. This should only occur when the Data Pump process processing many transformations or conversions. You must also configure multiple Replicat processes on the target database to apply trail files from the different Data Pumps.

The following example shows `top` output.

```
top - 14:47:27 up 8 days, 22:47,  3 users,  load average: 0.83, 0.38, 0.14
Cpu(s):  4.0%us,  0.4%sy,  0.0%ni, 95.5%id,  0.0%wa,  0.0%hi,  0.1%si,  0.0%st
Mem:  99060552k total, 61544204k used, 37516348k free,  1390584k buffers
Swap: 25165816k total,        0k used, 25165816k free, 26834216k cached

   PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 34485 oracle    20   0  249m  32m  15m R 99.3  0.0   2:18.08 /u01/
goldengate/latest/extract PARAMFILE
```

b. If there are I/O wait times on the Oracle GoldenGate trail file location, move the trail files to a higher performing file system. This problem also appears in Extract performance. For an example, refer to item 2d in the preceding section.

c. If there are I/O wait times on the Oracle GoldenGate trail file location on the target database, move the trail files to a higher performing file system. This problem also appears in Replicat trail file reading performance (discussed later in this white paper). For an example, refer to item 2d in the preceding section.

d. If there is a network bandwidth or high latency problem identified by operating system utilities or network monitoring tools, consider enabling Data Pump compression as detailed in "Data Pump Configuration" on page 9.

4. Latency is reported for a Replicat process.

a. If a Replicat or the network receiver process (ANR) is reaching maximum CPU (90-100%) as shown in `top`, create an additional Replicat process and partition the work between the new process and the bottlenecked Replicat process.

Before dividing the work between multiple Replicat processes, consider the following:

» Referential integrity between tables

To ensure data integrity, parent and child tables with referential integrity relationships should be processed by the same Replicat process. For tables that are not part of referential integrity constraints, (for example,

Peoplesoft Payroll tables), assigning tables among multiple Replicat processes becomes an easier task by evenly distributing the load among each Replicat process.

» Handling of DDL statements

In order to avoid locking conflicts between Replicat processes, it is very important to understand the nature of DDL statements that occur against replicated objects. You must apply DDL with the same Replicat process that applies DML for the table. If not configured in this way, the Replicat processes can abort when a DDL statement times out waiting for another process to finish applying DML to the same table. There are two ways to avoid this issue:

» Use coordinated Replicat. Coordinated Replicat is a multithreaded process that applies transactions in parallel instead of serially. Each thread handles all of the filtering, mapping, conversion, SQL construction, and error handling for its assigned workload. A coordinator thread coordinates transactions across threads to account for dependencies, and also ensures that DDL is applied in a synchronized fashion preventing DML from occurring on the same object at the same time. For more information about coordinated Replicat, refer to *Administrating Oracle GoldenGate for Windows and UNIX* at

http://docs.oracle.com/goldengate/c1221/gg-winux/GWUAD/GUID-3388C553-994A-4ECD-95C8-A54FC7446E67.htm#GWUAD953

» Oracle recommends that you avoid using the `@RANGE` function to divide a table among Replicat processes if DDL is also replicated for the table. It is not possible to predict if all DML is completed before the DDL is applied. To help alleviate the DDL timeout issue, use the `DDL EXCLUDE` or `INCLUDE` parameters to instruct the Replicat process to which tables DDL can be applied.

For more details about replicating DDL statements, refer to the *Installing and Configuring Oracle GoldenGate for Oracle Database* at

http://docs.oracle.com/goldengate/c1221/gg-winux/GIORA/GUID-0A230601-A447-499C-B31F-C9431E1FF034.htm#GIORA285

For information about automating the workload division between multiple Replicat processes, refer to MOS Note 2224542.1 at

https://support.oracle.com/CSP/main/article?cmd=show&type=NOT&id=2224542.1

If Replicat is doing a lot of data transformations, consider moving the transformations to the Data Pump process.

The following example shows SPADV output that shows the ANR process as the bottleneck.

```
PATH 2 RUN_ID 38 RUN_TIME 2016-DEC-12 15:43:23 CCA Y
|<PR> "replicat"=> 0% 0% 100% "CPU + Wait for CPU" |<Q>
"SOESMALL"."OGGQ$REP_1A" 148384 0.01 217 |<A> OGG$REP_1A 145696 29 -1 APR 6.7%
0% 93.3% "CPU + Wait for CPU" APC 100% 0% 0% "" APS (11) 886.7% 0% 213.3% "CPU
+ Wait for CPU" |<B> "replicat"=> 3873 9 100.% "CPU + Wait for CPU"
```

The `top` output also shows the ANR process as the highest CPU consumer.

```
  PID  USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 25038 oracle    20   0 38.9g 1.6g 1.6g S 92.6  0.7   8:38.34 oracleTEST11
(DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq)))
 25070 oracle    20   0 38.9g 1.5g 1.5g R 69.0  0.6   7:08.56 ora_as01_TEST11
 25030 oracle    20   0  504m  46m  21m S 65.5  0.0   6:24.45
/u01/oracle/goldengate/ga122_11g/replicat PARAMFILE /u01/oracle/golde
 25074 oracle    20   0 38.9g 1.7g 1.7g S 21.5  0.7   1:49.42 ora_as03_TEST11
 25072 oracle    20   0 38.9g 1.7g 1.7g S 21.0  0.7   2:00.73 ora_as02_TEST11
```

The ANR process can be identified using the following query.

```
SQL> SELECT DST_QUEUE_SCHEMA, DST_QUEUE_NAME, TOTAL_MSGS, SPID, STATE
     from V$PROPOGATION_RECEIVER;

DST_QUEUE_SCHEMA DST_QUEUE_NAME TOTAL_MSGS SPID
---------------- -------------- ---------- --------
SOE                OGGQ$REP_1A      90848 25038
```

b. If there are I/O wait times on the Oracle GoldenGate trail file location, move the trail files to a higher performing file system.

This problem also appears in Data Pump performance. If multiple Replicat processes are configured to read the same trail files, consider using additional Data Pump processes so that fewer Replicat processes are reading from the same files concurrently. For an example, refer to item in the preceding section 2d (If latency is reported for an Extract process).

c. If there are no bottlenecks for the Replicat or ANR processes (not constrained by CPU or trail file I/O), but lag is being reported for the Replicat process by GoldenGate, it is likely due to one of the integrated apply processes. Use SPADV to identify which process is closest to 100% CPU, and then use the following to guide you in resolving it.

i. Apply Reader (APR) process.

The apply reader process is responsible for accumulating the changes into transactions, computing dependencies between them, and then passing them to the apply coordinator.

Here is an example SPADV output showing the apply reader bottlenecked by CPU.

```
PATH 1 RUN_ID 37 RUN_TIME 2017-JAN-17 15:40:41 CCA Y
|<PR> "replicat"=> 13.3% 0% 86.7% "CPU + Wait for CPU" |<Q>
"SOESMALL"."OGGQ$REP_1A" 180238 0.01 2437 |<A> OGG$REP_1A 185704 37
4032936 APR 0% 0% 100% "CPU + Wait for CPU" APC 100% 0% 0% "" APS (6)
373.3% 0% 226.7% "CPU + Wait for CPU" |<B> OGG$REP_1A APR 3326 7 100.%
"CPU + Wait for CPU"
```

Linux `top` output also shows the apply reader is the top CPU consumer.

```
   PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 54761 oracle    20   0 18.1g 406m 376m R 99.4  0.4   7:41.58 ora_as01_GGA1
 54770 oracle    20   0 18.1g 5.5g 5.5g R 92.3  5.8   4:36.99 ora_as05_GGA1
 55314 oracle    20   0 18.1g 6.1g 6.1g S 91.5  6.5   5:49.73 ora_as06_GGA1
 54767 oracle    20   0 18.1g 5.5g 5.5g R 91.2  5.8   4:55.66 ora_as04_GGA1
 54765 oracle    20   0 18.1g 5.3g 5.3g S 80.8  5.6   4:26.92 ora_as03_GGA1
 54684 oracle    20   0  320m  73m  25m R 54.3  0.1   4:06.57
/u01/oracle/goldengate/replicat PARAMFILE /u01/oracle/golden
```

When the apply reader is bottlenecked on CPU, there are three possible solutions:

- Reduce the number of foreign key or primary key constraints to reduce the key dependency computations.
- Increase the source transaction sizes to reduce the overhead of transaction dependency tracking.
- Create multiple integrated Replicat processes and manually partition groups of dependent objects between them. Refer to MOS Note 2224542.1 at

  https://support.oracle.com/CSP/main/article?cmd=show&type=NOT&id=2224542.1

ii. Apply Coordinator (APC) process

The apply coordinator process is responsible for batching transactions together and scheduling them with the apply server processes. It is less common to see the apply coordinator as the bottlenecked process, but it can be caused by using a `BATCHSQL BATCHTRANSOPS` value that is too high. If the apply coordinator is constrained by CPU, try reducing the `BATCHTRANSOPS` size or create multiple integrated Replicat processes and manually partition groups of dependent objects between them. Refer to MOS Note 2224542.1 at

https://support.oracle.com/CSP/main/article?cmd=show&type=NOT&id=2224542.1

The following is an example of SPADV output.

```
PATH 2 RUN_ID 71 RUN_TIME 2014-JUL-24 11:26:18 CCA Y
|<R> REP_1A 51062 7371910 0 6.7% 80% 13.3% "" |<Q> "SOESMALL"."OGGQ$REP_1A"
50801 0.01 5001 |<A> OGG$REP_1A 43127 8304 3003835 APR 0% 46.7% 53.3% "CPU +
Wait for CPU" APC 0% 0% 100% "CPU + Wait for CPU" APS (8) 573.3% 0% 220% "CPU
+ Wait for CPU" |<B> OGG$REP_1A APC 786 26382 100.% "CPU + Wait for CPU"
```

iii. Apply Server (APS) processes

The apply server processes are responsible for applying the DML to the database. If constrained by CPU and if the SQL Statistics AWR report shows small numbers of rows per execution, enable Replicat `BATCHSQL` or increase the size of `BATCHTRANSOPS`.

The following is an example of SPADV output when APS is CPU bound.

```
PATH 2 RUN_ID 68 RUN_TIME 2014-JUL-28 13:25:55 CCA Y
|<R> REP_1A 148538 20965388 0 0% 13.3% 73.3% "CPU + Wait for CPU" |<Q>
"SOESMALL"."OGGQ$REP_1A" 148318 0.01 4289 |<A> OGG$REP_1A 141917 15280 -1 APR
0% 0% 100% "CPU + Wait for CPU" APC 66.7% 0% 33.3% "CPU + Wait for CPU" APS
(13) 400% 0% 880% "CPU + Wait for CPU" |<B> OGG$REP_1A APS 1497 21494 100.%
"CPU + Wait for CPU"
```

The following AWR report shows a small number of rows per DML statement because `BATCHSQL` is not enabled.

SQL ordered by Executions

Executions	Rows Processed	Rows per Exec	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Module	SQL Text
20,468,975	20,469,219	1.00	2,508.66	53.4	.6	bu28rqwk7y6rb	GoldenGate	UPDATE /*+ restrict_all_ref_c...
20,466,464	20,466,811	1.00	2,912.26	20.9	5.8	5q8xtx2bhquzj	GoldenGate	INSERT /*+ restrict_all_ref_c...
19,606,938	72,709,149	3.71	2,381.43	37.5	6.7	f4vq0jufrmsx0	GoldenGate	INSERT /*+ restrict_all_ref_c...
19,510,351	65,663,725	3.37	2,478.54	64.8	.1	7kmvq4xrdsuk7	GoldenGate	UPDATE /*+ restrict_all_ref_c...

When `BATCHSQL` is enabled, the rows per execution increases and the elapsed time decreases, as shown in the following report.

SQL ordered by Executions

Executions	Rows Processed	Rows per Exec	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Module	SQL Text
4,855,893	20,477,534	4.22	1,941.08	98.3	.2	bu28rqwk7y6rb	GoldenGate	UPDATE /*+ restrict_all_ref_c...
4,855,639	20,476,399	4.22	1,670.95	39.3	7.8	5q8xtx2bhquzj	GoldenGate	INSERT /*+ restrict_all_ref_c...
4,835,457	72,743,061	15.04	1,453.91	70.4	7.3	f4vq0jufrmsx0	GoldenGate	INSERT /*+ restrict_all_ref_c...
4,832,499	65,837,321	13.62	1,991.06	89.1	.1	7kmvq4xrdsuk7	GoldenGate	UPDATE /*+ restrict_all_ref_c...

It is possible that the apply server processes are contending for database resources, much in the same way a user process does. For example, for I/O, data block accesses, or index block updates. In such scenarios, SPADV or AWR indicate this.

The following example shows SPADV output.

```
PATH 2 RUN_ID 53 RUN_TIME 2014-JUL-30 13:13:24 CCA Y
|<R> REP_1A 51492 7242294 0 0% 86.7% 13.3% "" |<Q> "SOESMALL"."OGGQ$REP_1A"
51398 0.01 5001 |<A> OGG$REP 1A 46399 5193 -1 APR 0% 53.3% 46.7% "CPU + Wait
for CPU" APC 93.3% 0% 6.7% "" APS (11) 26.7% 0% 673.4% "cell single block
physical read" |<B> OGG$REP_1A APS 1440 14146 66.7% "cell single block
physical read"
```

The following report shows AWR background process wait events:

Background Wait Events

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn	% bg time
cell single block physical read	11,847,693	0	13,641	1.15	7.09	38.81
db file parallel write	335,208	0	4,896	14.61	0.20	13.93
write complete waits	3,245	0	3,981	1226.67	0.00	11.33

Note that the apply server processes are considered background processes, and so they are included in the background wait events section of the AWR report.

The following AWR report shows that the SQL being applied by Oracle GoldenGate has the highest I/O wait times.

SQL ordered by User I/O Wait Time

User I/O Time (s)	Executions	UIO per Exec (s)	%Total	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Module	SQL Text
10,173.47	1,652,136	0.01	73.55	14,718.69	18.16	69.12	7kmvq4xrdsuk7	GoldenGate	UPDATE /*+ restrict_all_ref_c...
2,612.04	1,652,174	0.00	18.88	4,662.34	19.12	56.02	5q8xtx2bhquzj	GoldenGate	INSERT /*+ restrict_all_ref_c...
601.62	1,652,152	0.00	4.35	2,273.72	26.88	26.46	f4vq0jufrmsx0	GoldenGate	INSERT /*+ restrict_all_ref_c...
233.50	1,652,159	0.00	1.69	2,189.53	49.35	10.66	bu28rgwk7y6rb	GoldenGate	UPDATE /*+ restrict_all_ref_c...

In such cases, evaluate database or object tuning techniques to improve performance. Refer to *Oracle Database Performance Tuning Guide* at

http://docs.oracle.com/database/121/TGDBA/toc.htm

There are cases when the cause for slowness among the apply server processes is large batch operations against a set of tables that differs from normal OLTP operations, such as large history or reporting tables. Distributing these objects into a separate Replicat process can significantly increase the apply performance.

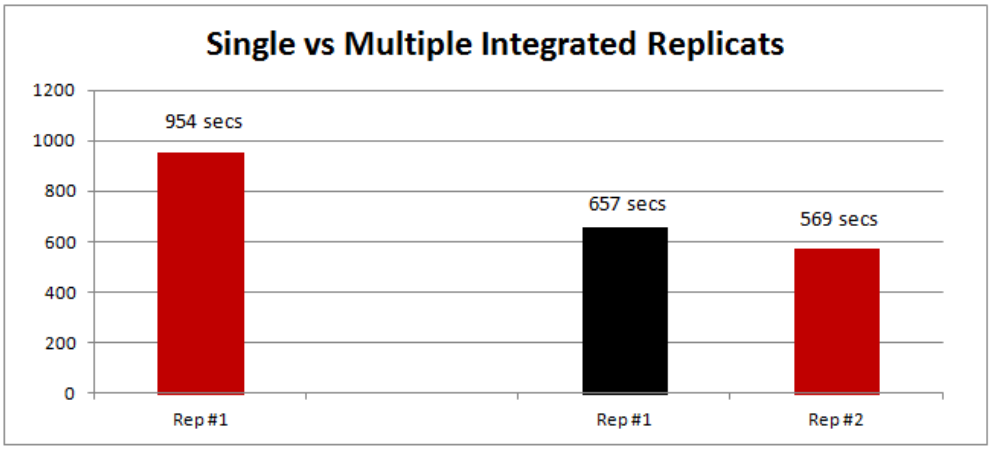
The following example SPADV output shows one apply server process (APS) consuming large amounts of CPU while the others are idle. With integrated Replicat the apply server processes must wait for the large transaction to be applied before the OLTP transactions can continue. There is only one APS process at 100% CPU and nine of them idle (900%).

```
PATH 2 RUN_ID 63 RUN_TIME 2017-JAN-17 20:58:19 CCA Y
|<R> REP_1A 26475 8091556 0 0% 0% 100% "CPU + Wait for CPU" |<Q>
"SOESMALL"."OGGQ$REP_1A" 26326 0.01 4931 |<A> OGG$REP_1A 115113 0.01 1208876
APR 26.7% 0% 73.3% "CPU + Wait for CPU" APC 100% 0% 0% "" APS (10) 900% 0%
100% "CPU + Wait for CPU" |<B> OGG$REP_1A APS 1504 3542 100.% "CPU + Wait for
CPU"
```

Linux `top` output also shows one apply server process consuming much larger amounts of CPU than the other processes.

```
   PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
88004 oracle    20   0 18.1g 9.5g 9.5g R 92.9 10.1   4:11.70 ora_as09_GGA1
85438 oracle    20   0  331m  75m  25m R 82.0  0.1  11:41.74
/u01/oracle/goldengate/HungFix1/replicat PARAMFILE /u01/oracle/golde
85444 oracle    20   0 18.1g 4.1g 4.1g R 79.4  4.4  11:31.97 oracleGGA1
(DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq)))
85500 oracle    20   0 18.4g 4.2g 4.0g R 73.1  4.5  11:37.00 ora_as01_GGA1
86892 oracle    20   0 18.1g 6.1g 6.1g R 32.8  6.5   4:50.19 ora_as08_GGA1
85504 oracle    20   0 18.1g 6.7g 6.7g R 32.2  7.2   7:01.56 ora_as03_GGA1
```

Dividing the work between two integrated Replicat processes shows a 31% performance increase.



NOTE: When using multiple integrated Replicat processes with Oracle GoldenGate Release 12.1.2.0, be sure to apply Oracle GoldenGate performance patch 19261665 to enable faster reading through the trail files containing uninterested transaction data.

Conclusion

Key configuration recommendations were presented for the source and target databases, along with the Oracle GoldenGate Extract, Data Pump, and Replicat processes.

To optimize an Oracle GoldenGate environment, the following crucial pieces of data must be gathered:

- Oracle GoldenGate process lag information
- Report files and error logs
- Active Workload Repository (AWR) and Active Session History (ASH)
- CPU data
- I/O data
- Oracle Streams Performance Advisor (SPADV) for integrated Extract and integrated Replicat
- Integrated Extract and integrated Replicat healthcheck

With all of this information gathered, the presented tuning methodology can be followed to identify and resolve the current cause of lag or latency.

Performance tuning is an iterative process, such that when the cause of lag is resolved, the process begins again with data gathering and analysis.

Using this approach, the previously described performance tuning exercise with a Swingbench OLTP workload demonstrated how Oracle GoldenGate Extract to Replicat performance could be increased by a factor of 20 times. Replicat apply rate of the source redo increased from 3.8MB/second to 78.1MB/second using an integrated Extract and integrated Replicat configuration.

Appendix A – Oracle GoldenGate Performance Information Gathering

To troubleshoot Oracle GoldenGate performance there are several key pieces of information that must be gathered and analyzed. The following information should always be gathered covering the same point in time and on all servers running Oracle GoldenGate processes that are part of the same replication data flow.

Oracle GoldenGate Latency

Latency or lag is the period of time that has passed between when a change (DML or DDL) occurred in the source database and the current point in time. For example, Extract latency is the time that has elapsed since the change occurred to the source table and the time it was extracted and written to the trail file. Conversely, Replicat latency is the time that has elapsed from the source table change to the time it was applied to the target database.

The amount of acceptable lag time is dependent on an agreed upon Service Level Agreement (SLA) that states how much time is allowed to pass between when the data was entered in the source database to the time it appears on the target database. A lag time of 30+ minutes may be acceptable for offloading data for ad-hoc queries but not for a banking application that often requires near zero latency.

When using integrated Extract or integrated Replicat, the latency can be determined by querying the database or by using the GoldenGate GGSCI utility.

Determining Latency for Integrated Extract

» Using a database query:

```
SQL> SELECT capture_name, 86400 *(available_message_create_time -
capture_message_create_time) latency_in_seconds FROM GV$GOLDENGATE_CAPTURE;
```

» Using GGSCI:

```
GGSCI> lag extract <extract_name>
```

Determining Latency for Integrated Replicat

» Using a database query:

```
SQL> SELECT r.apply_name, 86400 *(r.dequeue_time - c.lwm_message_create_time)
latency_in_seconds FROM GV$GG_APPLY_READER r, GV$GG_APPLY_COORDINATOR c WHERE
r.apply# = c.apply# and r.apply_name = c.apply_name;
```

» Using GGSCI:

```
GGSCI> lag replicat <replicat_name>
```

Lag is also reported by the Oracle GoldenGate manager process for both integrated and non-integrated Extract and Replicat. Specify the following GoldenGate manager parameters in the manager parameter file located at `$GG_install_dir/dirprm/mgr.prm`.

```
LAGREPORTMINUTES 5                -- Interval at which lag is checked
LAGINFOMINUTES 5                  -- Threshold at which lag is reported
LAGCRITICALMINUTES 15             -- Critical threshold reporting value
```

The values for these parameters depend on your acceptable lag time.

Latency is written to the `ggserr.log` file that is automatically created in the Oracle GoldenGate installation directory (in `hours:minutes:seconds` format). For example:

```
Manager for Oracle, mgr.prm:  Lag for REPLICAT REP_1A is 00:00:06 (checkpoint
updated 00:00:01 ago).
```

```
2011-09-30 23:48:38  WARNING OGG-00947  Oracle GoldenGate Manager for Oracle,
mgr.prm:  Lag for REPLICAT REP_1B is 00:06:37 (checkpoint updated 00:00:00 ago).
2011-09-30 23:48:38  WARNING OGG-00947  Oracle GoldenGate Manager for Oracle,
mgr.prm:  Lag for REPLICAT REP_1C is 00:05:23 (checkpoint updated 00:00:04 ago).
```

If the latency is higher than what is acceptable, gather the recommended data listed in this section and follow the performance tuning methodology described below to determine and resolve the performance bottleneck.

Oracle GoldenGate Report Files and Error Logs

Each Extract, Data Pump, and Replicat process generates an ongoing report file with the following information:

- Parameters in use
- Table and column mapping
- Runtime messages and errors
- Runtime statistics

To monitor Oracle GoldenGate performance, set the `REPORTCOUNT` parameter in the GoldenGate process parameter file to report real-time statistics.

```
REPORTCOUNT EVERY 15 MINUTES, RATE
```

This parameter should be set for all Extract, Data Pump, and Replicat processes to a suitable interval rate (recommended maximum value of `15 MINUTES`). The report file contains entries to show the current processing rates. For example:

```
 13688414 records processed as of 2014-07-28 22:17:17 (rate 114065, delta 132143)
141743251 records processed as of 2014-07-28 22:32:19 (rate 131239, delta 129957)
```

The Oracle GoldenGate process report files are located in the `$GG_install_dir/dirrpt` directory.

This example shows that a Replicat process applied 132,143 and 129,957 records in the two sample intervals, which are fifteen minutes apart.

Both the processing rates and the lag should be continually monitored for sudden changes to Oracle GoldenGate performance levels.

Automatic Workload Repository and Active Session History

Automatic Workload Repository (AWR) is a good starting point for identifying general database performance issues which can provide initial indicators to help locate problems with Extract or Replicat processes. Using AWR, you can easily determine if the bottlenecks are inside or outside of the database.

After analyzing the relevant AWR reports, use Active Session History (ASH) to look at more detailed information on particular sessions in the database, like those of a Replicat process. Each Replicat and Extract process is given a unique SQL module ID that can be used for identification in AWR and ASH reports.

For example:

Elapsed Time (s)	Executions	Elapsed Time per Exec (s)	%Total	%CPU	%IO	SQL Id	SQL Module	SQL Text
2,338.40	10,697	0.22	20.30	11.45	89.86	bkpt6p42st6dq	OGG-REP_1F2-OPEN_DATA_SOURCE	INSERT INTO "PSFT"."PS_TAX_BAL...
1,673.20	16,896	0.10	14.53	19.63	83.02	6xxrj1pzqz5s2	OGG-REP_1A-OPEN_DATA_SOURCE	INSERT INTO "PSFT"."PS_EARNING...

The Replicat names in this example are `REP_1F2` and `REP_1A`.

An ASH report can be created for a specific Replicat process by running the `$ORACLE_HOME/rdbms/admin/ashrpti.sql` script and using the SQL module name. Use the generated report to further investigate why a particular Replicat process is not performing as expected.

CPU Data

Gathering CPU data with operating system tools like `top` is essential to see if Oracle GoldenGate processes are bottlenecked on CPU rather than I/O or some other database process. As a general rule, if the Replicat process is not on CPU for at least 40% of the time, then it is constrained by something else such as I/O or database processing of the replicated SQL statements. It is important to gather CPU data that shows each thread of execution within a process. For example, an Extract process uses multiple threads, and it is important to be able to identify each thread instead of the entire process consuming CPU.

The following example shows the result of using `top` without any thread-specific parameters.

```
top - 12:51:02 up 182 days, 20:51,  3 users,  load average: 0.09, 0.14, 0.09
Cpu(s): 5.7%us,  0.9%sy,  0.0%ni, 93.4%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  99060552k total, 42003164k used, 57057388k free,  1219612k buffers
Swap: 25165816k total,        0k used, 25165816k free,  8591940k cached

   PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 76001 oracle    20   0  739m  68m  32m R 117.7  0.1  15:06.70
/goldengate/latest/extract PARAMFILE /goldengate/latest/dirprm/ext_1a.prm
```

The Extract process in the above example is using 117.7% CPU, so it is not possible to verify if one of the process threads is bottlenecked on CPU. Instead, use `top` parameters to show process threads (`top -H` for Linux) as shown below.

```
top - 12:51:45 up 182 days, 20:51,  3 users,  load average: 0.19, 0.16, 0.10
Cpu(s): 6.5%us,  1.2%sy,  0.0%ni, 92.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  99060552k total, 42148560k used, 56911992k free,  1219612k buffers
Swap: 25165816k total,        0k used, 25165816k free,  8583880k cached

   PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 76001 oracle    20   0  739m  68m  32m R 81.8  0.1   8:32.61
/goldengate/latest/extract PARAMFILE /goldengate/latest/dirprm/ext_1a.prm
 76016 oracle    20   0  739m  68m  32m R 48.8  0.1   5:03.39
/goldengate/latest/extract PARAMFILE /goldengate/latest/dirprm/ext_1a.prm
```

I/O Data

Gathering I/O data using operating system tools such as `iostat` or `OSWatcher` is crucial to understanding where the bottlenecks on I/O originate. For the source environment, you need to consider both reads from the redo log files

and concurrent reads and writes to and from the trail files by Extract and Data Pump processes. On the target environment, the concurrent access to the trail files by Data Pump and one or more Replicat processes must be monitored. As with normal database tuning, the database I/O should be monitored, and these results can be used along with AWR and ASH to identify and resolve bottlenecks.

For information about OSWatcher, refer to the OSWatcher Analyzer User's Guide at

https://support.oracle.com/oip/faces/secure/km/DocumentDisplay.jspx?id=461053.1&h=Y

Oracle Streams Performance Advisor (Integrated Extract and Integrated Replicat)

The Oracle Streams Performance Advisor (SPADV) enables monitoring of the integrated GoldenGate server processes which are used by integrated Extract and integrated Replicat, and provides information about how these processes are performing.

SPADV statistics are collected and analyzed using the `UTL_SPADV` package.

To install SPADV:

1. Grant the following privileges to a designated Oracle GoldenGate administrator database user if it hasn't been done already.

```
SQL> exec DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE('<db user name>');
```

2. Connect to the database with the user name that was granted permissions in Step 1.

3. Run the `utlspadv.sql` script. For example:

```
SQL> @$ORACLE_HOME/rdbms/admin/utlspadv.sql
```

Oracle recommends that you gather statistics for a 30-60 minute time period during which you are troubleshooting performance. It is also recommended that you gather statistics during a 30-60 minute time period where performance is good, to serve as a baseline comparison.

To gather statistics every 15 seconds, run the following SQL*Plus command as the Oracle GoldenGate administrator.

```
SQL> exec UTL_SPADV.START_MONITORING(interval=>15);
```

To stop statistics gathering, run the following command.

```
SQL> exec UTL_SPADV.STOP_MONITORING;
```

Run the following commands to determine if the monitoring job is currently running.

```
SET SERVEROUTPUT ON
DECLARE
  is_mon   BOOLEAN;
BEGIN
  is_mon := UTL_SPADV.IS_MONITORING(
              job_name    => 'STREAMS$_MONITORING_JOB',
              client name => NULL);
  IF is_mon=TRUE THEN
    DBMS_OUTPUT.PUT_LINE('The monitoring job is running.');
  ELSE
```

```
    DBMS_OUTPUT.PUT_LINE('No monitoring job was found.');
  END IF;
END;
/
```

To create a text report of SPADV statistics after monitoring for a period of time, from SQL*Plus as the Oracle GoldenGate administrator, run the following:

```
spool /tmp/spadv.txt
begin
     utl_spadv.show_stats(path_stat_table=>'STREAMS$_PA_SHOW_PATH_STAT',
                          bgn_run_id=> 1,
                          end_run_id=> 9999,
                          show_legend=> TRUE);
end;
```

After the reports have been generated, Oracle recommends purging the SPADV statistics using the following command:

```
SQL> exec UTL_SPADV.STOP_MONITORING(PURGE=>TRUE);
```

Appendix C contains a shell script example that displays SPADV statistics in real time so that you can monitor the GoldenGate processes during the monitoring period.

The following example shows the output for integrated Extract.

```
PATH 2 RUN_ID 59 RUN_TIME 2013-MAR-21 15:20:34 CCA Y
|<C> OGG$CAP_EXT_1A 129882 129851 57 LMR 0% 73.3% 26.7% "CPU + Wait for CPU"
LMP (1) 0% 0% 100% "CPU + Wait for CPU" LMB 80% 0% 20% "CPU + Wait for CPU" CAP
46.7% 0% 53.3% "CPU + Wait for CPU" |<Q> "STREAMSADMIN"."OGG$Q_EXT_1A" 129844
0.01 0 |<A> OGG$EXT_1A 0.01 0.01 0 |<B> NO BOTTLENECK IDENTIFIED
```

The general format for each process is:

```
<process name> <idle %> <flow control %> <top event %> <top event name>
```

The preceding integrated Extract example shows the following statistics:

- LogMiner captured an average of 128,882 messages per second.
- The LogMiner latency is currently 57 seconds.
- The LogMiner reader (LMR) server process spent:
  - 0% of its time idle
  - 73.3% of its time in flow control (waiting for the next process in the chain (LMP))
  - 26.7% of its time consuming or waiting for CPU
- The LogMiner preparer (LMP) server process spent:
  - 0% of its time idle
  - 0% of its time in flow control
  - 100% of its time consuming or waiting for CPU
- The LogMiner builder (LMB) server process spent:
  - 80% of its time idle
  - 0% of its time in flow control

    - 20% of its time consuming or waiting for CPU
- The capture process (CAP) session spent:
    - 46.7% of its time idle
    - 0% of its time in flow control
    - 53.3% of its time consuming or waiting for CPU

The SPADV statistics clearly indicate if any of the LogMiner server processes are causing performance bottlenecks. In the next example, the LogMiner preparer (LMP) process is bottlenecked on CPU.

The output from SPADV for integrated Replicat has a similar format to integrated Extract. The following example shows the output for integrated Replicat (excerpt using an Oracle Database 12*c* database).

```
PATH 2 RUN ID 69 RUN TIME 2014-JUL-15 08:34:57 CCA Y
|<R> REP_1A 111937 15724041 0 0% 31.3% 50% "CPU + Wait for CPU" |<Q>
"SOESMALL"."OGGQ$REP_1A" 111636 0.01 4870 |<A> OGG$REP_1A 114395 11729 -1 APR 0%
12.5% 87.5% "CPU + Wait for CPU" APC 56.3% 0% 43.8% "CPU + Wait for CPU" APS
(12) 237.5% 0% 931.3% "CPU + Wait for CPU" |<B> OGG$REP_1A APS 1374 47804 100.%
"CPU + Wait for CPU"
```

The output shows the following statistics:

- The apply rate at this sample time is 114,395 messages per second by the apply process, `OGG$REP_1A`.
- The apply latency is shown as -1 which indicates, as does zero, that there is no latency.
- The Replicat process spent:
    - 0% of its time idle
    - 31.3% of its time in flow control (waiting for another process further along in the chain)
    - 50% of its time consuming or waiting for CPU
- The apply reader (APR) process spent:
    - 0% of its time idle
    - 12.5% of its time in flow control (waiting on another process further along in the chain)
    - 87.5% of its time consuming or waiting for CPU
- The apply coordinator (APC) process spent:
    - 56.3% of its time idle
    - 0% of its time in flow control
    - 43.8% of its time consuming or waiting for CPU
- The apply server (APS) processes spent:
    - 237.5% of its time idle
    - 0% of its time in flow control
    - 931.3% of its time consuming or waiting for CPU
    - There are twelve APS processes; therefore, 931.1% of twelve processes equates to 77.6% of total time.
- A single APS process is identified as the bottleneck with 100% CPU consumption or time spent waiting for CPU.

The integrated Replicat SPADV clearly shows there is a bottleneck on apply server processes on CPU. To learn how to resolve such bottlenecks see the methodology described in “Oracle GoldenGate Performance Tuning Methodology” on page 17.

Integrated Capture and Integrated Replicat Healthcheck

The integrated capture and integrated Replicat healthcheck is a report that shows the current status of an Oracle GoldenGate integrated capture or integrated Replicat configuration. The report is divided into three main sections.

- Configuration - reports definitions relevant for Oracle GoldenGate integrated Extract and integrated Replicat.
- Analysis - provides output for the checks done by the healthcheck.
- Statistics - reports statistics for those elements of integrated capture and integrated Replicat that are enabled.

Healthcheck is a statically generated report, so it only reflects the status at one point in time. Oracle recommends gathering two or three healthcheck reports at several minute intervals to make sure that the components are flowing correctly.

For instructions about downloading and for more information about using the healthcheck script, refer to MOS Note 1448324.1 at

https://support.oracle.com/oip/faces/secure/km/DocumentDisplay.jspx?id=1448324.1&h=Y

Appendix B – Considerations for Non-Integrated GoldenGate Replicat Processes

If you are not using Oracle GoldenGate integrated Replicat there are some additional considerations that must be understood.

Use of BATCHSQL

By default, non-integrated Replicat operates in normal mode, where each row change is made one row at a time. Commits are issued based on the setting of the `GROUPTRANSOPS` parameter (which defaults to `1000`). After approximately 1000 SQL operations, a `COMMIT` command is issued. Replicat accumulates operations from source transactions, in transaction order, and applies them as a group within one transaction on the target database. The `GROUPTRANSOPS` parameter sets a minimum value rather than an absolute value to avoid dividing source transactions. Replicat waits until it receives all operations from the last source transaction in the group before applying the target transaction.

By enabling `BATCHSQL` mode, Replicat batches together SQL statements that affect the same table, operation type (`INSERT`, `UPDATE` or `DELETE`), and column list and applies them together as an array operation. By using array operations, apply rates generally increase because there is significantly less CPU utilization per row.

The following is an example of an insert-only workload with Replicat in normal mode (taken from an AWR report).

SQL ordered by Executions

Executions	Rows Processed	Rows per Exec	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Module	SQL Text
43,068,967	43,068,967	1.00	2,221.71	100	0	19rchjjpb3462	OGG-REP_1A-OPEN_DATA_SOURCE	INSERT INTO "SOESMALL"."ORDER_...

You can see that single row operations are carried out because the value of the Rows per Exec column is 1.00.

In contrast, the following example uses the `BATCHSQL` parameter with a default `OPSPERBATCH` value of `1200`.

SQL ordered by Executions

Executions	Rows Processed	Rows per Exec	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Module	SQL Text
43,003	43,069,020	1,001.54	461.05	99.9	0	19rchjjpb3462	OGG-REP_1A-OPEN_DATA_SOURCE	INSERT INTO "SOESMALL"."ORDER_...

The value of the Rows per Exec column has increased to approximately 1000 and there is a 4.8 times reduction in elapsed time and CPU time for these inserts.

In most cases, Oracle recommends that you leave the setting of the `OPSPERBATCH` parameter at the default value of `1000`. To enable `BATCHSQL` for a Replicat, add the `BATCHSQL` parameter to the Replicat parameter file.

When `BATCHSQL` is enabled for a non-integrated Replicat, Oracle recommends regularly checking the process report file and statistics to make sure that few transactions are reverting back to normal mode (non-batched) because an exception was encountered. When many exceptions occur, apply performance can suffer because of the rolling back of the batched transaction and reapplying it in normal mode. To determine how many batched transactions are being aborted, use the following GGSCI command:

```
GGSCI> send <replicat_name> report;
```

Then, look at the latest information in the Replicat report file located in the `dirrpt` directory. For example:

```
BATCHSQL statistics:
              Batch operations:  21322428
                       Batches:     21294
              Batches executed:     21294
                        Queues:     21294
              Batches in error:         8
         Normal mode operations:      8397
    Immediate flush operations:         0
                  PK collisions:     14381
                  UK collisions:         0
                  FK collisions:         0
```

In the preceding example, there are 8 transaction batches that encountered an exception, with 14381 primary key collisions.

The following example Replicat report file shows the reason for the exceptions.

```
2014-07-15 11:46:14  WARNING OGG-00869  Aborting BATCHSQL transaction. Database
error 60 (OCI Error ORA-00060: deadlock detected while waiting for resource
(status = 60), SQL <UPDATE "SOESMALL"."INVENTORIES" x SET x."QUANTITY_ON_HAND" =
:a2 WHERE x."PRODUCT_ID" = :b0 AND x."WAREHOUSE_ID" = :b1>).
2014-07-15 11:46:14  WARNING OGG-01137  BATCHSQL suspended, continuing in normal
mode.
2014-07-15 11:46:14  WARNING OGG-01003  Repositioning to rba 297226345 in seqno 1.
2014-07-15 11:46:14  INFO    OGG-01139  BATCHSQL resumed, recovered from error.
```

When these exceptions occur they should be investigated and resolved before changing the `BATCHSQL` configuration.

When using the `BATCHSQL` or `GROUPTRANSOPS` parameters, SQL operations from different transactions are merged into larger transactions while maintaining transactional order. If the target transactions must match the source transactions (for example, the number of DMLs per commit), then set `GROUPTRANSOPS=1`, which may limit the Replicat performance for small transactions.

The maximum size of each statement batch is controlled by the `BATCHSQL OPSPERBATCH` parameter. The default size of `1000` is adequate in most cases, but changing the batch size may result in performance gains. Setting the batch size too low or too high can result in performance degradation. When changing the `BATCHSQL` parameter do so in a controlled manner, so performance with the old and new settings can be accurately compared.

Dividing Workload Between Multiple Replicats

If you are using non-integrated Replicat, it is recommended that you use the Coordinated Replicat feature to divide work amongst Replicat processes. Coordinated Replicat was introduced with Oracle GoldenGate 12.1.2 release, providing an easier way to manually partition the workload to apply high volume transactions concurrently. Instead of creating multiple Replicat processes, each with their own parameter file, listing which objects to apply using `RANGE` parameter, a single parameter file is used to create a number of coordinated Replicats. A coordinator thread process coordinates transactions across Replicats that require coordination, such as DDL and primary key updates with `THREADRANGE` partitioning. Such a transaction is executed as one transaction in the target with full synchronization; it waits until all prior transactions are applied, and all transactions after this barrier transaction must wait until this barrier transaction is applied.

It is recommended that you use Coordinated Replicat when there are a few tables that are modified with large transactions such that applying the transactions is causing Replicat slowness.

For more information about Coordinated Replicat refer to *Administering Oracle GoldenGate for Windows and UNIX* at

http://docs.oracle.com/goldengate/c1221/gg-winux/GWUAD/GUID-3388C553-994A-4ECD-95C8-A54FC7446E67.htm#GWUAD953

Before configuring multiple integrated or non-integrated Replicat processes, it is important to identify the key characteristics of the data being replicated.

1. Heavily accessed tables (with DML)

   When a single Replicat process cannot keep apply latency low enough, Oracle recommends that you evenly distribute the data being applied across a number of Replicat processes until the latency is near zero or within an acceptable time. To do this, you must identify the heavily manipulated tables. There are three methods available to identify such tables:

   a. Normally done during the Oracle GoldenGate testing phase before implementation in a production environment, configure Extract to capture the schema or tables required for replication with the `TESTMAPPINGSPEED` parameter. This parameter stops Extract from creating any trail files, but allows you to see the type and volume of data captured, test the Extract configuration, and determine the overhead cost of mining the log files on the source database. After Extract has run long enough to capture a suitable amount of workload, stop Extract to create an Extract report. The report file is created in the `dirrpt` directory of the Oracle GoldenGate installation home. The report includes a list of tables with the number of inserts, updates, and deletes carried out against each table. For example:

```
      From Table PSFT.PS_PAY_LINE:
       #                  inserts:    750720
       #                  updates:   1501440
       #                  deletes:         0
       #                 discards:         0
From Table PSFT.PS_PAY_EARNINGS:
       #                  inserts:   2352256
       #                  updates:   4204032
       #                  deletes:    250240
       #                 discards:         0
From Table PSFT.PS_PAY_OTH_EARNS:
       #                  inserts:   1901824
       #                  updates:   1651584
       #                  deletes:    250240
       #                 discards:         0
```

   b. Use the Oracle GoldenGate `STATS EXTRACT` command to gather table statistics for a currently running Extract process. While Extract is running, use the following script to retrieve table statistics for the latest 15 minute period:

```
#!/bin/bash

cd <Oracle GoldenGate Install Home>

./ggsci <<!EOT > /tmp/table_stats.out
   stats extract ext_1a, reset
   pause 900
   stats extract ext_1a, total, latest
!EOT
```

The output produced includes table statistics since Extract was started and also since `reset` was issued 15 minutes before printing the statistics. For example:

```
Start of Statistics at 2013-02-18 15:41:29.
Output to /u01/goldengate/latest/dirdat_os/aa:
Extracting from SOESMALL.ORDERS to SOESMALL.ORDERS:

*** Total statistics since 2013-02-18 14:08:19 ***
        Total inserts                                 2411804.00
        Total updates                                 2411803.00
        Total deletes                                       0.00
        Total discards                                      0.00
        Total operations                              4823607.00

*** Latest statistics since 2013-02-18 15:26:28 ***
        Total inserts                                  224076.00
        Total updates                                  224075.00
        Total deletes                                       0.00
        Total discards                                      0.00
        Total operations                               448151.00
```

c. Use the Oracle GoldenGate `logdump` utility to retrieve the table statistics from one or more trail files. When one or more trail files have been created, use the following commands to retrieve the table statistics:

```
% cd <Oracle GoldenGate Install Home>
% ./logdump
Logdump> count detail <trail_file directory>/<trail file name>
```

Use a wildcard character to retrieve the count from more than one file:

```
Logdump> count detail <trail_file_directory>/aa00000*
```

Example output:

```
SOESMALL.INVENTORIES                              Partition 4
Total Data Bytes         781788504
  Avg Bytes/Record              42
FieldComp                 18614012
After Images              18614012

SOESMALL.ORDERS                                   Partition 4
Total Data Bytes        1140800152
  Avg Bytes/Record              98
Insert                     5790864
FieldComp                  5790863
After Images              11581727

SOESMALL.ORDER_ITEMS                              Partition 4
Total Data Bytes        1439690630
  Avg Bytes/Record              70
Insert                    20567009
After Images              20567009
```

For detailed information about using the `logdump` utility to determine table DML rates, refer to MOS Note 1301300.1 at

https://support.oracle.com/oip/faces/secure/km/DocumentDisplay.jspx?id=1301300.1&h=Y

Use the total number of DML statements for each table to divide the tables among the Replicat processes. This can be made easier using the Perl code provided in MOS Note 2224542.1 at

https://support.oracle.com/CSP/main/article?cmd=show&type=NOT&id=2224542.1

The number of Replicat processes to configure is determined by using an iterative process of adding Replicat processes until the latency reaches an acceptable number without causing I/O contention on reading the trail files, or without causing other database performance issues. Start with a single Replicat process and measure how it performs. If performance is not acceptable (after using `BATCHSQL`, if possible), distribute the tables among two or three Replicat processes or coordinated Replicat threads and retest the performance. Continue this exercise until a suitable performance level and latency time is reached.

If there are a small number of tables that contain a large percentage of DML which, after dividing into their own Replicat processes, are still not applying the data fast enough, these tables can be further distributed among coordinated Replicat threads.

For example, distributing a table between two coordinated Replicat threads would use the following MAP parameter:

```
MAP SOESMALL.ORDER_ITEMS , TARGET soesmall.ORDER_ITEMS, THREADRANGE (1-2);
```

For more information about using the `THREADRANGE` parameter to distribute work to coordinated Replicat threads, refer to *Reference for Oracle GoldenGate for Windows and UNIX* at

http://docs.oracle.com/goldengate/c1221/gg-winux/GWURF/GUID-C2356234-3780-48EE-9E7A-F21DC352638C.htm#GUID-C2356234-3780-48EE-9E7A-F21DC352638C__BABIEAFI

2. Referential integrity between tables

To ensure data integrity, parent and child tables with referential integrity relationships should be processed by the same Replicat process. For tables that are not part of referential integrity constraints (for example, Peoplesoft Payroll tables), assigning tables among multiple Replicat processes becomes an easier task by evenly distributing the load among each Replicat process.

3. Handling of DDL statements

To avoid locking conflicts between Replicat processes, it is very important to understand the nature of DDL statements that occur against replicated objects. You must apply DDL with the same Replicat process that is applying DML for the table. If not configured this way, the Replicat processes can abort when a DDL statement times out waiting for another process to finish applying DML to the same table. There are two ways to avoid this issue:

a. Use coordinated Replicat (for non-integrated Replicat only). Coordinated Replicat is a multithreaded process that applies transactions in parallel instead of serially. Each thread handles all of the filtering, mapping, conversion, SQL construction, and error handling for its assigned workload. A coordinator thread coordinates transactions across threads to account for dependencies, and also ensures that DDL is applied in a synchronized manner preventing DML from occurring on the same object at the same time. For more information about coordinated Replicat, refer to *Administrating Oracle GoldenGate for Windows and UNIX* at

http://docs.oracle.com/goldengate/c1221/gg-winux/GWUAD/GUID-3388C553-994A-4ECD-95C8-A54FC7446E67.htm#GWUAD953

b. If you are not using coordinated Replicat, Oracle recommends that you avoid using the `@RANGE` function to divide a table among Replicat processes if DDL is also applied to the table. It is not possible to predict if all of the DML is completed before the DDL is applied. To help alleviate the DDL timeout issue, use the DDL `EXCLUDE` or `INCLUDE` parameters to instruct the Replicat process to which tables DDL can be applied.

   For more details about replicating DDL statements, refer to *Installing and Configuring Oracle GoldenGate for Oracle Database* at

   http://docs.oracle.com/goldengate/c1221/gg-winux/GIORA/GUID-0A230601-A447-499C-B31F-C9431E1FF034.htm#GIORA285

Appendix C – Displaying Real-time SPADV Statistics

The following example shell script program displays the Oracle Streams Performance Advisor (SPADV) statistics in real time, once monitoring has been started.

```
#!/bin/bash

# Set the Oracle environment variables
export ORACLE_SID=GGS1
export ORACLE_HOME=/u01/app/oracle/product/12c
export PATH=$PATH:$ORACLE_HOME/bin

sqlplus -s <GG admin user>/<GG admin passwd> <<!EOS
set feedback off serveroutput on

-- First need to show first stat line with the legend:
begin
     utl_spadv.show_stats(path_stat_table=>'STREAMS\$_PA_SHOW_PATH_STAT',
                          bgn_run_id=> -1,
                          end_run_id=> -1,
                          show_legend=> TRUE);
end;
/
!EOS

sleep 15

-- Now loop through showing results every 15 seconds, until CTRL-C is issued
d=0
while [ $d -lt 1 ];
do
  date

  sqlplus -s streamsadmin/streamsadmin <<!EOS
  set feedback off serveroutput on

  begin
     utl_spadv.show_stats(path_stat_table=>'STREAMS\$_PA_SHOW_PATH_STAT',
                          bgn_run_id=> -1,
                          end_run_id=> -1,
                          show_legend=> FALSE);
  end;
/

!EOS

  sleep 15

done
```

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

Oracle GoldenGate Performance Best Practices

May 2017

Author: Stephan Haisley