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

Benoit Dagreville, Thierry Crouces, Marcin Zukowski, Vadim Antonov, Arin Avanes, Jon Book, Jonathan Dlaybaugh, Daniel Engroff, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Asish Moitra, Abdul Q. Murr, Steven Pelley, Peter Povhivak, Greg Rahn, Spyridon Testafyllis, Philipp Unterbrunner

Snowflake Computing

ABSTRACT

We live in the golden age of distributed computing. Public cloud platforms now offer virtually unlimited compute and storage resources on demand. At the same time, the Software-as-a-Service (SaaS) model offers enterprise-grade systems to users who previously could not afford such resources 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 in order to deliver the flexibility and business requirements of the cloud's new type of data storage and compute environment.

For these reasons, the idea of an all-in-one database that could be deployed in the cloud naturally came to the developers of Snowflake. The goal was to build an enterprise-ready 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 fault-tolerant with full SQL support and built-in connectors for many structured and unstructured data. The system is offered as a pay-as-you-go 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 began in late 2012 and Snowflake has been generally available since late 2014.

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: interactive elasticity and scalability, transactional structure and columnar data, fine-grained, user-defined security. It concludes with lessons learned and an outlook on ongoing work.

Categories and Subject Descriptors

Information systems [Data management systems]: Database management systems: Engine

Keywords

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

1. INTRODUCTION

The advent of the cloud enables a move away from software dependency on local servers in favor of a cloud-based service, removing the need to purchase, install, configure, and maintain expensive infrastructure. Today’s cloud platforms provide everything from relational databases to low-latency file systems. Cloud storage providers such as Amazon, Google, or Microsoft.

The cloud infrastructure of the cloud providers removes many concerns of scale, extreme availability and availability, and the pay-as-you-go model that adapts to unpredictable usage demands. But these advantages can only be captured if the software itself is able to scale efficiently across the pool of commodity resources that is the cloud. Traditional data warehousing solutions provide the cloud. They were designed to run on an on-premise, multi-cluster of dedicated machines, making them a poor architecture for the cloud.

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: interactive elasticity and scalability, transactional structure and columnar data, fine-grained, user-defined security. It concludes with lessons learned and an outlook on ongoing work.

In response to these shortcomings, parts of data warehousing community have turned to “Big Data” platforms such as Hadoop or Spark. While there are no substitutes for data sources or processing tools, and the open-source community continues to make big improvements such as the Apache initiative [19], they still lack much of the efficiency and feature set of unimpeded data warehousing technology. The most important, they require signifi-

SIGMOD 2016
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
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
Architecture – Storage

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
Architecture – Virtual Warehouse

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
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
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
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

Stateless services

Cloud Services

Metadata Storage

Data Storage

Data Center  Data Center  Data Center
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

Data Center

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?
A Follow-up Paper

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
Abstract
This paper argues that the data warehouse architecture as we know it today will wither in the coming years and be replaced by a new architectural pattern, the Lakehouse, which will be based on open stack data infrastructure such as Apache Parquet, (iii) layer-based disaster recovery and backup support for machine learning and data science, and (iii) ability to stream data into and out of the system. lakeside can help address several major challenges with data warehouses, including scalability, elasticity, and control over the cost of data, including data warehouses and data science workflows, and is not part of the architecture.

The history of data warehousing started with helping business leaders get analytical insights by collecting data from operational databases into centralized warehouses, which could be used for decisions 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 systems started to face several challenges. First, they were built on top of traditional data warehousing technologies, which were not well-suited to the needs of modern businesses. Second, these systems required significant upfront costs and capital expenditure. But more and more businesses were facing the need to make real-time decisions, and they were looking for a way to make their data work for them.

To solve these problems, the second generation data analytics platforms started offering data lakes that could store data in a variety of formats, including structured and unstructured data, such as Apache Parquet and ORC. This approach is referred to as the Lakehouse movement, which combined the benefits of a data lake with the flexibility of a data warehouse. This movement has led to the development of new generations of data analytics platforms, which are designed to be more scalable and cost-effective.

1 Introduction
This paper argues that the data warehouse architecture as we know it today will wither in the coming years and be replaced by a new architectural pattern, the Lakehouse, characterized by (i) open stack data formats, such as Apache Parquet and ORC, (ii) layer-based disaster recovery and backup support for machine learning and data science workflows, and is not part of the architecture.

The history of data warehousing started with helping business leaders get analytical insights by collecting data from operational databases into centralized warehouses, which could be used for decisions 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 systems started to face several challenges. First, they were built on top of traditional data warehousing technologies, which were not well-suited to the needs of modern businesses. Second, these systems required significant upfront costs and capital expenditure. But more and more businesses were facing the need to make real-time decisions, and they were looking for a way to make their data work for them.

To solve these problems, the second generation data analytics platforms started offering data lakes that could store data in a variety of formats, including structured and unstructured data, such as Apache Parquet and ORC. This approach is referred to as the Lakehouse movement, which combined the benefits of a data lake with the flexibility of a data warehouse. This movement has led to the development of new generations of data analytics platforms, which are designed to be more scalable and cost-effective.

Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics
Lakehouse Architecture
This article is published under the Creative Commons Attribution License http://creativecommons.org/licenses/by/4.0/ in: CIDR'19, January 30-31, 2019, Dallas.
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?
Before Next Lecture

Submit review for