Analyzing Data Migration to Cloud Columnar OLAP Data Warehouses

FA 2023 CS 511 Project Report

Hari Umesh
humesh2@illinois.edu

Lilesh Kurella
lileshk2@illinois.edu

YuanTing Wang
yw101@illinois.edu

Jia Le Tian
jialelt2@illinois.edu

Abstract

OLAP or online analytical processing systems allow users to gather data from a multitude of sources and perform analytical tasks in a time and storage-efficient manner. Currently, among the biggest cloud vendors, Amazon Redshift and Google BigQuery are the most widely used systems for online analytical processing tasks. Despite their widespread use, there exists a lack of documentation and benchmarks on the migration from a traditional OLTP (online transaction processing) database to data processing systems. This paper explores the different ways that data from a traditional OLTP system like MySQL can be synced to BigQuery and Redshift. This paper also evaluates the performance of the migration, as well as the effectiveness of migration by utilizing various benchmarks to measure the efficiency of OLAP systems.

1 Introduction

Background With the rise of big data, the need for data processing and analytics is increasingly important. OLAP, online analytical processing, seeks to analyze data from many different sources and group the data to perform analytics to make informed business decisions [2]. One part of the OLAP architecture involves data warehouses, where data is collected from different sources and grouped in a way for fast future processing. Typically, data warehouses act as the center for data across a number of sources and are all aggregated together. Data can enter data warehouses through a number of sources, for example, OLTP (online transaction processing systems) systems which include other databases consisting of business data [1].

Currently, there are a number of cloud offerings of data warehouses available. Looking at some of the biggest cloud vendors, Amazon offers Amazon Redshift [15] and Google Cloud offers BigQuery [8]. Both systems are geared toward online analytical processing tasks. The two systems are built as columnar storage, which stores data in the form of columns instead of rows. Through this mechanism, the rate of accessing portions of data is improved, which is designed for OLAP operations. Furthermore, since more of the columnar data is repeated, columnar storage also allows for better data compression, which is suitable for working with large amounts of data [16].

However, OLAP and OLTP tasks require different systems that are more suitable for different use cases. For example, while BigQuery and Redshift excel at storing and processing large amounts of data, due to their columnar nature, they do not work well in OLTP contexts where small and random writes are frequently performed [16]. In the current database landscape, many possible storage options are available, each with its advantages and costs. This flexibility allows developers to choose and change their storage options based on business needs. In this case, data migration is performed to transport data from the OLTP storage for downstream analysis.

Problem Summary Columnar-based storage systems like Bigquery and Redshift excel at table scans but do not work well when accessing random points in memory storage. Currently, there are numerous articles and literature addressing the replication of data from other sources to these OLAP systems in a streaming manner [13] [12]. In this use case, data writes and updates in different sources are duplicated in a pseudo-real-time manner to the OLAP system. However, while there exists some delay, the data replicated to the OLAP system is generally small and in a pseudo-real-time manner.

The problem arises when a user wishes to migrate an entire OLTP database to a cloud-based OLAP system for processing. The resources for performing this operation are generally scattered and there are no conclusive studies evaluating the performance of such an operation. On top of this, many OLTP systems such as Amazon MySQL or Google MySQL, generally use a row-oriented database like MySQL. This shift in paradigm can cause both slow downs during the migration process and can also cause later problems during the formatting of the data in the column OLAP storage system. For any user who wishes to perform this task, it is easy to become frustrated with the process and also experience potential slowdowns due to performance during this process.

Proposed Approach Summary In our proposed approach, we aim to evaluate the process and performance of running migrations from a traditional row-oriented OLTP database to a cloud-based OLAP system such as Bigquery and Redshift. Our evaluation includes two types of datasets, a singular large table with a relatively simple schema and the standard TPC-H dataset. In our approach, we would first load the sample data into a cloud-based RDS. In this case, the project chooses MySQL as the baseline cloud RDS service. For Amazon Redshift, the data would be stored in Amazon MySQL [14], and for Google Bigquery the data would be stored in GCP MySQL [5]. Then for each of the respective platforms, the migration would come from the respective cloud-based MySQL. For

example, migration to Google BigQuery would come from Google MySQL. However, the project also contains a third cross-cloud platform migration where data is synced from Google MySQL to Amazon Redshift.

2 Problem Definition

In the realm of relational databases, our attention is drawn to a company currently relying on conventional RDBMS, specifically MySQL, to manage its burgeoning business data. However, as the volume of data grows exponentially, the limitation of the existing solution to storing the data becomes evident. The extended wait time for both data retrieval and update not only hinders immediate operational efficiency but further limits the organization's ability on potential opportunities to extract meaningful insights through advanced analytical workloads.

To address these constraints, the company aims to target a new database storage system that is able to achieve the following objectives:

(1) Improved Performance: The company's current row-based architecture is struggling to keep up with the increasing volume and complexity of data. This causes delays in data retrieval and update operations, which also impacts operational efficiency and hinders decision-making processes.

(2) Enhanced Analytical Capabilities: The current database structure is not optimized for complex analytical workloads. This is preventing the company from fully leveraging its data for business insights and decision support.

(3) Seamless Migration Process: The migration process must be carefully planned and executed to minimize downtime and ensure a seamless transition. Cloud-native technologies and services could provide a scalable, performant, and analytical-friendly database solution for the company.

To cope with the need mentioned above, the company decides to migrate its database architecture from the conventional row-based model to a more adaptive columnar-based storage system. Given the advantages of a columnar database in this context, the company aims to assess various database implementations to determine the most efficient application for its data. To ensure a seamless transition with minimal engineering overhead, the migration is further specified to occur in a cloud-based environment. The company is also interested in the possible overhead of the migration process, as this information is crucial for establishing realistic goals and initiating the migration with a well-informed strategy. Following an examination of both database performance and migration overhead, the company will make an informed decision on the columnar-based database system to adopt. Upon completion of the migration, the company anticipates mitigating its current performance bottlenecks and unlocking the full potential of its data for future sophisticated analytical tasks.

3 Methodology

To thoroughly evaluate the migration process and performance gains, we have devised a comprehensive methodology that involves benchmarking the migration process and query processing performance for both Google BigQuery and Amazon Redshift, the two leading cloud service providers offering full cloud services related to database storage systems. This approach aims to address the company's current database limitations and ensure a smooth transition to a more efficient cloud-based columnar-based storage system. For comparison, we also benchmark the original MySQL database using the same queries.

3.1 Dataset Selection

(1) **Transaction Processing Performance Council Benchmark H (TPC-H)** [17] is a widely recognized standard for measuring the performance of different database systems, and support systems. For this experiment, TPC-H helps standardize our performance measures, schemes, and queries. It also sets a standard for industry-level relational data.

(2) **Food_1 table from Public BI Benchmark** [9] is a user-generated benchmark taken from one of the 46 main Tableau Public datasets. This dataset gives this experiment the opportunity to deal with a bigger set of data which can be later used to properly understand the impact of big data migration from platforms.

3.2 Migration Process

The migration process involves moving data from the existing MySQL row-oriented database to both Google BigQuery and Amazon Redshift. The steps include:

(1) *Schema Mapping and Transformation*: Analyzing and mapping the existing MySQL schema to a format suitable for columnar storage. Transformation steps are applied to ensure compatibility with the target databases.

(2) *Data Extraction and Loading*: Extracting data from MySQL and loading it into Google BigQuery and Amazon Redshift. This process is monitored for efficiency, speed, and any potential challenges encountered during extraction and loading.

(3) *Data Consistency and Integrity Checks*: Verifying the consistency and integrity of the migrated data to ensure that it aligns with the source MySQL database.

Since both Amazon Web Services and Google Cloud Service offer their respective cloud-native MySQL servers, we loaded the datasets into Google Cloud SQL Instance for migration to BigQuery and Amazon MySQL for migration to Amazon Redshift. To replicate real-world migration scenarios that typically originate from a local MySQL server, we additionally conducted a migration from Google Cloud SQL Instance to Amazon Redshift, incorporating network latency into the migration process.

3.3 Experiment Setup

To ensure a comprehensive evaluation of the migration process and performance improvements, we conducted experiments for

each migration path, focusing on Google BigQuery and Amazon Redshift.

Google SQL to Google BigQuery For the Google SQL to Google BigQuery migration, we import data into an instantiated MySQL instance, then utilize a create query that copies data into BigQuery.

(1) Initialize database and tables inside Google MySQL. Create and import a SQL file that creates required databases and tables. Then import data into tables in CSV format [6].

(2) Connection Setup and Permissions: Initialize instances for Google MySQL and Google BigQuery under the same project.
   (a) Allow permissions to allow BigQuery access, or make instance public.
   (b) Per table needed to migrate, create a connection object in Google BigQuery

(3) Migration Querying: Initialize Google BigQuery queries that access all of Google MySql data through the EXTERNAL_QUERY function [7].

Amazon MySQL to Amazon Redshift For migrating from Amazon MySQL to Amazon Redshift, we employ Amazon MySQL and Amazon S3 as intermediate storage, the experimental setup for this migration path unfolds as follows:

(1) Connection Setup: Secure connection to Amazon MySQL and to Amazon Redshift is established using via MySQLServer and serverless redshift instance, with necessary environment variables configured for password management.

(2) Schema Extraction and Definition: Each table schema is extracted and utilized as both a MySQL and PostgreSQL table schema to ensure consistent data structure alignment during the migration from MySQL.

(3) Data Migration Loop: The migration process systematically iterates through tables with key steps:
   (a) Initialize MySQL Schema in MySQL Server and upload CSV data to RDS via database connection with MySQLServer
   (b) Create Data Connections to RDS and Redshift in AWS Glue
   (c) Create data Crawlers that connects to the data stores, determines the schema for the data, and creates metadata tables in your data catalog as new database tables
   (d) Verify and Create IAM Permissions for access to RDS, Redshift, and S3
   (e) Create ETL Job specifying the source: RDS, target: Redshift, and the table to be migrated
   (f) Run the migration and report the results

Google SQL to Amazon Redshift For migrating from Google SQL to Amazon Redshift, we employ Google Cloud Storage and Amazon S3 as intermediate storage, the experimental setup for this migration path unfolds as follows:

(1) Connection Setup: Secure connection to Google SQL and to Amazon Redshift is established using gcloud and psql respectively, with necessary environment variables configured for password management.

(2) Schema Extraction and Definition: Each table schema is extracted from Google SQL, modified as a PostgreSQL table schema to ensure consistent data structure alignment during the migration from MySQL.

(3) Data Migration Loop: The migration process systematically iterates through tables with key steps:
   (a) Export from MySQL to Google Cloud Storage: Utilizes Google Cloud SQL's export feature to export CSV data from the MySQL instance to Google Cloud Storage.
   (b) Upload CSV to Amazon S3: Synchronizes and uploads exported CSV files from Google Cloud Storage to Amazon S3 using gsutil.
   (c) Redshift Table Creation and Data Import: Utilizes psql to create Redshift tables based on the predefined schema and imports data into Redshift from S3.

This experiment setup allows us to assess the efficiency and performance of each migration path under realistic conditions, providing valuable insights for the company's decision-making process. Each migration path was executed separately to evaluate the specific challenges and advantages associated with the chosen cloud services and data warehousing solutions.

4 Evaluation

4.1 Migration Evaluation

Table 1: Migration Time Across Different Strategies

Time		Record #	Google MySQL to BigQuery	Amazon MySQL to Redshift	Google MySQL to Redshift
Dataset	Table				
FOOD	Food_1	5216593	22076 ms	158000 ms	97794 ms
tpch1g	lineitem	1199969	33883 ms	118000 ms	77338 ms
tpch1g	nation	25	792 ms	95000 ms	22306 ms
tpch1g	orders	300000	6804 ms	91000 ms	30225 ms
tpch1g	part	40000	1786 ms	138000 ms	24805 ms
tpch1g	partsupp	40000	2010 ms	81000 ms	25438 ms
tpch1g	region	5	878 ms	73000 ms	22372 ms
tpch1g	supplier	2000	976 ms	76000 ms	23093 ms
tpch1g	customer	3000	1906 ms	83000 ms	25198 ms

To perform the migration evaluation, we performed each of the three migrations according to the experiment setup. For Google MySQL to Google BigQuery, the migration is measured using the query processing information [11]. For the Amazon MySQL to Amazon Redshift, the migration is measured using the AWS Glue monitoring job dashboard [4]. For the Google MySQL to Amazon Redshift, a bash script is used as a wrapper script to the job to measure the total execution time for the migration job.

Looking at the results found in Table 1, it is clear that Google BigQuery is significantly faster at syncing the data sets than both of the other approaches with the target being Amazon Redshift. In particular, the migration time from Amazon MySQL to Redshift is particularly interesting as it is even slower than migrating the

data from Google MySQL as it is cross-platform. The main reason for the slow rate of native Amazon migration is due to the Amazon migration using Spark as part of the default option for AWS Glue [3]. While using a distributed computing framework like Spark is effective for syncing large amounts of data, for small amounts of data, the significant overhead for starting up a spark job does not compensate for the distributed computing performance boost. The observed latency underscores that our database may not be sufficiently large to fully showcase the performance advantages of Amazon Redshift.

Table 2: Migration Size Across Different Strategies

space		record #	Google MySQL to BigQuery	Amazon MySQL to Redshift	Google MySQL to Redshift
dataset	table				
FOOD	Food_1	5216593	643.5 MB	225.1 MB	224.1 MB
tpch1g	lineitem	1199969	409.8 MB	187.7 MB	165.9 MB
tpch1g	nation	25	5.09 KB	2.2 KB	2.2 KB
tpch1g	orders	300000	77.64 MB	33.4 MB	34.9 MB
tpch1g	part	40000	12.08 MB	5.0 MB	5.1 MB
tpch1g	partsupp	40000	11.33 MB	5.0 MB	4.9 MB
tpch1g	region	5	922 B	409 B	411 B
tpch1g	supplier	2000	573.05 KB	231.8 KB	239.8 KB
tpch1g	customer	3000	9.48 MB	4 MB	4 MB

We also record the size used during the migration process. From Table 2, we can see that the size associated with the "Google SQL to Google BigQuery" path is considerably larger compared to those of migrations directed to Amazon Redshift. It might be one of the reasons that its migration time outperforms the other paths. When contrasting the sizes utilized in migrations from Amazon MySQL to Amazon Redshift and from Google SQL to Amazon Redshift, a notable similarity emerges. This similarity suggests the possibility of shared underlying techniques, such as AWS Glue employing consistent methods for migration across these paths.

4.2 Benchmarks

The datasets chosen play a pivotal role in testing the performance of our different data storage systems. To best test the capabilities of the different data storage systems (Google SQL, Google BigQuery, Amazon Redshift, etc.) this experiment utilizes predetermined benchmarking queries that use relational information on data from TPC-H in order to understand database run time, aggregation advantages, coalescing differences, and much more.

Table 3: Benchmarks Across Different Systems

	Google MySQL	Google BQ	Amazon Redshift
Benchmark 1	3555 ms	660ms	5095 ms
Benchmark 2	3190 ms	905ms	8448 ms
BenchMark 3	6208 ms	1356ms	10464 ms

Table 3 holds time elapsed while running three different benchmark queries to best understand the speed and usability of these different databases.

4.2.1 Benchmark 1 **Pricing Summary Report Query** is a query that provides a price report for all items in the line-item table between certain dates. The query design incorporates summations, date-where clause filtering, as well as group and order by statements. This is a more basic benchmark but it tests basic SQL database filtering. Table 3 shows Google SQL spending 3555 ms in order to complete and return the correct output. For this experimentation, we considered Google SQL as the baseline for our querying capabilities. Google BigQuery and Amazon Redshift had much different results in comparison to the baseline. BigQuery's elapsed time is monumentally quicker than GCP MySQL and Redshift's time while running this benchmark is much larger. We can conclude regular evaluation is more efficiently used in BigQuery.

4.2.2 Benchmark 2 **Volume Shipping Query** is the benchmark 2 query. The purpose of this benchmark is to give 2 different nations, the gross discounted revenues from the line items table where the parts were shipped from the supplier between particular dates. This query utilizes embedded queries, number extraction, embedded filtering statements, etc. The pattern continues in Table 3 where the baseline Google SQL time elapsed is in between Redshift's poor time complexity and Google BigQuery's efficient query evaluation. This benchmark again proves filtering efficiency with columnar data, particularly in Google BigQuery.

4.2.3 Benchmark 3 **Business Question** is a much more convoluted functional TPC-H query that evaluates market share for given nations within a region for products of a specified region. This query deals with the aggregation of embedded queries, CASE evaluation, and a large amount of joined tables. This benchmark was by far the most taxing query on time with growth in elapsed time shown in every different database system. Table 3 shows that benchmark 3 corroborates the findings discussed in the previous two benchmarks. Although there is growth in each database's computational speed, they are still ordered the same in speed.

When evaluating the speed of each benchmark on the three different database systems, we clearly see the advantage of Google BigQuery in utilizing joins, aggregating large chunks of data, and referencing embedded queries. A large portion of this can be attributed to how BigQuery predetermines space complexity before any execution of querying [10]. Another important feature to understand about Amazon Redshift's overhead when dealing with small tables and queries. Some tables utilized in benchmarking are very small, as shown in Table 1. Given incredibly large datasets, there tends to be more normalization of elapsed time. This is vital when understanding the use cases of different database systems. We can conclude Google BigQuery is the best source for runtime cost when given a regular to small-sized dataset with complicated querying assignments.

5 Discussion & Limitations

Based on the analysis and experiment setup, we have identified the following assumptions and possible limitations.

(1) **Cloud Hardware**: For different cloud vendors and systems, the underlying hardware may be different which can impact the performance. To address this, we choose similar pricing options for all cloud platforms. However, this assumption for similar hardware may not fully hold depending on the underlying implementation.

(2) **Dataset Size**: To limit the cost of running the experiments, we used data sets that are large enough to identify differences between OLAP platforms and migration approaches but not too expensive in terms of cloud storage costs.

(3) **Network Latency**: During our experiments, network latency can be a factor that slows down sync between different cloud services. For our experiment, we assume that the cross-platform migration would incur some level of network latency but the same platform sync would incur much less latency. This assumption is a possible limitation since the assumption could impact our analysis of our benchmark and migration data.

References

[1] Amazon. [n.d.]. What is a Data Warehouse. https://aws.amazon.com/what-is/data-warehouse/

[2] Amazon. [n.d.]. What is OLAP. https://aws.amazon.com/what-is/olap/#:~:text=Online%20analytical%20processing%20(OLAP)%20is,smart%20meters%2C%20and%20internal%20systems.

[3] Amazon. 2023. *AWS Glue: How it works.* Retrieved November 26th, 2023 from https://docs.aws.amazon.com/glue/latest/dg/how-it-works.html

[4] Amazon. 2023. *Monitoring AWS Glue.* Retrieved November 26th, 2023 from https://docs.aws.amazon.com/glue/latest/dg/monitor-glue.html

[5] Google Cloud Compute. 2023. Cloud SQL for MySQL documentation. Retrieved November 26th, 2023 from https://cloud.google.com/sql/docs/mysql

[6] Google Cloud Compute. 2023. Export and import using CSV files. Retrieved November 26th, 2023 from https://cloud.google.com/sql/docs/mysql/import-export/import-export-csv#:~:text=v1%20REST%20v1beta4-,In%20the%20Google%20Cloud%20console%2C%20go,the%20Cloud%20SQL%20Instances%20page.&text=Cloud%20SQL%20Instances-,To%20open%20the%20Overview%20page%20of%20an%20instance%2C%20click%20the,to%20use%20for%20the%20import.

[7] Google Cloud Compute. 2023. Introduction to external tables. Retrieved November 26th, 2023 from https://cloud.google.com/bigquery/docs/external-tables

[8] Google Cloud Compute. 2023. Overview of BigQuery storage. Retrieved November 26th, 2023 from https://cloud.google.com/bigquery/docs/storage_overview

[9] Bogdan Ghita. 2019. Public bi benchmarks. Retrieved November 26th, 2023 from https://github.com/cwida/public_bi_benchmark

[10] Google. 2023. *Data definition language (DDL) statements in GoogleSQL.* Retrieved November 26th, 2023 from https://cloud.google.com/bigquery/docs/reference/standard-sql/data-definition-language#on-demand-query-size-calculation

[11] Google. 2023. Introduction to External Data Sources. Retrieved November 26th, 2023 from https://cloud.google.com/bigquery/docs/external-data-sources

[12] Google. 2023. Replicate data to Bigquery in near real-time with Datastream. Retrieved November 26th, 2023 from https://cloud.google.com/datastream/docs/quickstart-replicationto-bigquery

[13] Noritaka Sekiyama. 2021. Stream data from relational databases to Amazon Redshift with upserts using AWS Glue streaming jobs. Retrieved November 26th, 2023 from https://aws.amazon.com/blogs/big-data/stream-data-from-relational-databases-to-amazon-redshift-with-upserts-using-aws-glue-streaming-jobs/

[14] Amazon Web Services. 2023. Amazon RDS for MySQL. Retrieved November 26th, 2023 from https://aws.amazon.com/rds/mysql/

[15] Amazon Web Services. 2023. Amazon Redshift. Retrieved November 26th, 2023 from https://aws.amazon.com/redshift/

[16] Mike Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O'Neil, Pat O'Neil, Alex Rasin, Nga Tran, and Stan Zdonik. 2005. C-Store: A Column-Oriented DBMS. In *Proceedings of the 31st International Conference on Very Large Data Bases* (Trondheim, Norway) *(VLDB '05)*. VLDB Endowment, 553–564.

[17] TPC. 2023. TPC-H Version 2 and Version 3. Retrieved November 26th, 2023 from https://www.tpc.org/tpch/