A QLIK TECHNICAL WHITEPAPER

Snowflake Cloud Services Consumption Impact by Qlik Replicate

Best Practices for Cost Optimization

John Neal
Partner Engineering, Qlik

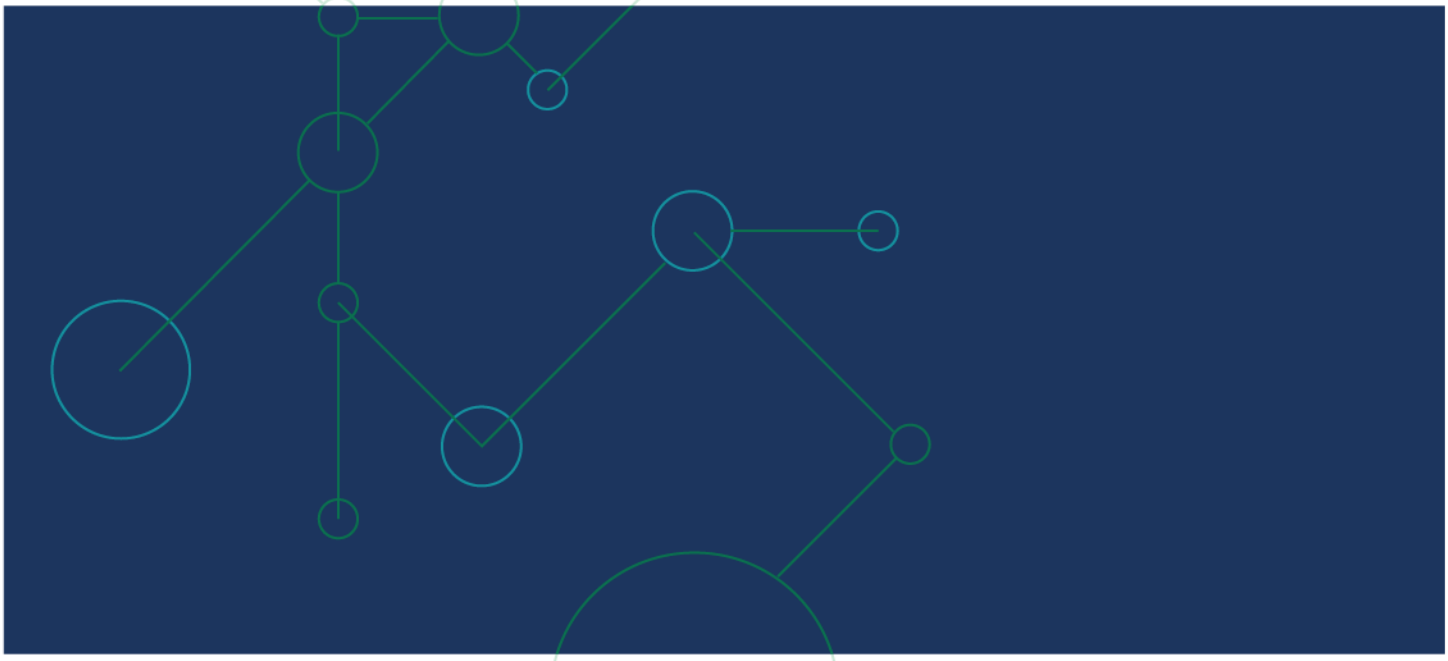

Table of Contents

Introduction

Qlik is a strategic partner for Snowflake. Qlik Replicate – and the Qlik Data Integration platform – is a market leader when it comes to continuous and real-time data ingestion from a wide range of enterprise data sources. We work closely with Snowflake to provide an aligned and optimized integration. Following the changes Snowflake introduced to its Cloud Services consumption billing in 2020, we realized this may have an impact on our joint customers. To get ahead of this and allow us to provide guidance to our customers, we embarked on an effort to gain detailed insight into this new billing model, how the use of Qlik Replicate could impact a customer's Snowflake bill, and what changes or guidance we need to provide to help customers optimize their cost/performance.

Snowflake has historically billed its customers based on consumption of data warehouse "compute" resources, along with a fee for storage. Not billed for were the computational resources consumed when parsing a SQL statement, generating an execution plan, etc. These non-warehouse computational resources that are expended are what Snowflake calls "Cloud Services".

In the Fall of 2019, Snowflake announced that it would be adjusting its billing model and begin charging customers for Cloud Services (also sometimes called "GS" or "Global Services"). This Snowflake blog post provides some background from the Snowflake side of things.

When you net things out, virtually every interaction you might have with Snowflake involves execution of a SQL statement of some sort, which in turn will result in consumption of Cloud Services resources. As Qlik Replicate can deliver very high volumes of data into Snowflake in real-time, it was important for us to understand what Qlik Replicate's delivery of data into Snowflake would mean to a customer's Snowflake bill.

Measuring Cloud Services Credit Consumption

In order to be able to make recommendations on how one might tune Qlik Replicate to optimize Cloud Services consumption, we first needed to figure out how to measure it. It turns out that Snowflake provides several database views that Snowflake customers can query in order to get an understanding of Cloud Services utilization. A couple of those views have proven to be extremely helpful.

Warehouse Metering History

The "warehouse metering history" view gives the user an hour-by-hour summary of billing credit consumption for a Snowflake warehouse. Credits consumed are broken down by both compute credits and cloud services credits.

The warehouse metering history view is not particularly wide, and all columns have proven useful.

```
#> desc table snowflake.account_usage.warehouse_metering_history;
name                          type
START_TIME                    TIMESTAMP_LTZ(9)
END_TIME                      TIMESTAMP_LTZ(9)
WAREHOUSE_ID                  NUMBER(38,0)
WAREHOUSE_NAME                VARCHAR(16777216)
CREDITS_USED                  NUMBER(38,9)
CREDITS_USED_COMPUTE          NUMBER(38,9)
CREDITS_USED_CLOUD_SERVICES   NUMBER(38,9)
```

```
#> select * from snowflake.account_usage.warehouse_metering_history
      where warehouse_name = '$REPLICATE_WH' and start_time >= '$START_TIME'
      and end_time <= '$END_TIME'
      order by start_time;
```

The query will yield results like this:

START_TIME	END_TIME	WAREHOUSE_ID	WAREHOUSE_NAME	CREDITS_ USED	CREDITS_ USED_COMPUTE	CREDITS_ USED_CLOUD_ SERVICES
2020-05-15 09:00:00.000 -0700	2020-05-15 10:00:00.000 -0700	31	SNOWTEST	1.2856994440	1.2444444440	0.0412550000
2020-05-15 10:00:00.000 -0700	2020-05-15 11:00:00.000 -0700	31	SNOWTEST	2.0684366670	2.0000000000	0.0684366670
2020-05-15 11:00:00.000 -0700	2020-05-15 12:00:00.000 -0700	31	SNOWTEST	2.0657513890	2.0000000000	0.0657513890
2020-05-15 12:00:00.000 -0700	2020-05-15 13:00:00.000 -0700	31	SNOWTEST	0.4408244440	0.4300000000	0.0108244440

As you can see, there are some useful aspects of this first query.

- First off, we can look at accumulated charges hour-by-hour, which will allow us to focus on periods of peak usage when we are doing our analysis.
- We can also look at the percentage of Cloud Services credits used vs. Compute Credits used. Snowflake – at the present time – provides a discount on cloud services used equal to 10% of the compute credits consumed. Even though it isn't reasonable to expect Cloud Services consumption to be less than that 10% (except in relatively low-volume scenarios), it is still a useful number to have for comparison purposes as we tune Qlik Replicate's behavior.
- Finally, we can find and total the Cloud Services credits used over a window of time, which is useful for validating the result of our analysis of the next query, described below.

Query History

The other view that has proven useful in the analysis of Cloud Services credit consumption is the "query history" view. This view essentially tracks every SQL statement that gets executed against Snowflake: the time the query began and ended, the database and warehouse, the user, the Cloud Services credits consumed by that query, and much more.

As you can see, there is a whole lot going on in this view. The fields we have used most extensively have been highlighted in yellow.

```
#> desc table snowflake.account_usage.query_history;
name                                    type
QUERY_ID                                VARCHAR(16777216)
QUERY_TEXT                              VARCHAR(16777216)
DATABASE_ID                             NUMBER(38,0)
DATABASE_NAME                           VARCHAR(16777216)
SCHEMA_ID                               NUMBER(38,0)
SCHEMA_NAME                             VARCHAR(16777216)
QUERY_TYPE                              VARCHAR(16777216)
SESSION_ID                              NUMBER(38,0)
```

```
USER_NAME                              VARCHAR(16777216)
ROLE_NAME                              VARCHAR(16777216)
WAREHOUSE_ID                           NUMBER(38,0)
WAREHOUSE_NAME                         VARCHAR(16777216)
WAREHOUSE_SIZE                         VARCHAR(16777216)
WAREHOUSE_TYPE                         VARCHAR(16777216)
CLUSTER_NUMBER                         NUMBER(38,0)
QUERY_TAG                              VARCHAR(16777216)
EXECUTION_STATUS                       VARCHAR(16777216)
ERROR_CODE                             VARCHAR(16777216)
ERROR_MESSAGE                          VARCHAR(16777216)
START_TIME                             TIMESTAMP_LTZ(6)
END_TIME                               TIMESTAMP_LTZ(6)
TOTAL_ELAPSED_TIME                     NUMBER(38,0)
BYTES_SCANNED                          NUMBER(38,0)
PERCENTAGE_SCANNED_FROM_CACHE          FLOAT
BYTES_WRITTEN                          NUMBER(38,0)
BYTES_WRITTEN_TO_RESULT                NUMBER(38,0)
BYTES_READ_FROM_RESULT                 NUMBER(38,0)
ROWS_PRODUCED                          NUMBER(38,0)
ROWS_INSERTED                          NUMBER(38,0)
ROWS_UPDATED                           NUMBER(38,0)
ROWS_DELETED                           NUMBER(38,0)
ROWS_UNLOADED                          NUMBER(38,0)
BYTES_DELETED                          NUMBER(38,0)
PARTITIONS_SCANNED                     NUMBER(38,0)
PARTITIONS_TOTAL                       NUMBER(38,0)
BYTES_SPILLED_TO_LOCAL_STORAGE         NUMBER(38,0)
BYTES_SPILLED_TO_REMOTE_STORAGE        NUMBER(38,0)
BYTES_SENT_OVER_THE_NETWORK            NUMBER(38,0)
COMPILATION_TIME                       NUMBER(38,0)
EXECUTION_TIME                         NUMBER(38,0)
QUEUED_PROVISIONING_TIME               NUMBER(38,0)
QUEUED_REPAIR_TIME                     NUMBER(38,0)
QUEUED_OVERLOAD_TIME                   NUMBER(38,0)
TRANSACTION_BLOCKED_TIME               NUMBER(38,0)
OUTBOUND_DATA_TRANSFER_CLOUD           VARCHAR(16777216)
OUTBOUND_DATA_TRANSFER_REGION          VARCHAR(16777216)
OUTBOUND_DATA_TRANSFER_BYTES           NUMBER(38,0)
INBOUND_DATA_TRANSFER_CLOUD            VARCHAR(16777216)
INBOUND_DATA_TRANSFER_REGION           VARCHAR(16777216)
INBOUND_DATA_TRANSFER_BYTES            NUMBER(38,0)
LIST_EXTERNAL_FILES_TIME               NUMBER(38,0)
CREDITS_USED_CLOUD_SERVICES            FLOAT
RELEASE_VERSION                        VARCHAR(16777216)
EXTERNAL_FUNCTION_TOTAL_INVOCATIONS    NUMBER(38,0)
EXTERNAL_FUNCTION_TOTAL_SENT_ROWS      NUMBER(38,0)
EXTERNAL_FUNCTION_TOTAL_RECEIVED_ROWS  NUMBER(38,0)
EXTERNAL_FUNCTION_TOTAL_SENT_BYTES     NUMBER(38,0)
EXTERNAL_FUNCTION_TOTAL_RECEIVED_BYTES NUMBER(38,0)

#> select start_time, end_time, query_type, schema_name, database_name, warehouse_name,
       user_name, credits_used_cloud_services
       from snowflake.account_usage.query_history
       where database_name = '$REPLICATE_DB' and start_time >= '$START_TIME' and end_time <= '$END_TIME'
       order by start_time;
```

There are some things that would be good to make note of here as they help explain the reason for including these columns:

- Use ***start_time*** and ***end_time*** to query against a window of time that aligns with the window you selected from the "warehouse metering history" query above. Having them both is also useful because there can be a lot going on in parallel with Qlik Replicate that we need to get our heads around:
    - Multiple Replicate tasks can be concurrently delivering data into the same database / warehouse
    - Qlik Replicate can utilize multiple threads to deliver data into Snowflake in parallel.
- ***Query_type*** is extremely useful as it allows us to group and sum information based on the kind of SQL statement that was being executed: ALTER, COMMIT, INSERT, UPDATE, DELETE, etc. More on this later.
- ***Schema_name*** is there mostly for a sanity check to validate that the statements we are looking at are working against the schema we are expecting.
- Having ***database_name***, ***warehouse_name***, and ***user_name*** allow us to ensure that we are only looking at data generated by the Qlik Replicate user. We can then filter out statements against the databases that aren't using the "Qlik Replicate warehouse" and / or were generated by users who aren't the "Qlik Replicate user".
- ***Credits_used_cloud_services*** is the amount of Cloud Services credits consumed by each SQL statement.
- ***Query_text*** gives us the actual query that was executed. This info is useful if we need to drill into one query or another for some reason. With a little parsing, we can also pull the table name out of INSERT, UPDATE, and DELETE statements to help identify hot tables, etc. for tuning purposes. Be aware, however, that including this column in the query can result in very large files being returned. For example, it can easily generate north of a gigabyte for a 24-hour period in systems with only moderate load. Unless you are handy at parsing large files with python, perl, or bash scripts it would probably be best to only include this column when doing more surgical analysis over a shorter window of time.

The result of this query, minus the ***query_text*** column would look something like this:

Start_time	End_time	Query_type	Schema_name	Database_name	Warehouse_name	User_name	Credits_used_cloud_services
2020-05-15 10:25:21.441 -0700	2020-05-15 10:25:21.908 -0700	INSERT	PUBLIC	SNOWTEST	SNOWTEST	JNEAL	0.00004700
2020-05-15 10:25:21.549 -0700	2020-05-15 10:25:21.873 -0700	DELETE	PUBLIC	SNOWTEST	SNOWTEST	JNEAL	0.00003400
2020-05-15 10:25:21.912 -0700	2020-05-15 10:25:22.164 -0700	COMMIT	PUBLIC	SNOWTEST	SNOWTEST	JNEAL	0.00003400
2020-05-15 10:25:21.957 -0700	2020-05-15 10:25:22.265 -0700	COMMIT	PUBLIC	SNOWTEST	SNOWTEST	JNEAL	0.00004200
2020-05-15 10:25:22.299 -0700	2020-05-15 10:25:23.398 -0700	UPDATE	PUBLIC	SNOWTEST	SNOWTEST	JNEAL	0.00009800
2020-05-15 10:25:23.446 -0700	2020-05-15 10:25:23.743 -0700	COMMIT	PUBLIC	SNOWTEST	SNOWTEST	JNEAL	0.00004100
2020-05-15 10:25:54.395 -0700	2020-05-15 10:25:54.783 -0700	TRUNCATE_TABLE	PUBLIC	SNOWTEST	SNOWTEST	JNEAL	0.00005400
2020-05-15 10:25:54.831 -0700	2020-05-15 10:25:55.108 -0700	REMOVE_FILES	PUBLIC	SNOWTEST	SNOWTEST	JNEAL	0.00004200
2020-05-15 10:25:55.228 -0700	2020-05-15 10:25:55.376 -0700	PUT_FILES	PUBLIC	SNOWTEST	SNOWTEST	JNEAL	0.00002300

2020-05-15 10:25:55.946 -0700	2020-05-15 10:25:56.849 -0700	COPY	PUBLIC	SNOWTEST	SNOWTEST	JNEAL	0.00010900
2020-05-15 10:25:56.888 -0700	2020-05-15 10:25:57.031 -0700	REMOVE_FILES	PUBLIC	SNOWTEST	SNOWTEST	JNEAL	0.00002200
2020-05-15 10:26:00.652 -0700	2020-05-15 10:26:01.867 -0700	UPDATE	PUBLIC	SNOWTEST	SNOWTEST	JNEAL	0.00007400
2020-05-15 10:26:00.709 -0700	2020-05-15 10:26:01.479 -0700	INSERT	PUBLIC	SNOWTEST	SNOWTEST	JNEAL	0.00008500

Data Collection

Historically, Qlik's testing and validation of any Qlik Replicate source or target endpoint has been focused on performance and scalability. With the changes to the billing model we now also had to think about things from a cost perspective. Given that Qlik Replicate is designed to run and deliver data continuously, we especially had to focus not just on the "micro" costs – the cost of individual interactions with Snowflake, but also the "macro" cost – how Cloud Services consumption might accumulate over time.

Once we understood how to measure credit consumption, we needed to generate, collect, and analyze a considerable amount of data.

Internal Research

We set out to run a series of benchmark tests to help us:

- Improve our understanding of Snowflake's Cloud Services in general
- Understand Cloud Services credit consumption by different types of SQL statements
- Measure objectively the impact of Qlik Replicate on Cloud Services consumption
- Identify Replicate tuning guidelines that would impact Cloud Services consumption
- Identify product optimization opportunities

Our test environment was isolated: no other users were accessing the Snowflake database and warehouse that were used in testing. Our tests involved running a consistent and repeatable source database load that contained a mixture of INSERT, UPDATE, and DELETE operations, using Qlik Replicate to deliver the change data into Snowflake. All tests were configured to use Snowflake "internal stage" when loading data from Qlik Replicate into Snowflake. Each test ran for a little more than two-and-one-half hours and generated more than 6 million CDC records, using different combinations of Qlik Replicate tuning parameters. The load we generated was not a large load by any means, but it was enough for us to draw some important conclusions.

We ran quite a few tests with different versions of Qlik Replicate and the Snowflake ODBC driver, and different Snowflake warehouse sizes. We won't cover all the tests we ran here, but instead will focus on two tests that will highlight what we learned.

Qlik Replicate Defaults

One test we ran as a baseline involved configuring a task using only the Replicate defaults. This was an important test for two reasons:

- If the data volume is not excessive, this configuration will deliver data into Snowflake in real-time with essentially no lag.
- Since this configuration is the most "real-time", we have found that many customers in fact take the defaults in order to keep their Snowflake data warehouse current.

You can review the results of this test in the following table. The columns are defined as follows:

- ***Query Type***: the general "type" of SQL statement that was executed
- ***Count***: the number of statements of this type that were executed over the period measured.
- ***CS Max, CS Min, CS Average***: the maximum, minimum, and average Cloud Services credit "cost" encountered for this type of statement.
- ***CS Sum***: the sum of the Cloud Services credit "cost" for all queries of this type.

Table 1: Cloud Services consumption using Replicate defaults

Query Type	Count	CS Max	CS Min	CS Average	CS Sum
ALTER_SESSION	2199	0.00002300	0.00000000	0.00000005	0.00010800
DROP	5	0.00004300	0.00001400	0.00002580	0.00012900
DESCRIBE_QUERY	14	0.00003700	0.00000500	0.00000936	0.00013100
CREATE	8	0.00002900	0.00001600	0.00002075	0.00016600
SET	14	0.00009300	0.00001200	0.00002071	0.00029000
USE	21	0.00008500	0.00000800	0.00002452	0.00051500
CREATE_CONSTRAINT	32	0.00004800	0.00001600	0.00002438	0.00078000
SHOW	129	0.00011200	0.00000000	0.00001196	0.00154300
CREATE_TABLE	33	0.00032100	0.00002400	0.00005045	0.00166500
PUT_FILES	1122	0.00113300	0.00000900	0.00002427	0.02722700
DELETE	332	0.00074400	0.00002600	0.00008592	0.02852500
INSERT	355	0.00105600	0.00003300	0.00009210	0.03269500
UPDATE	641	0.00111700	0.00003100	0.00011112	0.07122500
REMOVE_FILES	2244	0.00086400	0.00001300	0.00003842	0.08622300
TRUNCATE_TABLE	1091	0.00094700	0.00003900	0.00008349	0.09108600
COMMIT	1328	0.00137600	0.00003300	0.00007187	0.09544900
COPY	1125	0.00144500	0.00004800	0.00013103	0.14740400
Grand Total	**10693**	**N/A**	**N/A**	**N/A**	**0.58516100**

The most important takeaway from this test is that DML[1]-related operations comprised 77.04% of the total (8238 of 10,693) and were responsible for 99.09% of the cloud services credits that were consumed. (The PUT_FILE and REMOVE_FILE query types are an artifact of our use of Snowflake internal stage. Had we configured Replicate to use external staging, those queries would not have been present in the data.)

"Tuned" Qlik Replicate

In each of our other tests, we experimented with several Replicate tuning parameters to determine the impact those parameters had on the consumption of Cloud Services Credits.

The following table highlights the result of what we considered our most optimal test from the perspective of Cloud Services credit consumption. It was optimal in that we configured Replicate to use the largest batch size of our tests, allowing a bit more latency in the replication process vs. when using the Qlik Replicate defaults.

[1] DML stands for Data Manipulation Language – statements that modify or otherwise access data stored in the database.

Table 2: Cloud Services consumption when using larger batch sizes.

Query Type	Count	CS Max	CS Min	CS Average	CS Sum
DROP	4	0.00002900	0.00001800	0.00002350	0.00009400
DESCRIBE_QUERY	24	0.00001400	0.00000400	0.00000683	0.00016400
ALTER_SESSION	17	0.00002300	0.00000000	0.00001006	0.00017100
CREATE	8	0.00008300	0.00001900	0.00003450	0.00027600
SET	24	0.00004100	0.00001200	0.00001775	0.00042600
USE	36	0.00003100	0.00000800	0.00001306	0.00047000
CREATE_CONSTRAINT	31	0.00005800	0.00001700	0.00002323	0.00072000
CREATE_TABLE	31	0.00006300	0.00002600	0.00003348	0.00103800
SHOW	131	0.00002800	0.00000000	0.00000966	0.00126500
PUT_FILES	165	0.00010000	0.00001000	0.00001661	0.00274000
INSERT	55	0.00010000	0.00003100	0.00005433	0.00298800
DELETE	54	0.00013800	0.00003400	0.00005819	0.00314200
TRUNCATE_TABLE	135	0.00017200	0.00004700	0.00006971	0.00941100
REMOVE_FILES	330	0.00014800	0.00001300	0.00002885	0.00952000
COPY	166	0.00016300	0.00003800	0.00010084	0.01673900
COMMIT	700	0.00014400	0.00003100	0.00004816	0.03371100
UPDATE	591	0.00022900	0.00003300	0.00006560	0.03876700
Grand Total	**2502**	**N/A**	**N/A**	**N/A**	**0.12164200**

Once again, we see that DML made up the bulk of our total operations, 2196 of 2502 (87.77%) of the total operations and consumed 96.20% of the Cloud Services credits that were consumed.

It was quite clear across all our internal tests that the area of focus when tuning Qlik Replicate to reduce / minimize the consumption of Cloud Services credits should be on the DML we are generating rather than other (DDL[2]) operations.

[2] DDL stands for Data Definition Language – statements that perform administrative functions, define the structure of tables and indexes … essentially all statements that aren't DML.

Production Data

As a part of our study, we also worked with several customers to examine their production environments. We looked at the configuration of all their production Qlik Replicate tasks to understand how they had Qlik Replicate configured, and we looked at a combined 3 million SQL statements generated by Qlik Replicate in those environments to understand Cloud Services resource consumption. We will review the results of our analysis for two of those customers in the sections that follow.

Customer #1

This customer collected data for us over a 48-hour period. The table below reflects the first 24-hours. The data from the second 24-hour period was quite similar. As has proven to be commonplace, this customer had Qlik Replicate batch sizes configured to the defaults. The customer was running Qlik Replicate version 6.5 and an older version of the Snowflake ODBC driver.

The customer had 8 production Qlik Replicate tasks configured. Two tasks had "*Store Changes*" enabled, creating an additional Qlik Replicate change ("__ct" table) for each source table and creating additional load by creating an insert in the corresponding change table for each INSERT, UPDATE, or DELETE applied to the CDC target tables. Two tasks also had "*Store task recovery data in target database*" selected (a recovery option to help protect against Qlik Replicate internal data corruption issues). This causes task recovery information to be written to an *attrep_txn_state* table created in Snowflake. This table is updated with great frequency based on source log transaction boundaries which correspond to a point where Replicate might be repositioned and restarted. Including the change tables and the *attrep_txn_state* tables, Qlik Replicate is delivering data to around 375 tables in total.

If you look below, you'll see that over half of the SQL statements issued were "ALTER SESSION" statements. This is an artifact of a pre-6.6 version of Qlik Replicate coupled with a pre-2.21 version of the Snowflake ODBC driver. Note that the Cloud Services "cost" for these ALTER SESSION statements is negligible.

Table 3: 24-hours of Cloud Services consumption for customer #1 before tuning.

Query Type	Count	CS Min	CS Max	CS Average	CS Sum
ALTER_SESSION	525363	0.00000000	0.00000000	0.00000000	0.00000000
UNKNOWN	20	0.00001500	0.00005500	0.00002440	0.00048800
DESCRIBE_QUERY	193	0.00000400	0.00009300	0.00001336	0.00257900
CREATE	84	0.00002100	0.00013900	0.00004057	0.00340800
CREATE_CONSTRAINT	118	0.00001500	0.00051100	0.00004607	0.00543600
DROP	160	0.00001300	0.00048200	0.00003559	0.00569400
SET	192	0.00001300	0.00055500	0.00003703	0.00710900
USE	288	0.00000800	0.00031500	0.00002526	0.00727500
CREATE_TABLE	162	0.00002800	0.00068300	0.00005699	0.00923300
SELECT	263	0.00000700	0.00041900	0.00004680	0.01230900
SHOW	1080	0.00000000	0.00176700	0.00002095	0.02262900
DELETE	1353	0.00003400	0.00114400	0.00010911	0.14762400
TRUNCATE_TABLE	5523	0.00003900	0.00313500	0.00009712	0.53636800
COPY	5842	0.00007500	0.00170900	0.00016606	0.97010000
UPDATE	35258	0.00000400	0.00342000	0.00012547	4.42387900
INSERT	94959	0.00000000	0.00329600	0.00011265	10.69677800
COMMIT	131272	0.00003600	0.00481400	0.00010894	14.30136700
Grand Total	**802130**	**N/A**	**N/A**	**N/A**	**31.15227600**

First, note that this customer was configured to use Snowflake “external stage”, so you will notice that there are no “PUT FILE” and “REMOVE FILE” statements listed.

Confirming what we determined from our internal testing, DML operations accounted for the bulk of the Cloud Services credit consumption (99.76%). If you do the math, you’ll also see that the number of commits over the 24-hour window averages out to a little more than 1.5 commits per second.

As a result of our analysis, we recommended to the customer that they upgrade to Qlik Replicate 6.6 along with the latest Snowflake ODBC driver, and that they tune Qlik Replicate to make use of larger batch sizes.

The customer implemented our recommendations in a methodical series of tests: first increasing the batch size; followed by an upgrade of the ODBC driver; and finally upgrading to Replicate 6.6.

The table below shows the result after implementing all of the recommended changes.

Table 4: Customer#1 Cloud Services consumption after tuning (24-hours).

Query Type	Count	CS Min	CS Max	CS Average	CS Sum
UNKNOWN	1	0.000006	0.000006	0.000006	0.000006
DESCRIBE_QUERY	164	0.000004	0.000036	0.000009	0.001439
ALTER_SESSION	124	0.000000	0.000047	0.000012	0.001477
CREATE_CONSTRAINT	101	0.000015	0.000061	0.000026	0.002660
CREATE	80	0.000020	0.000255	0.000045	0.003621
USE	248	0.000000	0.000090	0.000018	0.004517
SET	164	0.000012	0.000156	0.000034	0.005580
DROP	146	0.000014	0.000184	0.000039	0.005639
CREATE_TABLE	146	0.000028	0.000257	0.000056	0.008124
SHOW	990	0.000000	0.000205	0.000014	0.013578
SELECT	305	0.000000	0.001025	0.000047	0.014280
DELETE	1330	0.000036	0.001870	0.000115	0.152639
TRUNCATE_TABLE	3060	0.000042	0.001068	0.000097	0.297330
COPY	3323	0.000078	0.001818	0.000169	0.562248
UPDATE	23232	0.000011	0.003199	0.000123	2.867486
INSERT	74414	0.000000	0.003074	0.000111	8.263174
COMMIT	98979	0.000029	0.002262	0.000099	9.755113
Grand Total	**206807**	**N/A**	**N/A**	**N/A**	**21.958911**

A quick analysis reveals that their tuning exercise – which introduced only a minute or two of lag into the replication process – reduced the total number of commits by 24.6% and resulted in a 29.5% reduction in total Cloud Services credit consumption. Note also the dramatic reduction in the number of ALTER SESSION statements that were executed, which was a direct result of **both** an upgrade of the ODBC driver from 2.20.x to 2.21.x and upgrading to Qlik Replicate version 6.6. Based on the intermediate results reported, we confirmed that upgrading only the ODBC driver or Qlik Replicate alone would have resulted in a smaller reduction.

From a Cloud Services consumption perspective, the biggest savings Without question, tuning to even larger batch sizes will reduce the Cloud Services even more.

Customer #2

Another customer we worked with provided us with 15 hours of query history data to analyze. As with Customer #1, this customer had Qlik Replicate batch sizes configured to the defaults and was running Qlik Replicate version 6.5 and an older version of the Snowflake ODBC driver.

The customer had 9 production Qlik Replicate tasks configured, replicating a total of 266 source tables.

If you look below, just as with Customer #1 you'll see that there were a significant number of "ALTER SESSION" statements. As mentioned, this is an artifact of a pre-6.6 version of Qlik Replicate coupled with a pre-2.21 version of the Snowflake ODBC driver. Note once again that the Cloud Services "cost" for these ALTER SESSION statements is negligible.

Table 5: Customer #2 Cloud Services consumption before tuning (15 hours)

Row Labels	Count	Min	Max	Average	Sum
ALTER_SESSION	405660	0.000000	0.000000	0.000000	0.000000
DESCRIBE_QUERY	44	0.000004	0.000014	0.000007	0.000321
PUT_FILES	20	0.000012	0.000068	0.000021	0.000420
CREATE	20	0.000019	0.000056	0.000028	0.000557
REMOVE_FILES	40	0.000013	0.000045	0.000018	0.000722
SET	44	0.000011	0.000031	0.000018	0.000774
USE	66	0.000008	0.000026	0.000014	0.000941
CREATE_CONSTRAINT	82	0.000018	0.000050	0.000025	0.002055
DROP	81	0.000014	0.000120	0.000026	0.002076
CREATE_TABLE	82	0.000025	0.000098	0.000038	0.003078
UNKNOWN	240	0.000010	0.000049	0.000017	0.003989
SHOW	668	0.000000	0.000118	0.000010	0.006672
SELECT	2026	0.000013	0.000342	0.000041	0.084074
DELETE	1512	0.000032	0.001677	0.000076	0.114944
MERGE	5876	0.000000	0.002298	0.000133	0.781724
TRUNCATE_TABLE	21789	0.000025	0.002072	0.000063	1.363953
COPY	21505	0.000068	0.003536	0.000113	2.427055
UPDATE	60053	0.000021	0.004161	0.000144	8.652612
COMMIT	202838	0.000007	0.000822	0.000066	13.425300
INSERT	144365	0.000000	0.004123	0.000097	14.033577
Grand Total	**867011**	**N/A**	**N/A**	**N/A**	**40.904844**

As with Customer #1, this customer was also configured to make use of Snowflake "external stage". The presence of PUT FILES and REMOVE FILES above indicate that Replicate was not the only process using this combination of Snowflake database, warehouse, and user. The same can be said for the presence of the MERGE statements, which at the present time Qlik Replicate does not utilize. This will have caused some small degree of skew in the data, but likely not enough to impact our findings.

Yet again, we have validation that DML is the primary cause of cloud services consumption with DML operations accounting for 99.74% of the total. You will also notice that the transaction load was heavier than Customer #1, averaging roughly 3.76 commits per second.

We recommended upgrading to Qlik Replicate 6.6 and the latest Snowflake ODBC driver, as well as tuning for larger batch sizes.

We have not yet received data collected after our recommendations were implemented. We fully expect the findings we have made with our testing and with Customer #1 to be further validated.

Tuning Qlik Replicate

There are several Qlik Replicate parameters that are helpful when working to tune Qlik Replicate to consume fewer Cloud Services credits by applying batches larger than the default. In more traditional circumstances, these parameters would be configured in this manner to support very high-volume scenarios by generating fewer, larger transactions against the target. It turns out that this same bit of tuning can help drive down the number of transactions generated against Snowflake even if the load isn't so high.

Snowflake Target Endpoint Configuration

The only change we need to make to the Snowflake target endpoint itself is to configure it to make use of larger files when uploading data from the Qlik Replicate server into the staging area, whether Snowflake "Internal" or "External" stage is being used.

The *Max file size (MB)* parameter can be found on the Advanced tab of the Snowflake endpoint configuration page. It specifies the maximum size of the CSV file used to transfer data to Snowflake. Setting this parameter to larger values can result in fewer file uploads, application of larger batches, etc.

The default value for the *Max file size (MB)* parameter is 100. The illustration below shows it having been set to 200, which will improve things. Increasing the value to 500 or even 1000 will have measurable impact. The maximum value for the field is 2000.

Figure 1: Snowflake Target Endpoint Configuration

External vs. Internal Stage

On the subject of "internal" and "external" stage, it is your decision which to use. Snowflake generally recommends using internal stage, and it is certainly the easiest to configure. There are valid reasons why one might use external stage as well. In the context if this Cloud Services discussion (as mentioned previously), use of internal stage will generate PUT FILE and REMOVE FILE statements which will impact Cloud Services consumption measurably. Using external stage will avoid these statements, but at the cost of a bit of added complexity and cloud storage fees.

Snowflake Task Settings

There are several parameters related to the tuning of batch sizes when delivering CDC data to Snowflake. These can be found within the Task Settings of your Qlik Replicate Snowflake task (a task is a combination of a source endpoint and a target endpoint, and where you select the tables you want to replicate, configure transformations, etc.).

To get to the appropriate page in your task settings, select "Change Processing", followed by "Change Processing Tuning."

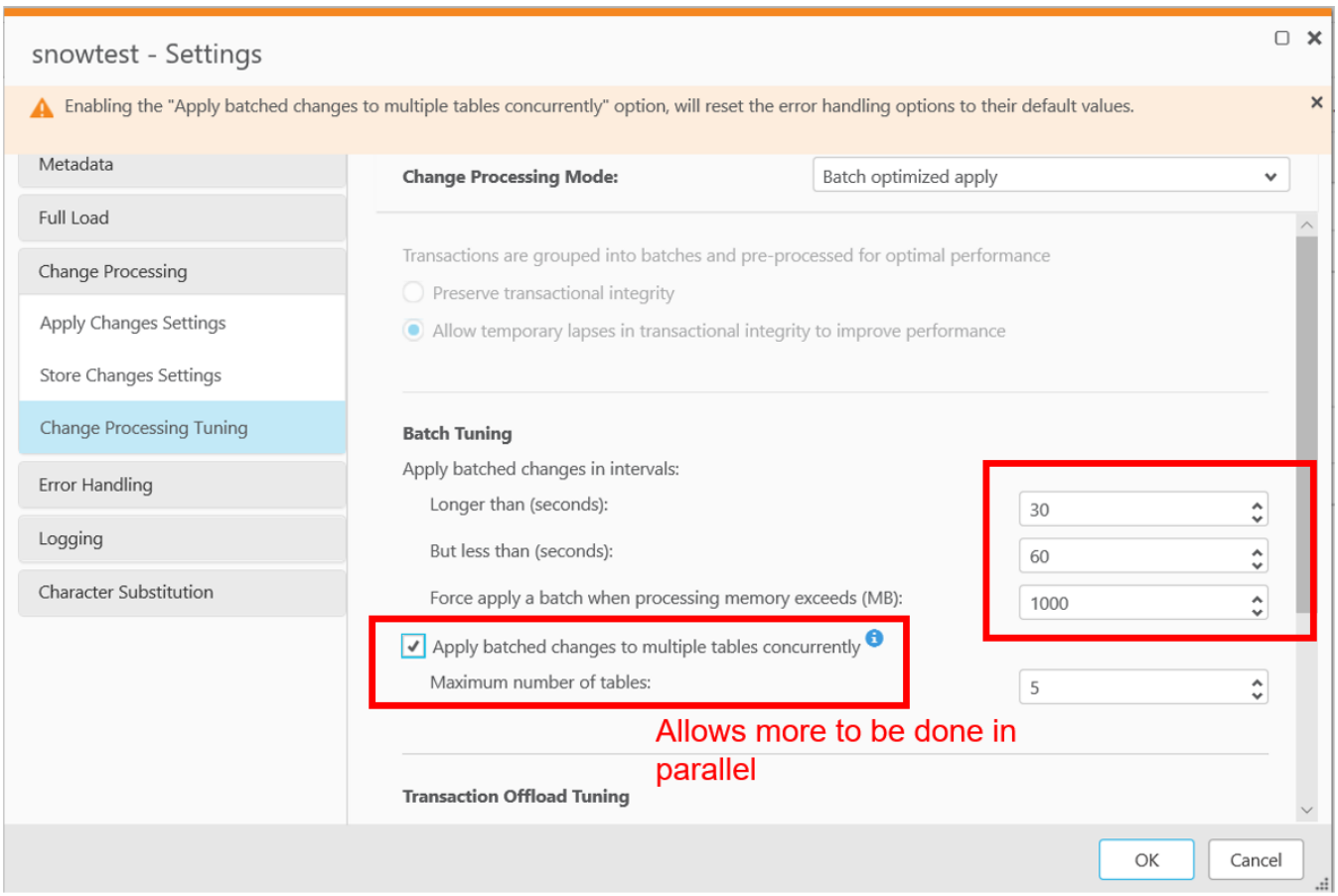

Figure 2: Change Processing Tuning settings

There are several important parameters on this page:

- *Change Processing Mode*: The default for Snowflake is *Batch Optimized Apply*. You should leave this setting alone. The other option is "Transactional Apply", which would be extremely inefficient – and expensive – when Snowflake is the target.
- *Apply batched changes to multiple tables concurrently*: this option essentially configures the number of threads that will work in parallel to upload and apply data to Snowflake. The default value is 5. Increasing this value won't reduce your Cloud Services consumption, but it can improve your throughput when there are many tables in flight at any point in time.
- *Longer than (seconds)*: This specifies the minimum amount of time to wait between each application of batch changes. The default value is 1 and has proven to be quite inefficient where Cloud Services consumption is concerned. Increasing this value decreases the frequency with which changes are applied to the target while increasing the size of the batches, essentially creating larger batches at the expense of some additional latency. We are recommending customers start with a value of 60 and increase even further if some additional latency is acceptable.
- *But less than (seconds)*: This value specifies the maximum amount of time to wait between each application of batch changes (before declaring a timeout). In other words, the maximum acceptable latency. The default value is 30. This value determines the maximum amount of time to wait before applying the changes, after the Longer than value has been reached. We are recommending customers

configure this value to 120 (combined with a *Longer Than* value of 60 and tuning the value even higher if more latency is acceptable.

- *Force apply a batch when processing memory exceeds (MB)*: this setting specifies the maximum amount of memory to use for pre-processing in Batch optimized apply mode. The default value is 500. For maximum batch size, set this value to the highest amount of memory you can allocate to Qlik Replicate. We are recommending customers start with a value of 2000 and consider tuning it even higher if possible.

Best Practice Recommendations Summary

We learned quite a bit in our testing and highlighted it in the preceding sections. The bullets that follow are our best practice recommendations.

- Isolate Qlik Replicate loading from other usage for clearer analysis
    - Create a separate user for Replicate
    - Use a separate warehouse for loading … loading data requires less compute than queries for analytics, etc.
- Always set *Apply batched changes to multiple tables concurrently*
    - Even if task only has a single table
    - This option causes Qlik Replicate to follow a slightly different code path that has proven to be more optimal.
- Tune Qlik Replicate to use larger batch sizes as described in the prior sections
    - Be prepared to compromise on batch size vs. latency
- Transactional apply should be avoided
- Option: Use external stage to avoid charges for PUT FILE and REMOVE FILE
- Upgrade to Replicate 6.6 or later
    - Qlik has made optimizations for Snowflake in this release
- Use latest Snowflake ODBC driver
    - Snowflake's is continually improving and making optimizations

Conclusions

We learned quite a bit during our testing. Our key takeaways were:

- All SQL statements result in Cloud Services resource consumption.
- DML rather than DDL statements are responsible for a significantly higher percentage of Cloud Services charges. The reasons for this are twofold:
    - Parsing statements and creation of an execution plan tends to be more involved for DML
    - Over time, there tend to be far more DML than DDL operations.
- As DML execution had the highest impact on Cloud Services consumption, we found that we could significantly impact that consumption by configuring Qlik Replicate to deliver larger batch sizes, accepting a small degree of increased latency in exchange for reduced cost.

Our findings from our internal benchmarks were backed up by what we learned from the customers we worked with. We found in all cases that DML operations resulted in the bulk of Cloud Services resource consumption. The customers consistently had Qlik Replicate configured to deliver the data in near real-time (as Qlik Replicate

can do), leaving considerable room to tune Qlik Replicate to deliver larger batches at a reduced Cloud Services cost. We learned that the closer that Qlik Replicate is configured to deliver data in real-time the higher Cloud Services consumption will be, with larger batch sizes generating lower consumption. The challenge for the customers was and will be finding the balancing point between acceptable latency and cost.

Customers should also be aware that configuring certain Qlik Replicate options, particularly “Store Changes” and “Store task recovery data in target database” will result in additional load. Use them where required but always keep in mind that in an environment where every DML operation matters there is no free lunch. Qlik Replicate is a powerful tool, but as they say, “With great power comes great responsibility.”

About Qlik

Qlik is on a mission to create a data-literate world, where everyone can use data to solve their most challenging problems. Only Qlik's end-to-end data management and analytics platform brings together all of an organization's data from any source, enabling people at any skill level to use their curiosity to uncover new insights. Companies use Qlik to see more deeply into customer behavior, reinvent business processes, discover new revenue streams, and balance risk and reward. Headquartered in King of Prussia, Pennsylvania, Qlik does business in more than 100 countries with over 48,000 customers around the world.

qlik.com