CS 764: Topics in Database Management Systems
Lecture 25: Snowflake

Xiangyao Yu
11/30/2022
Announcements

Exam solutions announced on Piazza

Exam grade announced on Canvas
  – per-question grades are emailed to individuals
The Snowflake Elastic Data Warehouse

Benoît Dagolle, Thierry Cournède, Marcin Zukowski, Vadim Antonov, Arin Avanes, Jon Book, Jonathan Daybaugh, Daniel Engledow, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Ashish Motwani, Abdul Q. Munir, Steven Pelley, Peter Povhien, Greg Rahm, Spyridon Triantafyllis, Philipp Unterbrunner

Snowflake Computing

ABSTRACT
Snowflake is the new golden age of distributed computing. Public cloud platforms now offer virtually unlimited storage and compute resources on demand. At the same time, the software-as-a-service (SaaS) model for large enterprise-class systems to users who previously could not afford such systems due to their cost and complexity. Also, traditional data warehousing systems are struggling to fit into this new environment. For one thing, they depend on complex ETL pipelines and physical tuning to be able to handle the flexibility and business requirements of the cloud's new types of unstructured data and rapidly evolving workloads.

Snowflake is the first system that is specifically designed to build an enterprise-wide data warehousing solution for the cloud. The result is the Snowflake Elastic Data Warehouse, or "Snowflake," for short. Snowflake is a multi-tenant, transactional service, highly scalable and easy to use system with SQL support and high performance for semi-structured and relational data. The system is offered as a pay-per-service in the Amazon cloud. Users upload their data to the cloud and can immediately manage and query it using familiar tools and interfaces. Implementation begins in late 2015 and Snowflake has been generally available since January 2016. The system can be a great match for a growing number of small and large organizations alike. The system runs several million queries per day over multiple petabytes of data.

In this paper, we describe the design of Snowflake and its novel multi-tenant, shared-data architecture. The paper highlights some of the key features of Snowflake, showing elasticity and scalability, semantic structure and schema-less data, time travel, and enhanced security. It concludes with lessons learned and an outlook on ongoing work.

Categories and Subject Descriptors
Information systems; [Data management systems]; Database management system engines

Keywords
Data warehousing, database as a service, multi-cloud shared-data architecture

1. INTRODUCTION
The advent of the cloud enables a move away from software delivery and operation on local servers, and toward shared services that are delivered as a pay-per-use service. Many platform providers such as Amazon, Google, or Microsoft.

The shared infrastructure of the cloud provides enormous economies of scale, extreme scalability and availability and a pay-pay-per-use model that adapts to unprecedented usage demands. But these advantages can only be captured if the software itself is able to scale dynamically over the pool of commodity resources that is the cloud. Traditional data warehousing solutions present the cloud. They were designed to run on small, static clusters of well-behaved machines, making them a poor architectural fit.

We describe the design of Snowflake, a system that has been both as well. It is based on the fact that most of the data is a data warehouse once it has been used by the organization. The system is running in production for thousands of companies, and the system has been evaluated.

In response to these requirements, parts of the data warehousing community have turned to "Big Data" platforms such as Hadoop or Spark. While these tools are indispensable for data source-wide processing tasks, and the open-source community continues to make big improvements such as the Apache Hadoop [25], they still fall much of the efficiency and feature set of established data warehousing technology. Not surprisingly, they require significant engineering effort to build and maintain. (See open source and commercial implementations of the same open source workloads which can benefit from the economies, elasticity, and service aspects of the cloud, but which are not well served by other traditional data warehousing technology or...
On-Premises vs. Cloud

**On-premises**
- Fixed and limited hardware resources
- **Shared-nothing** architecture

**Cloud**
- Virtually infinite computation & storage, Pay-as-you-go price model
- **Disaggregation** architecture
Shared Nothing – Advantages

Scalability: horizontal scaling
• Scales well for star-schema queries
Shared Nothing – Disadvantages

Heterogeneous workload
- Static resource provisioning cannot adjust to heterogeneous workloads

- More CPU intensive
- Less CPU intensive
Shared Nothing – Disadvantages

- Heterogeneous workload
- Membership changes
  - Add a node: data redistribution
Shared Nothing – Disadvantages

Heterogeneous workload
Membership changes
• Add a node: data redistribution
• Delete a node: similar to the fault tolerance problem
Shared Nothing – Disadvantages

- Heterogeneous workload
- Membership changes
- Online upgrade
  - Similar to membership change but affect all nodes
Multi-Cluster Shared-Data Architecture

Control layer

Compute layer

Storage layer
Data format: PAX

Data horizontally partitioned into immutable files (~16MB)
- An update = remove and add an entire file
- Queries download file headers and columns they are interested in

Intermediate data spilling to S3
Architecture – Virtual Warehouse

T-Shirt sizes: XS to 4XL

Elasticity and Isolation

- Created, destroyed, or resized at any point (may shutdown all VWs)
- User may create multiple VWs for multiple queries
- Determine the VW size based on performance and cost requirements
Local caching

– S3 data can be cached in local memory or disk
Architecture – Virtual Warehouse

Local caching
– S3 data can be cached in local memory or disk

Consistent hashing
• When the hash table (n keys and m slots) is resized, only n/m keys need to be remapped
Architectural – Virtual Warehouse

Local caching
- S3 data can be cached in local memory or disk

Consistent hashing
- When the hash table (n keys and m slots) is resized, only n/m keys need to be remapped
Architecture – Virtual Warehouse

Local caching
- S3 data can be cached in local memory or disk

Consistent hashing
- When the hash table (n keys and m slots) is resized, only n/m keys need to be remapped
Architecture – Virtual Warehouse

Local caching
– S3 data can be cached in local memory or disk

Consistent hashing
• When the hash table (n keys and m slots) is resized, only n/m keys need to be remapped
Architecture – Virtual Warehouse

Local caching
- S3 data can be cached in local memory or disk

Consistent hashing
- When the hash table (n keys and m slots) is resized, only n/m keys need to be remapped
- When a VW is resized, no data shuffle required; rely on LRU to replace cache content
Architecture – Virtual Warehouse

Local caching
- S3 data can be cached in local memory or disk

Consistent hashing
- When the hash table (n keys and m slots) is resized, only n/m keys need to be remapped
- When a VW is resized, no data shuffle required; rely on LRU to replace cache content

File stealing to tolerate skew
Architecture – Virtual Warehouse

Execution engine
- Columnar: SIMD, compression
- Vectorized: process a group of elements at a time
- Push-based
Architecture – Cloud Services

Multi-tenant layer shared across multiple users

Query optimization

Concurrency control
- Isolation: snapshot isolation (SI)
- S3 data is immutable, update entire files with MVCC
- Versioned snapshots used for time traveling

Pruning
- Snowflake has no index (same as some other data warehousing systems)
- Min-max based pruning: store min and max values for a data block
High Availability and Fault Tolerance

Snowflake Web UI, BI Tools, ETL Tools, ODBC, JDBC, Python...

Load Balancer

Cloud Services

Metadata Storage

Data Storage

Data Center

Data Center

Data Center

S3

fowdd.io-DB.es

Stateless services

fowdd.io-DB.

S3.
High Availability and Fault Tolerance

Snowflake Web UI, BI Tools, ETL Tools, ODBC, JDBC, Python ...

Load Balancer

Cloud Services

Metadata Storage

Replicated metadata (FoundationDB)

Data Storage

Data Center
High Availability and Fault Tolerance

One node failure in VW
- Re-execute with failed node immediately replaced
- Re-execute with reduced number of nodes

Whole AZ failure
- Re-execute by re-provisioning a new VW

Hot-standby nodes
High Availability and Fault Tolerance

S3 is highly available and durable
Online Upgrade

Deploy new versions of services and VWs

Previous version terminates after active queries finish
Semi-Structured Data

Extensible Markup Language (XML)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<customers>
    <customer>
        <customer_id>1</customer_id>
        <first_name>John</first_name>
        <last_name>Doe</last_name>
        <email>john.doe@example.com</email>
    </customer>
    <customer>
        <customer_id>2</customer_id>
        <first_name>Sam</first_name>
        <last_name>Smith</last_name>
        <email>sam.smith@example.com</email>
    </customer>
    <customer>
        <customer_id>3</customer_id>
        <first_name>Jane</first_name>
        <last_name>Doe</last_name>
        <email>jane.doe@example.com</email>
    </customer>
</customers>
```

JavaScript Object Notation (JSON)

```json
{
    "orders": [
        {
            "orderno": "748745375",
            "date": "June 30, 2088 1:54:23 AM",
            "trackingno": "TN0039291",
            "custid": "11045",
            "customer": {
                "custid": "11045",
                "fname": "Sue",
                "lname": "Hatfield",
                "address": "1409 Silver Street",
                "city": "Ashland",
                "state": "NE",
                "zip": "68003"
            }
        }
    ]
}
```
Transform (e.g., converting to column format) adds latency to the system
ETL vs. ELT

Optimization for Semi-Structured Data

Automatic type inference

Hybrid columnar format
  – Frequently paths are detected, projected out, and stored in separate columns in table file (typed and compressed)
  – Collect metadata on these columns for optimization (e.g., pruning)
Q: What are the limitations of Snowflake’s design?
Limitations of current Snowflake design and potential research directions

- Decoupling of compute and ephemeral storage
- Deep storage hierarchy
- Pricing at sub-second timescales
Distributed Ephemeral Storage

Intermediate data is short-lived
- Need low-latency and high throughput
- Strong durability not needed
- Caching of intermediate data vs. persistent data
- Query scheduling: locality-aware task + work stealing
Lakehouse Architecture

Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics

Michael Armbrust 1, Ali Ghodsi 1,2,3, Reynold Xin 1, Matei Zaharia 1,3

1UC Berkeley, 2Stanford University

Abstract

This paper argues that the data warehouse architecture as we know it today will either be in the coming years and be replaced by a new architectural pattern, the Lakehouse, which will (i) be based on open source data formats, such as Apache Parquet, (ii) have first-class support for machine learning and data science, and (iii) offer superior or the same performance. Lakehouses can help address some major challenges with data warehouses, including data staleness, scalability, cost of ownership, data tech lock-in, and limited use-case support. We discuss how the industry is already moving toward Lakehouses and how this shift may offer a new data management. We also report results from a Lakehouse system using Parquet that is a competitor with popular cloud data warehouses on TPC-DS.

1 Introduction

This paper argues that the data warehouse architecture as we know it today will either be in the coming years and be replaced by a new architectural pattern, which we refer to as the Lakehouse, characterized by (i) open source data formats, such as Apache Parquet and ORC, (ii) first-class support for machine learning and data science workflows, and (iii) better or the same performance.

The history of data warehousing started with helping business leaders get analytical insights by collecting data from operational databases into centralized warehouses, which data could be used for decision support and business intelligence (BI). Data in these warehouses would be written with schema on write, which ensured that the data model was optimized for downstream BI consumption. We refer to this as the first-generation data analytics platform.

A decade ago, the first generation system started to face some challenges. First, they typically required compute and storage in an on-premises appliance. This forced enterprises to provision and pay for the peak of read and write data management, which became very costly and unscalable. Second, not only did data grow rapidly, but more and more dashboards were being constructed, e.g., sales, audit, and test datasets, which data warehouses could not store as fast and at a cost.

To solve these problems, the second generation data analytics platforms started offloading all core data into low-cost storage systems with the aim that both data is persistent and easily open file formats, such as Apache Parquet and ORC. This approach started with the Apache Hadoop ecosystem, using the Hadoop File System (HDFS) for cheap storage. The data lake was a schema-less architecture that enabled the agility of storing any data at low cost, but on the other hand, the problem of data quality and governance downstream.

In this architecture, a small subset of data on the lake would later be ETL’d to a downstream data warehouse (such as Teradata) for the most important decision support and BI applications. The use of open formats also made data lake data directly accessible to a wide range of analytics engines, such as machine learning engines, as well as expensive BI systems.

From 2015 onwards, cloud data lakes, such as Hadoop on AWS and Azure, started replacing HDFS. They have experience talk (often >10 years), pre-replication, and most importantly, extremely low cost with the possibility of automatic, even cheaper, archival storage, e.g., KIOX. The cost of the architecture is largely the same as the cloud in the second generation system, with a downstream data warehouse such as Redshift or Snowflake. This two-tier data lake + warehouse architecture is now dominant in the industry in our experience and virtually all Fortune 500 enterprises.

This brings us to the challenges with current data architectures.

While the cloud data lake and warehouse architecture is extremely cheap, due to separate storage (e.g., S3) and compute (e.g., Redshift), a two-tier architecture is highly effective for cost. In the first generation platform, all data was transferred from operational data systems directly into a warehouse. In today’s architecture, data is first ETL’d into lakes, and then again ETL’d into warehouse, covering compute, delays, and cost failure modes. Moreover, enterprise use cases now include advanced analytics such as machine learning, for which data lakes now need new technologies that could lead to significant inefficiencies.

Reliability: Keeping the data lake and warehouse consistent is difficult and costly. Continuous engineering is required to ETL data between two systems and make it available in high-performance decision support and BI. Each ETL step also incurs failures or introducing bugs that reduce data quality, e.g., due to subtle differences between the data lake and warehouse engine.

Data staleness: the data in the warehouse is stale compared to the data in the lake, with new data frequently taking days to load. This is a step back compared to the first generation of analytics systems, where new operational data was immediately available for queries. According to a survey by Dimensional Research and Forrester, 86% of analysts use out-of-date data and 70% report waiting on engineering resource consumption times past yearly.

Limited support for advanced analytics: businesses want to ask predictive questions using their warehousing data, e.g., “What customers should I offer discounts to?” Despite much research on the efficiency of ML and data management, none of the leading ML systems, such as TensorFlow, PyTorch and XGBoost, work well on top of warehouses. Unlike in a database, which stores a small amount of data, these systems need to process large datasets using complex non-SQL code. Reading this data via JDBC/JDBC is inefficient, and there is no way to directly access the internal

This article is published under the Creative Commons Attribution License http://creativecommons.org/licenses/by/4.0/
Data Warehouse vs. Data Lake

Lakehouse = Data warehouse + data lake
Snowflake – Q/A

Key differences between Snowflake vs. Databricks?
Can we use cloud functions instead of VMs?
Can Snowflake work with other backend like Azure?
Does Snowflake support transactional workloads?
Does spawning VWs for every query incur considerable overhead?
Submit review for
– Yifei Yang, et al., FlexPushdownDB: Hybrid Pushdown and Caching in a Cloud DBMS. VLDB, 2021