

Oracle GoldenGate Performance Best Practices

ORACLE WHITE PAPER | NOVEMBER 2014

Table of Contents	
Introduction	1
Oracle Software	2
Database Configuration	2
Source Database	2
Target Database	5
Oracle GoldenGate Configuration	8
Extract Configuration	8
Data Pump Configuration	8
Replicat Configuration	11
Database File System (DBFS) Configuration	19
Data Gathering for Oracle GoldenGate Performance	20
Oracle GoldenGate Latency	20
For integrated Extract:	20
For integrated Replicat:	20
Oracle GoldenGate Report Files and Error Logs	21
Automatic Workload Repository and Active Session History	21
CPU Data	22
I/O Data	23
Oracle Streams Performance Advisor (Integrated Extract and Integrated Replicat)	23
Integrated Capture and Integrated Replicat Healthcheck	26
Oracle GoldenGate Performance Tuning Methodology	26

Oracle GoldenGate Performance Swingbench Case Study	33
Workload Description	33
Environment Configuration	34
Oracle GoldenGate Performance	35
Conclusion	37
Appendix A – Displaying Real-time SPADV Statistics	38
Appendix B – Dividing Tables Between Replicat Processes Using Perl Code	39

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, 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. Some examples of performance throughput improvements when applying the Oracle GoldenGate tuning methodology achieved in our MAA validation are also presented. Non-Oracle databases are not covered in this paper.

The key Oracle GoldenGate configuration best practices are:

- Enable supplemental logging to ensure the correct data is replicated to the target database.
- Configure Oracle GoldenGate Extract in integrated capture mode to take advantage of the database LogMiner server functionality and simplify management.
- Configure Oracle GoldenGate integrated Replicat processes to leverage the apply processing functionality that is available within the Oracle database.
- Configure multiple parallel Replicat processes using batched SQL for improved apply performance.

Refer to *"Oracle GoldenGate on Oracle Exadata Database Machine Configuration"* white paper 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-wp-gg-oracledbm-128760.pdf>

Oracle Software

Use Oracle GoldenGate Release 12.1.2 or later to take advantage of the increased functionality and enhanced performance features. 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.

To make use of integrated Extract, you must use an Oracle database release of 11.2.0.3 or later. The specific patch numbers required for 11.2.0.3 are listed in My Oracle Support (MOS) Note 1557031.1. Extract in integrated capture mode can also be used to capture changes from Oracle releases starting with 10.2.0.4 with downstream mining using a release 11.2.0.3 or later mining database.

To make use of integrated Replicat, use an Oracle database release of 11.2.0.4 or later.

Database Configuration

This section contains 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.

Source Database

The source database should be configured with the following:

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 and then you can 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 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. Add supplemental logging at the schema level using the Oracle GoldenGate command `ADD SCHEMATRANDATA`.

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

<http://docs.oracle.com/goldengate/1212/gg-winux/GIORA.pdf>

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. If you are using Extract in classic mode (non-integrated capture mode), the Streams pool is not necessary.

The size requirement of the Streams pool for Extract in integrated capture mode is based on two integrated capture mode parameters:

- `MAX_SGA_SIZE` – controls the amount of shared memory used by the LogMiner server. The default value is 1GB and, in most cases, this is adequate. This is not the same as the database initialization parameter `SGA_MAX_SIZE`.

By monitoring Automatic Workload Repository (AWR) reports during peak times for a high number of background process waits on 'LogMiner preparer: memory' or 'LogMiner reader: buffer' with high *Avg wait (ms)* times (>5ms) and high *% bg time* (>25%), increasing the `MAX_SGA_SIZE` parameter by 25% can

improve Extract performance. For example, the following AWR report shows that the `MAX_SGA_SIZE` parameter is not set high enough (Background Wait Events).

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

Once the `MAX_SGA_SIZE` parameter is increased, the amount of wait time drops significantly and no longer causes a performance slowdown.

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn	% bg time
LogMiner reader: buffer	743,701	9	1,375	2	81.15	19.46
LogMiner preparer: memory	1,538	43	786	511	0.17	11.13

Set the `MAX_SGA_SIZE` parameter using the following Extract parameter. This example sets the `MAX_SGA_SIZE` parameter to 2G:

```
TRANLOGOPTIONS INTEGRATEDPARAMS (MAX_SGA_SIZE 2048)
```

It is not recommended to use a `MAX_SGA_SIZE` setting that exceeds 3.5G.

- PARALLELISM** – controls the number of LogMiner preparer (LMP) server processes used by the LogMiner server. The default value for Oracle Database Enterprise Edition is 2 and is adequate for most workloads. Oracle Database Standard Edition defaults to 1 and cannot be increased. To identify when to increase the parallelism parameter, use the Oracle Streams Performance Advisor (SPADV) and evaluate if all LogMiner preparer (LMP) processes are nearing 100% CPU. When the parallelism parameter has been set to 1, you are overriding the default of 2 as shown in the following example:

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

While the LogMiner preparer (LMP) process is peaking at 100% CPU, the processes that are next in the chain of data movement have plenty of available bandwidth to handle more work. Both the LogMiner builder (LMB) and the outbound capture process (CAP) show plenty of idle time (80% and 46.7%, respectively). By increasing the capture parallelism parameter, it is possible to increase throughput and reduce the idle times as shown in the following example:

```
PATH 2 RUN_ID 52 RUN_TIME 2013-MAR-21 15:50:35 CCA Y
|<C> OGG$CAP_EXT_1A 169413 169361 1 LMR 0% 86.7% 13.3% "" LMP (2) 6.7% 53.3%
140% "CPU + Wait for CPU" LMB 66.7% 0% 33.3% "CPU + Wait for CPU" CAP 0% 0% 100%
"CPU + Wait for CPU" |<Q> "STREAMSADMIN"."OGG$Q_EXT_1A" 169383 0.01 0 |<A>
OGG$EXT_1A 0.01 0.01 0 |<B> NO BOTTLENECK IDENTIFIED
```

The LogMiner preparer processes are now using 140% CPU (40% more than before) and extracting at a 30% higher rate (169,361 messages per second versus 129,851).

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

```
(MAX_SGA_SIZE * PARALLELISM) + 25% head room
```

For example, using the default values for the `MAX_SGA_SIZE` and `PARALLELISM` parameters:

```
( 1GB * 2 ) * 1.25 = 2.50GB
```

```
STREAMS_POOL_SIZE = 2560M
```

5. 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 additional information on the `UTL_SPADV` package, refer to *Oracle Database PL/SQL Packages and Types Reference* at the following URL:

http://docs.oracle.com/cd/E16655_01/appdev.121/e17602/u_spadv.htm#BABBEFCI

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

<http://docs.oracle.com/goldengate/1212/gg-winux/GIORA.pdf>

Target Database

The target database should be configured with the following:

1. Run the database in ARCHIVELOG mode

Although Oracle GoldenGate does not require the target database to 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 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.

Refer to the previous section titled “Source Database” for instructions on how to enable force logging mode.

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. Instead, set the `STREAMS_POOL_SIZE` initialization parameter using the following calculation:

```
(MAX_SGA_SIZE (defaults to 1G) * number of integrated Replicat processes) + 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
```

If the Streams pool is not large enough for integrated Replicat, when it runs out of memory, the Replicat process stalls with most of the apply processes becoming idle.

The Replicat process report file shows a gradual slowing down until no further entries are being logged. For example:

```
Opened trail file /home/goldengate/latest/dirdat_os/aa000032 at 2014-07-09 09:52:39  
    97280095 records processed as of 2014-07-09 09:53:36 (rate 59196,delta 45902)  
Switching to next trail file /home/goldengate/latest/dirdat_os/aa000033 at 2014-07-09  
09:53:49 due to EOF, with current RBA 499999850  
Opened trail file /home/goldengate/latest/dirdat_os/aa000033 at 2014-07-09 09:53:49  
    99403194 records processed as of 2014-07-09 09:54:55 (rate 57691,delta 26642)  
    101226093 records processed as of 2014-07-09 10:03:36 (rate 45113,delta 3500)
```

The Oracle Streams Performance Advisor shows that most processes are in an idle state and the Replicat (<R>) process is at 100% flow control, further indicating there is a problem with the Apply server processes. For example:

```
PATH 2 RUN_ID 209 RUN_TIME 2014-JUL-09 22:43:01 CCA Y  
|<R> REP_1A 0.01 0.01 0 0% 100% |<Q> "SOESMALL"."OGGQ$REP_1A" 0.01 0.01 1 |<A>  
OGG$REP_1A 0.01 0.01 0 APR 100% 0% 0% "" APC 100% 0% 0% "" APS (6) 600% 0% 0% ""  
|<B> NO BOTTLENECK IDENTIFIED
```

Querying the `V$GG APPLY_RECEIVER` view shows the out-of-memory condition. For example:

```
SQL> SELECT state FROM V$GG APPLY_RECEIVER;  
  
STATE  
-----  
Waiting for memory
```

The integrated healthcheck report also highlights an out-of-memory condition in the Summary section. For example:

Summary of GoldenGate Integrated Replicats configured in this database ([ConfigDetails](#) [StatsDetails](#))

Current Time	Replicat Name	Server Name	Apply User	Status	Registered	Last DDL Time	CON_ID	Current Receiver State	Current Coordinator State	Unassigned Complete Txns
2014-07-09 20:23:48	REP_1A	OGG\$REP_1A	SOESMALL	ATTACHED	2014-07-09 09:25:58	2014-07-09 09:26:10	0	Waiting for memory	IDLE	0

When this situation occurs and the `MAX_SGA_SIZE` parameter has been configured, stop the Replicat process, increase the `MAX_SGA_SIZE` parameter by 25%, then restart the Replicat. Remember to increase the `STREAMS_POOL_SIZE` initialization parameter to hold the larger `MAX_SGA_SIZE` value.

If the `MAX_SGA_SIZE` parameter is not configured, stop the Replicat process and increase the `STREAMS_POOL_SIZE` initialization parameter by 25%. The Streams pool parameter is dynamic, therefore, it may be possible to change its value while the database is open and while there is unallocated memory available in the SGA. For example:

```
-- First make sure there is enough free space to increase the Streams pool
SQL> SELECT CURRENT_SIZE/1024/1024 FREESPACE_MB FROM V$SGA_DYNAMIC_FREE_MEMORY;

FREESPACE_MB
-----
          1536

-- Alter the Streams pool size:
SQL> ALTER SYSTEM SET STREAMS_POOL_SIZE=2048M SCOPE=BOTH;
```

If there is no free space in the SGA, shrink another area of memory first. Alternatively, the database can be restarted with a new `STREAMS_POOL_SIZE` initialization parameter value.

Once the Streams pool has been altered, continue to monitor the Replicat process performance to ensure memory starvation does not become an issue.

4. Target SGA parameters

The database parameters controlling the size of the shared memory components in the System Global Area (SGA) need to 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 the *Oracle GoldenGate Installing and Configuring Oracle GoldenGate for Oracle Database* at the following URL:

<http://docs.oracle.com/goldengate/1212/gg-winux/GIORA.pdf>

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 12.1.2.0 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. The only adjustment is to set the `STREAMS_POOL_SIZE` initialization parameter correctly, as explained earlier in this paper.

Data Pump Configuration

The primary function of the Data Pump is to read the trail files created by Extract and copy them to the target host. To increase performance of the Data Pump when table names and table structures are not altered or data is being filtered, use the `PASSTHRU` parameter in the Data Pump parameter file. 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 used with a wildcard to apply to multiple tables.

If there are tables that require mapping or data conversions, the `NOPASSTHRU` parameter should be used. Tables listed with the `NOPASSTHRU` parameter must be specified after the `PASSTHRU` parameter. Doing this increases Data Pump performance and reduces CPU usage.

For example:

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

REPORTCOUNT EVERY 15 MINUTES, RATE

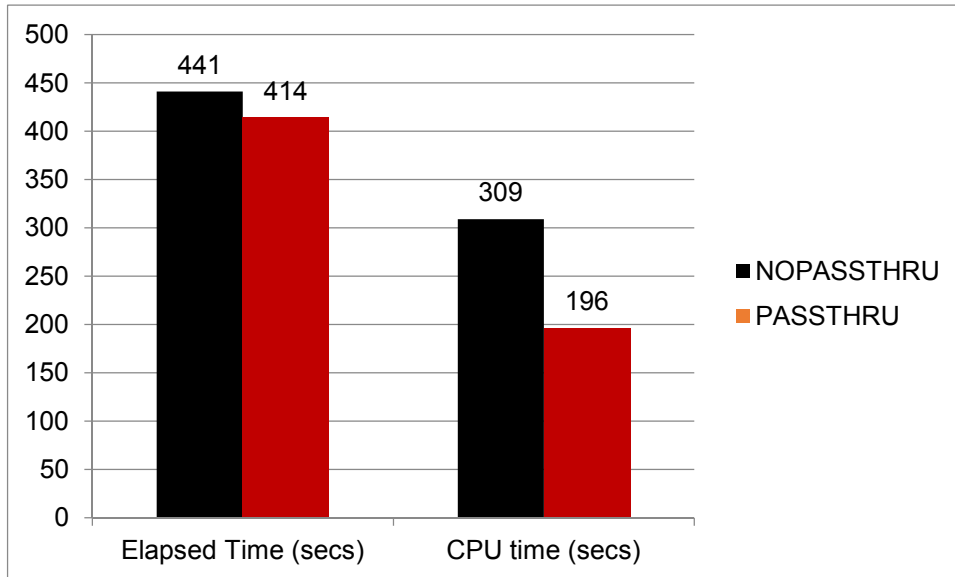
PASSTHRU
TABLE SOESMALL.*;

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

In this 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.

A performance comparison test was carried out 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:



The elapsed time difference is a 6% reduction when using the `PASSTHRU` parameter, but the bigger gain 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`

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.

These parameters should be set to a value of 1MB (1,048,576 bytes) or the calculated value, whichever is larger.

To determine suitable values for these parameters, execute 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 (Mbps) and the latency is 39ms, the calculation would be as follows:

$$\text{BDP} = (155,000,000 / 8) * 0.039 = 755,625\text{bytes}$$

c. Multiply the result by 3 to determine 3xBDP. For example:

$$3x\text{BDP} = 755,625 \times 3 = 2,266,875$$

In this example, because the result is more than 1MB, set the `TCPBUFSIZE` and `TCPFLUSHBYTES` parameters 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. Enable the asynchronous network streaming protocol

Starting with Oracle GoldenGate Release 11.2, an asynchronous network streaming protocol is enabled by default. This protocol maximizes the network utilization and efficiency, reduces the number of acknowledgments sent from the Collector process, and still provides network packet reliability. Enabling asynchronous network streaming protocol has advantages on higher latency networks.

Oracle recommends leaving this default setting enabled so that streaming is used.


If there is no setting in the Data Pump parameter file to disable streaming (`RMTHOST NOSTREAMING`), the asynchronous network streaming protocol is used.

Refer to the *Oracle GoldenGate Windows and UNIX Reference Guide* for more information on the `RMTHOST STREAMING` parameter located at the following URL:

<http://docs.oracle.com/goldengate/1212/gg-winux/GWURF.pdf>

3. 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. 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.

Refer to the *Oracle GoldenGate Windows and UNIX Reference Guide* for more information on `RMTHOST COMPRESSION` parameter for Data Pump located at the following URL:

<http://docs.oracle.com/goldengate/1212/gg-winux/GWURF.pdf>

Replicat Configuration

The Oracle GoldenGate Replicat processes running on the target database are reading the trail files and applying 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 Release 11.2.0.4. Integrated Replicat leverages the apply process functionality that is available inside the database. Referential integrity and DDL operations are automatically applied in the correct order. 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 apply processes based on the current workload and database performance. Management and tuning of Replicat performance is simplified since 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.

Refer to the *Installing and Configuring Oracle GoldenGate for Oracle Database* for additional information regarding integrated Replicat configuration located at the following URL:

<http://docs.oracle.com/goldengate/1212/gg-winux/GIORA.pdf>

2. Use a checkpoint table if using non-integrated Replicat

If non-integrated Replicat is used, create a checkpoint table in the target database.

The Replicat process maintains checkpoints that provide a known position in the trail file for recovery and restart. By default, this checkpoint information is stored in a checkpoint file for the Replicat process. Additionally, checkpoint information can also be stored in a checkpoint table in the target database so that the checkpoint information is included within the Replicat transaction itself. This provides a higher level of protection against inconsistencies between the checkpoint file and the applied transactions. Integrated Replicat automatically records checkpoint information within the target database so checkpoint table creation is not necessary.

When non-integrated Replicat uses a checkpoint table, it also takes advantage of the asynchronous commit feature that was introduced in Oracle Database 10g Release 2 (10.2). This feature enables Replicat to continue processing immediately after issuing the `COMMIT` command, which improves performance. If a checkpoint table is not used,

Replicat uses the `COMMIT` command with the `WAIT` option to prevent inconsistencies in case a database failure causes the state of the transaction in the checkpoint file to be different from the state of the transaction after the recovery.

To create the checkpoint table in the target database, use the following GoldenGate Software Command Interface (GGSCI) command:

```
GGSCI> ADD CHECKPOINTTABLE owner.table
```

Use a checkpoint table with all Replicat processes by modifying the `GLOBALS` file with the following parameter:

```
CHECKPOINTTABLE <checkpoint_table_owner>.<checkpoint_table>
```

For further details on using a checkpoint table, refer to section 9.3 of the *Installing and Configuring Oracle GoldenGate for Oracle Database* located at the following URL:

<http://docs.oracle.com/goldengate/1212/gg-winux/GIORA.pdf>

3. Set BATCHSQL in the Replicat parameter file

The default behavior of DML processing by Replicat differs slightly between integrated and non-integrated as explained in the following list items:

a. Integrated Replicat

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

To increase the Replicat apply performance further, `BATCHSQL` can be enabled 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. For example:

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	5q8xtx2bhquzi	GoldenGate	INSERT /*+ restrict_all_ref_c...
19,606,938	72,709,149	3.71	2,381.43	37.5	6.7	f4vq0uufrmsx0	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. For example:

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	5q8xtx2bhquzi	GoldenGate	INSERT /*+ restrict_all_ref_c...
4,835,457	72,743,061	15.04	1,453.91	70.4	7.3	f4vq0uufrmsx0	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 decreased elapsed time by approximately 31%.

The maximum size of each statement batch is controlled by the `BATCHSQL BATCHTRANSOPS` parameter. The default size of 50 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 causes 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.

b. Non-integrated Replicat

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	19rchijpb3462	OGG-REP_1A-OPEN_DATA_SOURCE	INSERT INTO "SOESMALL"."ORDER_...

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

In contrast, using 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	19rchijpb3462	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 1200. 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 few transactions are reverting back to normal mode (non-batched) because an exception was encountered. When many exceptions occur, apply performance can suffer due to 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 1200 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. When changing the `BATCHSQL` parameter, do so in a controlled manner so performance with the old and new settings can be accurately compared.

Further information on the `BATCHSQL` and `GROUPTRANSOPS` parameters can be found in the *Reference for Oracle GoldenGate Windows UNIX* at the following URL:

<http://docs.oracle.com/goldengate/1212/gg-winux/GWURF.pdf>

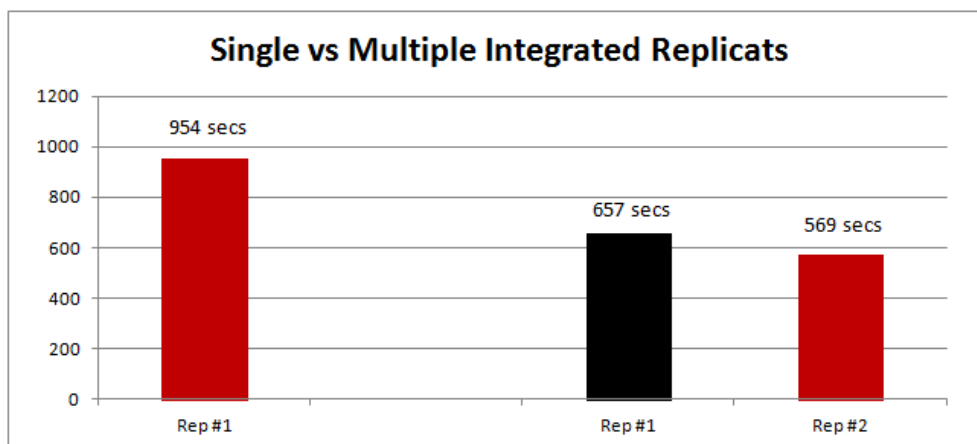
4. Multiple Replicat processes

Integrated Replicat uses dynamic allocation of apply server processes to distribute the work of applying transactions between the processes. New apply processes are created or removed dynamically to maintain the optimal Replicat throughput. This automation reduces the administrative overhead of manually dividing the work between Replicat processes.

Even when using integrated Replicat, there are times when configuring multiple integrated Replicat processes can offer a performance advantage. The following main reasons to do this are when:

- i. The integrated Replicat process is bottlenecked on the CPU. This most likely occurs when the process is carrying out many data transformations.
- ii. Large batch operations are being carried out 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 increase the apply performance.

The following is an example of a mixed workload of OLTP and large insert transactions. With integrated Replicat the apply server processes have to wait for the large transaction to be applied before the OLTP transactions can continue. When dividing the work between two integrated Replicat processes, the following graph shows a 31% decrease in elapsed time needed to apply the entire workload:



NOTE: When using multiple integrated Replicat processes with 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.

When a workload is generated by multiple processes, a single non-integrated Replicat process is seldom able to apply data at the same rate at which it was generated. The slower apply rate results in an increased latency between the source and destination databases. In some situations, a single Replicat process, along with a suitable `BATCHSQL` setting, can keep latency to an acceptable level. If after enabling `BATCHSQL` the Replicat latency is not

reduced to an acceptable time, multiple Replicat processes can be configured to apply data to different tables in parallel.

Oracle recommends to first configure a single Replicat process and monitor apply lag and performance. Adding more Replicat processes should only be carried out when performance of a single Replicat causes unacceptable apply latency.

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

a. 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:

- i. 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 also 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
```

- ii. 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

```

- iii. 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>

```

Or, use a wildcard:

```

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	

Refer to MOS Note 1301300.1 for detailed information about using the `logdump` utility to determine table DML rates.

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 example in Appendix B.

The number of Replicat processes to configure is determined by using an iterative process of adding Replicat processes until the latency is 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 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 Replicat processes using the `@RANGE` function.

For example, distributing a table between two Replicat processes would use the following MAP parameters:

```

Replicat #1:
  MAP SOESMALL.ORDER_ITEMS , TARGET soesmall.ORDER_ITEMS
  FILTER (@RANGE (1, 2, ORDER_ID));

Replicat #2:
  MAP SOESMALL.ORDER_ITEMS , TARGET soesmall.ORDER_ITEMS
  FILTER (@RANGE (2, 2, ORDER_ID));

```

The preceding example shows that the table `SOESMALL.ORDER_ITEMS` is partitioned using the primary key column `ORDER_ID` between two Replicat processes.

This is explained further in the *Administering Oracle GoldenGate for Windows and UNIX* at the following URL:

<http://docs.oracle.com/goldengate/1212/gg-winux/GWUAD.pdf>

b. 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.

c. 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 is applying 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:

- i. 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 fashion preventing DML from occurring on the same object at the same time. For further information on coordinated Replicat, refer to the *Administrating Oracle GoldenGate for Windows and UNIX* at the following URL:

<http://docs.oracle.com/goldengate/1212/gg-winux/GWUAD.pdf>

- ii. If 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 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 further details on replicating DDL statements, refer to the *Installing and Configuring Oracle GoldenGate for Oracle Database* at the following URL:

<http://docs.oracle.com/goldengate/1212/gg-winux/GIORA.pdf>

Database File System (DBFS) Configuration

When running Oracle GoldenGate on Oracle Exadata Database Machine, 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 the mounting of the DBFS file systems on a surviving node in the Oracle RAC cluster. This allows Oracle GoldenGate processes to automatically start once the required file systems have been mounted.

Refer to the *Oracle GoldenGate on Exadata Database Machine Configuration* white paper for details on how best to configure DBFS for optimal performance and availability located at the following URL:

<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. Refer to the *Transparent Zero Data-Loss Role Transition with Oracle Data Guard and Oracle GoldenGate* white paper for the configuration of such an environment located at the following URL:

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

Data Gathering for Oracle GoldenGate Performance

To monitor 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.

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 from the database using the following queries:

For integrated Extract:

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

For integrated Replicat:

```
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;
```

Lag is also reported by the Oracle GoldenGate manager process for both integrated and non-integrated Extract or Replicat. The following manager parameters must be specified 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 `ggsserr.log` 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 performance of Oracle GoldenGate, 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 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	bkpt6p42st6dg	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	6xxri1pzqz5s2	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. The generated report can be used 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 is using 148.3% 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). For example:

```
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.

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, execute the following steps:

- a. Grant the following privileges to a designated Oracle GoldenGate administrator database user:

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

- b. Connect to the database with the user name that was granted permissions in Step a.
- c. 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 to gather statistics during a 30-60 minute time period where performance is good, serving 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.');
```

Appendix A contains a shell script example that displays SPADV statistics in real time.

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 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 any performance bottlenecks. In this example, the LogMiner preparer (LMP) process is bottlenecked on CPU.

The following example shows the output for integrated Replicat (excerpt using an Oracle Database 12c 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 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. See the methodology described later to learn how to resolve such bottlenecks.

It is also possible to create a static report of SPADV statistics after monitoring for a period of time. The report can be generated in text form much like the display of real-time statistics.

To generate a text report, from SQL*Plus as the Oracle GoldenGate administrator, execute 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);
```

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.

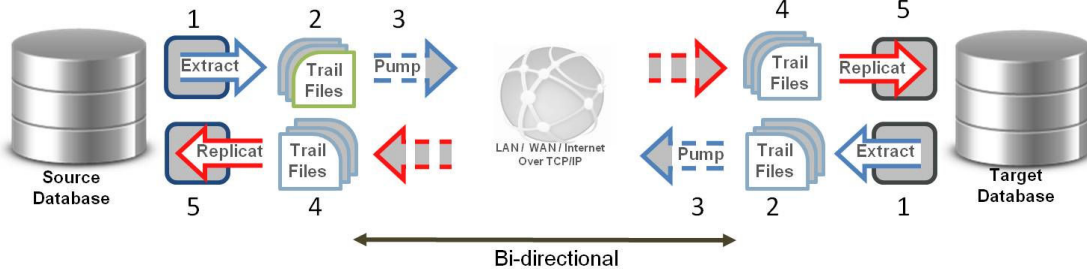
Healthcheck is a statically generated report, so it only reflects the status at one point in time. Oracle recommends gathering two or three healthcheck reports at several minute intervals to make sure that the components are flowing correctly.

For instructions on downloading and for further information about using the healthcheck, refer to MOS Note 1448324.1 located at the following URL:

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

Oracle GoldenGate Performance Tuning Methodology

Before trying 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. Where is the latency being reported first, moving from Extract to Replicat?

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

Once the left most process with lag has been found, 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_la.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. If latency is reported for an Extract process (not a Data Pump Extract)

a. If the Extract process (classic or Extract in 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.

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
```

b. If one or more of the LogMiner preparer processes are reaching maximum CPU (90-100%) 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 of `iostat` can quickly identify the issue. For example:

```
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    kB/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: 12:36:00 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    kB/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 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. If 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 is carrying out many transformations or conversions. You will also need to 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 shows up 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 shows up 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 previously detailed in the section titled “Data Pump Configuration”.

4. If latency is reported for a Replicat process (integrated and non-integrated)

a. If a Replicat process 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 (discussed earlier in the section titled “Replicat Configuration”). If you are doing a lot of data transformations, consider moving the transformations to the Data Pump process.

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 shows up in Data Pump performance. If multiple Replicat processes are configured to read the same trail files, consider using additional Data Pump processes so fewer Replicat processes are reading from the same files concurrently. For an example, refer to item 2d in the preceding section.

For Integrated Replicat:

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

c. Apply Reader (APR)

The apply reader process is responsible for accumulating the changes into transactions, computing dependencies between them, and then passing them to the apply coordinator. 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. This is covered earlier in the section titled “Multiple Replicat Processes.”

The following is an example of SPADV output:

```
PATH 2 RUN_ID 13 RUN_TIME 2014-JUL-24 10:07:57 CCA Y
|<R> REP_1A 54372 7906576 0 0% 33.3% 53.3% "CPU + Wait for CPU" |<Q>
"SOESMALL"."OGGQ$REP_1A" 54421 0.01 4654 |<A> OGG$REP_1A 48144 8770 -1 APR 0% 0%
100% "CPU + Wait for CPU" APC 93.3% 0% 6.7% "" APS (7) 46.7% 0% 640% "CPU + Wait
for CPU" |<B> OGG$REP_1A APR 848 10376 100.% "CPU + Wait for CPU"
```

d. Apply Coordinator (APC)

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 use multiple Replicat processes, as described in a previous section of this white paper.

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"
```

e. Apply Server (APS)

The apply server process is responsible for applying the DML to the database. When the apply server process is bottlenecked, it is treated in a similar way as the non-integrated Replicat process when it is also bottlenecked.

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 due to BATCHSQL not being 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	bu28rqrwk7y6rb	GoldenGate	UPDATE /*+ restrict_all_ref_c...
20,466,464	20,466,811	1.00	2,912.26	20.9	5.8	5q8xtx2bhquzi	GoldenGate	INSERT /*+ restrict_all_ref_c...
19,606,938	72,709,149	3.71	2,381.43	37.5	6.7	f4vq0iufrmsx0	GoldenGate	INSERT /*+ restrict_all_ref_c...
19,510,351	65,663,725	3.37	2,478.54	64.8	.1	7krmvq4xrdsuk7	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	bu28rqrwk7y6rb	GoldenGate	UPDATE /*+ restrict_all_ref_c...
4,855,639	20,476,399	4.22	1,670.95	39.3	7.8	5q8xtx2bhquzi	GoldenGate	INSERT /*+ restrict_all_ref_c...
4,835,457	72,743,061	15.04	1,453.91	70.4	7.3	f4vq0iufrmsx0	GoldenGate	INSERT /*+ restrict_all_ref_c...
4,832,499	65,837,321	13.62	1,991.06	89.1	.1	7krmvq4xrdsuk7	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, therefore, 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	I/O 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	7krmvq4xrdsuk7	GoldenGate	UPDATE /*+ restrict_all_ref_c...
2,612.04	1,652,174	0.00	18.88	4,662.34	19.12	56.02	5q8xtx2bhquzi	GoldenGate	INSERT /*+ restrict_all_ref_c...
601.62	1,652,152	0.00	4.35	2,273.72	26.88	26.46	f4vq0iufrmsx0	GoldenGate	INSERT /*+ restrict_all_ref_c...
233.50	1,652,159	0.00	1.69	2,189.53	49.35	10.66	bu28rqrwk7y6rb	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 the following URL:

<http://docs.oracle.com/database/121/TGDBA/E49058-04.pdf>

Finally, if database or object tuning does not help, configure additional integrated Replicat processes and divide the work of the slow performing Replicat process between multiple Replicat processes (as previously described in this white paper).

For Non-Integrated Replicat:

f. If the target database AWR report shows single row operations executed by the Replicat process, consider using `BATCHSQL` to take advantage of array processing in the database and reduce CPU usage.

The following is an example from an AWR report of non-integrated Replicat without `BATCHSQL` enabled, showing single row operations:

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	19rchijpb3462	OGG-REP_1A-OPEN_DATA_SOURCE	INSERT INTO "SOESMALL"."ORDER_...

In contrast, the following is an example using the `BATCHSQL OPSPERBATCH` parameter (number of operations the Replicat process tries to batch together) set to the default 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	19rchijpb3462	OGG-REP_1A-OPEN_DATA_SOURCE	INSERT INTO "SOESMALL"."ORDER_...

The rows per execution have increased to approximately 1000 and there is a 4.8 times reduction in elapsed time and, consequently, the CPU consumed is also reduced (not shown in the previous example).

g. If using `BATCHSQL` and a Replicat process is not applying data fast enough, evaluate database or object tuning techniques to improve performance. Refer to *Oracle Database Performance Tuning Guide* at the following URL:

<http://docs.oracle.com/database/121/TGDBA/E49058-04.pdf>

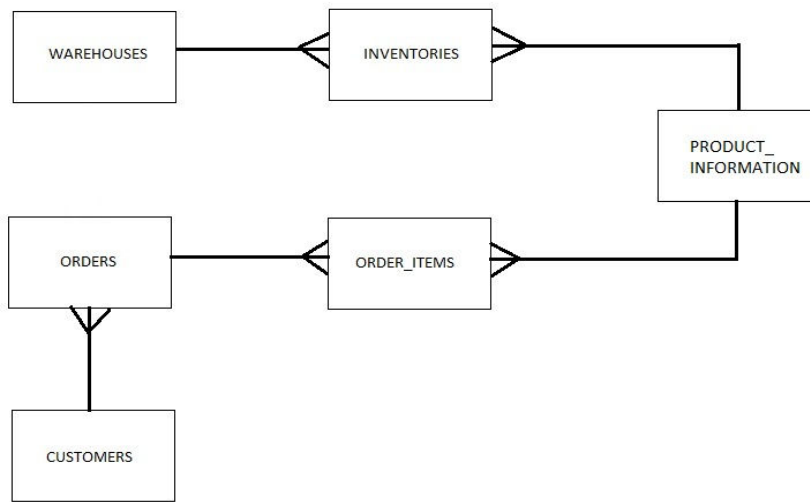
h. If database or object tuning does not help, configure additional Replicat processes and divide the work of the slow performing Replicat process (method described earlier in this white paper) between multiple Replicat processes.

Oracle GoldenGate Performance Swingbench Case Study

Workload Description

The Swingbench Order Entry workload is an Order Entry OLTP type benchmark, much like TPC-C. Swingbench was modified to remove many of the queries and sleep time to create a more DML-intensive workload against the database.

The workload applies DML against three tables: `ORDERS`, `ORDER_ITEMS` and `INVENTORIES`. The following figure shows the parent-child relationships that exist between tables in the schema:



The workload runtime duration is 90 minutes, with 38 concurrent user processes executing continuous Order Entry DML with no think or sleep time between transactions. This generates approximately 106GB of redo (21MB/second) with the following DML:

- Inserts and updates 20,480,644 rows into `ORDERS`
- Inserts 72,752,805 rows into `ORDER_ITEMS`
- Updates 65,846,925 rows in `INVENTORIES`

The workload was staged such that the Oracle GoldenGate processes were not started until the workload was completed to determine how fast the data could be replicated.

Environment Configuration

The source and target hardware are Oracle Exadata Database Machine X2-2 quarter racks using three storage cells.

The source and target database software are both release 12.1.0.2.

Both databases were configured with the following initialization parameters and values:

- `STREAMS_POOL_SIZE = 8G`
- `DB_CACHE_SIZE = 8G`
- `LOG_BUFFER = 128000000`
- `SHARED_POOL_SIZE = 1536M`
- `ARCHIVELOG` mode is enabled

For this exercise Oracle GoldenGate release 12.1.2.0.1 was used with the following configuration:

- A single integrated Extract with the following parameters:
 - `MAX_SGA_SIZE = 2048`
 - 500MB trail files stored on DBFS configured according to instructions in the *Oracle GoldenGate on Exadata Database Machine Configuration* white paper at the following URL:
<http://www.oracle.com/technetwork/database/features/availability/maa-wp-gg-oracledbm-128760.pdf>
- A single Data Pump using the `PASSTHRU` parameter to reduce CPU usage
- Multiple Replicat processes configured and tuned using the details in the section titled “Oracle GoldenGate Performance.”

Oracle GoldenGate Performance

Because this was a 90 minute staged workload, in order for Oracle GoldenGate to replicate the data at the same rate it was generated, the entire workload needs to be extracted and applied within 90 minutes.

Using a single integrated Extract process (configuration was discussed earlier in this white paper), the workload was extracted from the redo logs and the trail files written in 13 minutes and 34 seconds. Reading the 106GB of redo generated by the workload, this equates to a redo read rate of 134MB/second.

The Data Pump kept up with Extract and completed as soon as Extract had written the last entry in the source trail file.

On the target database, a single Replicat process with `BATCHSQL` disabled was initially configured. The single process took almost 8.5 hours to apply the workload. Using the performance tuning methodology previously outlined, this time was reduced to approximately 24 minutes. The following table shows the effects of adding additional Replicat processes with `BATCHSQL` enabled on the apply time:

Replicat Configuration	Time Taken (hrs:mm:ss)	Replicat Apply Rate (MB/second)*
One Replicat process (no <code>BATCHSQL</code>)	8:23:11	3.8
One Replicat process (with <code>BATCHSQL</code>)	1:57:12	16.2
Three Replicat processes (with <code>BATCHSQL</code>) One Replicat process per table (ORDERS, ORDER_ITEMS and INVENTORIES)	48:57	38.8
Six Replicat processes (with <code>BATCHSQL</code>) Two Replicat processes per table	35:00	54.3

(determined by the PK column)		
Eight Replicat processes (with BATCHSQL)	27:21	69.5
Four Replicat processes on the ORDERS table, two Replicat processes each for the ORDER_ITEMS and INVENTORIES tables		
Integrated Replicat (no BATCHSQL)	31:27	60.4
Integrated Replicat (with BATCHSQL)	24:19	78.1
Integrated Replicat (with BATCHSQL) All foreign keys are enabled during this test	26:04	72.9

* The Replicat apply rate is the rate at which the Replicat processes applied the workload-generated redo at the source database. For example, if the source workload generated 106GB, and the Replicat process applied the workload at the target database in approximately 24 minutes, the Replicat apply rate would be 78.1MB/second (106GB every 24 minutes).

The previous table demonstrates a number of things:

- In order to distribute the apply work between the non-integrated Replicat processes, any foreign key constraints must be disabled or dropped to allow the transactions to be distributed between the processes.
- When the foreign key constraints are enabled in the Swingbench schema, the integrated Replicat performs 4.5 times faster than a single non-integrated Replicat (with BATCHSQL enabled).
- Using an integrated Replicat process with BATCHSQL enabled, applying the workload is 4.8 times faster than a single non-integrated Replicat. The single integrated Replicat is 12% faster than eight manually configured non-integrated Replicat processes.
- When using multiple non-integrated Replicat processes, performance improves by a factor of 3.4 times when comparing a single Replicat process to eight Replicat processes.
- Performance improves by a factor of 3.4 times when enabling BATCHSQL for a non-integrated Replicat process.

These performance numbers are particular to the workload and the environment used for this testing. As with all performance comparisons, your extract and apply rates may differ.



Conclusion

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

In order to optimize an Oracle GoldenGate environment, the following crucial pieces of data need to 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 – 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
```

Appendix B – Dividing Tables Between Replicat Processes Using Perl Code

First, start with an Extract or integrated Replicat report file or with Oracle GoldenGate logdump output (detailed earlier in this white paper).

The following example shows report file table statistics:

```
From Table SOESMALL.ORDERS to SOESMALL.ORDERS:
#           inserts: 11517190
#           updates: 18554190
#           deletes: 0
#           discards: 0
From Table SOESMALL.ORDER_ITEMS to SOESMALL.ORDER_ITEMS:
#           inserts: 40915977
#           updates: 1278938
#           deletes: 0
#           discards: 0
From Table SOESMALL.INVENTORIES to SOESMALL.INVENTORIES:
#           inserts: 0
#           updates: 37035458
#           deletes: 0
#           discards: 0
```

Using these statistics, you can divide the number of rows between the Replicat processes being configured. The Perl code listed at the end of this Appendix takes an Extract or Replicat report file containing table statistics as an argument. The maximum number of configurable Replicat processes is based on the largest row count for a single table being the target number of rows to evenly allocate to each Replicat process.

The following Perl code does not consider further division of a large table between Replicat processes using the `FILTER` parameter. To do this takes some manual intervention to understand the consequences of possible DDL commands with foreign key and primary key relationships, as described earlier.

Step 1: Run the `balance_replicats.pl` script to determine the total row counts per table. For example:

```
% ./balance_replicats.pl ext_1a.rpt
Reading Extract report file: EXT_1A_HiDepSwing.rpt

Number of tables read in report file: 6
Total Rows: 121248275

**** ALL tables found in Extract report file (ordered by row count)****
SOESMALL.ORDER_ITEMS      45200510      37.3%
SOESMALL.INVENTORIES      39708064      32.7%
SOESMALL.ORDERS           32231079      26.6%
SOESMALL.LOGON            2055225       1.7%
SOESMALL.CUSTOMERS        2053397       1.7%
```

Enter number of Replicat processes required (1 to 3):

Step 2: When prompted, enter the number of Replicat processes that you want to configure:

```
Enter number of Replicat processes required (1 to 3): 3
```

```
** Target max Rows for each replicat = 45200510
```

```
** Replicat Configuration **
```

```
Replicat# 1 - Tables: 1 Rows: 45200510
```

```
MAP SOESMALL.ORDER_ITEMS, TARGET SOESMALL.ORDER_ITEMS
```

```
Replicat# 2 - Tables: 4 Rows: 43816687
```

```
MAP SOESMALL.CUSTOMERS, TARGET SOESMALL.CUSTOMERS  
MAP SOESMALL.INVENTORIES, TARGET SOESMALL.INVENTORIES  
MAP SOESMALL.LOGON, TARGET SOESMALL.LOGON  
MAP STREAMSTEST.GG_EVENT, TARGET STREAMSTEST.GG_EVENT
```

```
Replicat# 3 - Tables: 1 Rows: 32231079
```

```
MAP SOESMALL.ORDERS, TARGET SOESMALL.ORDERS
```

Step 3: Place the MAP statements provided into the Replicat parameter files to apply only the specified tables.

NOTE: Before dividing the tables between Replicat processes, you must consider any foreign key and primary key relationships. If present, either keep the related tables in the same Replicat process or disable the constraints.

Using a Swingbench OLTP workload example, Replicat maximum latency decreased from 31 minutes to 3 seconds using this approach.

balance_replicats.pl:

```
#!/usr/bin/perl  
#  
# balance_replicats.pl - Reads an Extract/Replicat report file or logdump  
#                       outfile to try and evenly divide up the tables  
#                       between multiple replicat processes  
#  
# The Extract or Replicat report file must contain table statistics, for example:  
#  
# From Table PSFT.PS_PAY_EARNINGS:  
#   #           inserts: 2352256  
#   #           updates: 4204032  
#   #           deletes: 250240  
#   #           discards: 0  
#  
# Alternatively, logdump can be used (see MOS note 1301300.1 for details) to show  
# count of table data, and must include data:  
#  
# SOESMALL.INVENTORIES                               Partition 4
```

```

# Total Data Bytes          46142040
#   Avg Bytes/Record        42
#   FieldComp               1098620
#   After Images            1098620
#
#
# Usage:
#
# Linux/Unix: ./balance_replicats.pl <report file or logdump outfile>
#
# Windows: perl balance_replicats.pl <report file or logdump outfile>
#
# NOTE;
#   - There is no consideration to FK/PK constraints between tables. It is
#     suggested you keep FK/PK tables within the same Replicat. If able to
disable
#     the constraints during time of replication, it can enable much higher
#     parallelism and faster apply rates.
#   - There is no consideration to performance of DML statements on the target
#     tables. Inserts, updates & delete performance to some tables can be highly
#     dependent on chosen index strategy.
#

# Table data
my %table_data;
my %Replicat; # Hash table to store table and replicat number

$logdump=0; # Set to 1 if logdump, else 0 for report file

# Right trim function to remove trailing whitespace
sub rtrim($)
{
    my $string = shift;
    $string =~ s/\s+$//;
    return $string;
}

# Perl version of grep to make it run on windows without real 'grep'
sub pgrep {

# First need to determine what type of file being read, Report or Logdump
# For each table Extract/Replicat report file includes:
#   From Table <owner>.<table_name>
# For logdump will include:
#   Current LogTrail is
#

    $rpt_string="From Table";
    $logdmp_string="Current LogTrail is";

    open(FH, $filename);

    while(my $line = <FH>) {
        if($line =~ m/$rpt_string/) {
            close FH;
            return(1);
        }
        elsif ($line =~ m/$logdmp_string/) {
            close FH;
            return(2);
        }
    }
}

```

```

}
# Not found either from report or logdump
return(0);
}

# readReportFile - Reads an Extract or Replicat report file and populates table_data
# hash
sub readLogDump {
    print "\nReading Logdump file: $filename\n";

    open(FILE, "<$filename") || # Open the segment header tracefile ready for
reading
        die "Cannot open file: $filename\n";

    $found = 0; # Found start of "Run Time Statistics"
    $done = 0;

    $table_count = 0;

    while ($done != 1) { # While not reached end of file
        $line = <FILE>; # Read in a line

        $where = index("$line", "*FileHeader*"); # Find start of entries

        if ($where != -1) {
            $done = 1;
        }

# Continue working through file
        while ($line = <FILE>) { # Not reached end of file
            $where = index("$line", "Partition 4"); # Denotes a table

            if ($where != -1) { # New table
                @words = split (' ', $line);

                $tablename=$words[0];

#Now read in the number of DMLS:
                $line = <FILE>;
                @words = split (' ', $line);
                $bytes = rtrim($words[3]);

                $total = $bytes;

# Add to hash table:
                $table_data{$tablename} = $total;

                if ( $total > $max_total ) { $max_total = $total; }

                $table_count++;
                $row_total = $row_total + $total;
            }
        }
    }

    print "\nNumber of tables read in file: $table_count\n";
    close FILE;
} # End of readReportFile

# readReportFile - Reads an Extract or Replicat report file and populates table_
# data hash
sub readReportFile {

```

```

print "\nReading Extract report file: $filename\n";

open(FILE, "<$filename") || # Open the segment header tracefile ready for
reading
    die "Cannot open file: $filename\n";

$found = 0; # Found start of "Run Time Statistics"
$done = 0;

$table_count = 0;

while ($done != 1) { # While not reached end of file
    $line = <FILE>; # Read in a line

    $where = index("$line", "*** Run Time Statistics ***"); #Find start of TOP
interval

    if ($where != -1) { # Start of section is found
        $done = 1; # Found it!
    }

# Continue working through file
    while ($line = <FILE>) { # Not reached end of file
        $where = index("$line", "From Table");

        if ($where != -1) { # New table
            @words = split (' ', $line);

            $tablename=$words[2];
            chop($tablename);# Remove the ":" from each tablename

#Now read in the number of DMLS:
            $line = <FILE>;
            @words = split (':', $line);
            $ins = rtrim($words[1]);

            $line = <FILE>;
            @words = split (':', $line);
            $supd = rtrim($words[1]);

            $line = <FILE>;
            @words = split (':', $line);
            $del = rtrim($words[1]);

            $total = int($ins + $supd + $del);

# Add to hash table:
# Need to make sure table not already in hash table to avoid duplicates

            if (exists $table_data{$tablename}) {
                print "*** WARNING: Table $tablename already found in report file. This
should not happen.\n";
            }
            else {
                $table_data{$tablename} = $total;

                if ( $total > $max_total ) { $max_total = $total; }

                $table_count++;
                $row_total = $row_total + $total;
            }
        }
    }
}

```

```

    }
}
print "\nNumber of tables read in report file: $table_count\n";
close FILE;
} # End of readReportFile

#####
# ** MAIN ** #
#####

# Check passed in arguments:
if ($#ARGV !=0) { # Check that 1 parameter been passed
in
    print "Usage: $0 <dir/filename>\n";
    print "\nExample: ./balance_replicats.pl /home/goldengate/dirrpt/ext_la.rpt\n\n";
exit 1;
}

($filename) = @ARGV;

$max_total = 0;
$row_total = 0;

#Determine which file we are looking at (Report or Logdump)
$result=pgrep;

if ( $result == 1 ) {
    # Read the Extract/Replicat report file:
    readReportFile();
    $logdump=0;
    $which='Rows';
}
elseif ( $result == 2 ) {
    readLogDump();
    $logdump=1;
    $which='Bytes';
}
else {
    print "\n*** ERROR: Cannot determine if file is Report file or Logdump file -
check file correctness\n\n";
    exit 1;
}

if ( $logdump ) {
    print "Total Bytes: $row_total\n";
    print "\n**** ALL tables found in Logdump file (ordered by bytes)***\n";
}
else {
    print "Total Rows: $row_total\n";
    print "\n**** ALL tables found in Extract report file (ordered by row
count)***\n";
}

foreach $table (reverse sort {$table_data{$a} <=> $table_data{$b} } keys
$table_data) {
    $perc = ($table_data{$table} / $row_total)*100 ;
    if ( $perc <= 0 ) {
        print "$table \t $table_data{$table} \t <1%\n";
    }
}

```

```

}
else {
    printf "$table \t $table_data{$table} \t %.1f%\n", $perc;
}
}

# Determine maximum number of Replicat processes and assign rows;
#   - Take the maximum #rows for a single table and set that to approximate min.
#     rows
#     that a single Replicat can apply
#   - Divide total# rows by single table max rows (previous step) to get max #
# Replicat processes (round up)
#   - Rounding up may leave one Replicat with lower #rows, which can be
#     manually altered
#   - Ask user to enter number of desired Replicat processes
#   - Divide total rows by #Replicat processes to find target row count per
#     Replicat
#   - Loop through Replicat processes, and list of tables until replicat up to
#     target row
#     count or out of tables
#   - Any leftover tables get assigned to the Replicat with the least number of
#     rows

$max_whole_replicats = int($row_total / $max_total);
$rem = $row_total % $max_total;
if ( $rem > 0.5 ) {
    $max_replicats = $max_whole_replicats + 1;
}
else {
    $max_replicats = $max_whole_replicats;
}

# Enter how many replicats desired (1 to $max_replicats)
# NOTE: Only need to do this if > 1 tables in report/log dump

if ( $max_replicats > 1 ) {
    $done=0;

    while ( $done != 1 ) {
        print "\n Enter number of Replicat processes required (1 to $max_replicats): ";
        $replicats = <STDIN>;
        chomp ($replicats);      # Remove carriage return
        if ( $replicats > 0 && $replicats <= $max_replicats ) {
            $done = 1;
        }
    }
}
else {
    $replicats = $max_whole_replicats;
}

# Now work out some kind of even distribution:
# 1. Determine approximate number of rows per replicat
# 2. Traverse sorted array and when find suitable table add into another table that
#     stores table_name, total and rep#
if ( $replicats > $max_whole_replicats ) {
    $max_rows = $max_total; }
else {
    $max_rows = int($row_total / $replicats);
}

```



```

print "\n** Target max $which for each replicat = $max_rows\n\n";
$table_count = 0;
$totals = 0;
$minrows = 0;

for ($i=1; $i <= $replicats; $i++) {
    $rep_rows = 0;

    # Array of tables for the current Replicat
    my @rep_tables = ();
    my $entry = $Replicat{$i};

    foreach $table (reverse sort {$table_data{$a} <=> $table_data{$b} } keys
%table_data)
    {
        if ( $rep_rows < $max_rows && ($rep_rows + $table_data{$table}) <= $max_rows ) {
            # Can add more tables
            $rep_rows = $rep_rows + $table_data{$table};
            $totals = $totals + $table_data{$table};
            $table_count++;

        # Add table to array
            push(@rep_tables, $table);

        # Delete element from master hash of all tables to make sure does not get assigned
        # again
            delete($table_data{$table});
        }
    }

    # Save array of tables for current Replicat
    $entry->{$tables} = [ @rep_tables ];
    $entry->{num_rows} = $rep_rows;
    $Replicat{$i} = $entry;

    # Determine Replicat with the lowest row count:
    if ( $rep_rows < $minrows || $minrows < 1 ) {
        $minrows = $rep_rows;
        $minrep = $i;
    }
} # End for all Replicat processes


$size = keys %table_data;
if ( $size > 0 ) {
    print "*** Tables NOT yet assigned are assigned to Replicat # $minrep: \n";
    while (($table, $total) = each(%table_data)){
        print "Name: $table \t $which: $total\n";
    }

    # Any tables left should be added to the replicat with the least number of rows!
    my $entry = $Replicat{$minrep};
    $entry->{num_rows} = $entry->{num_rows}+$total;
    push(@{$entry->{$tables}}, $table);
}
print "\n";
}

print "*** Replicat Configuration **\n";

foreach my $reps ( sort keys %Replicat ) {
    my $entry = $Replicat{$reps};
    my $rows = $entry->{num_rows};
}

```



```
my @tabs = @{$entry->{$tables}};

print "Replicat# $reps - $which: $rows\n";
# Print tables:
foreach ( sort @tabs ) {
    print $_ . "\n";
}
$numtabs = @tabs;
print "Number of tables: $numtabs\n\n";
}

exit;
```



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

November 2014

Author: Stephan Haisley
Contributors: Lawrence To

Hardware and Software, Engineered to Work Together

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. 1114

