CPSC 5990 - Query Processing in Multi Cloud Systems

Naveen Kumar Vadlamudi

Spring 2023

Contents

1 Introduction to Cloud

In this section, we briefly summarize the definition of cloud computing and introduce the fundamental ideas of what can be called a cloud from the perspective of founding authors who had created a greater impact on the cloud community.

1.1 What is Cloud and Cloud Computing

As per the authors Armbrust et al. [17] a cloud is defined as a combination of data center hardware and software, whereas cloud computing is defined as the delivery of cloud resources over the Internet providing services to users.

1.2 Who is a Cloud User

Cloud users [17] are the privileged users over a public/private cloud platform, that has the ability to provision resources and build applications based on their need. Generally cloud users are developers working for an organization or the organization itself which interacts with cloud services.

Key Functions of a cloud user:

1. Maintaining security standards to preserve data confidentiality.
2. Scaling up Services when required.
3. Discontinuing services whenever not needed.

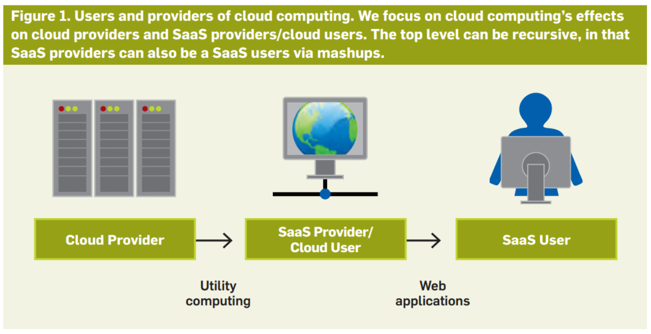


Figure 1. Users and providers of cloud computing. We focus on cloud computing's effects on cloud providers and SaaS providers/cloud users. The top level can be recursive, in that SaaS providers can also be a SaaS users via mashups.
Cloud Provider
SaaS Provider/ Cloud User
SaaS User
Utility computing
Web applications

Figure 1: Cloud User

1.3 Types of Clouds

According to the literature, there are four types of clouds defined

1. Public Cloud
2. Private Cloud
3. Hybrid Cloud
4. Multi Cloud

1.3.1 Public Cloud

Public clouds [22] are privately owned cloud platforms, that offer their resources such as compute and storage over the Internet, through a pay-as-you-go model.

Some of the top public cloud providers in the technological world are AWS (Amazon Web Services), GCP (Google Cloud Platform), Azure, and Digital Ocean. In this model, all the hardware, software and other supporting infrastructure are managed and maintained by the cloud vendor.

1.3.2 Private Cloud

Private cloud [22] is a dedicated cloud for an entity or organization that uses a data center to scale up and scale down its resources based on necessity. A private cloud is one in which the services and infrastructure are maintained on a private network. Typically organizations build their own private clouds. Otherwise they use a third party service provider to establish one. Google, Facebook, Apple has their own private clouds to maintain their user traffic and data in their data centers.

The primary difference between public and private cloud is, mainly the lack of utility computing [17] in private cloud. However, in general there are few key aspects where private cloud is better such as

1. Ownership and Control
2. Accessibility
3. Costs
4. Customization and Flexibility

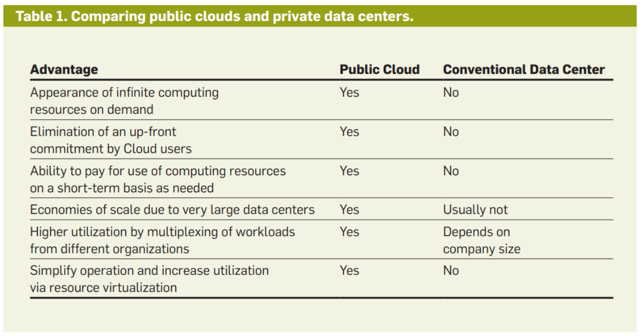


Table 1. Comparing public clouds and private data centers.

Advantage	Public Cloud	Conventional Data Center
Appearance of infinite computing resources on demand	Yes	No
Elimination of an up-front commitment by Cloud users	Yes	No
Ability to pay for use of computing resources on a short-term basis as needed	Yes	No
Economies of scale due to very large data centers	Yes	Usually not
Higher utilization by multiplexing of workloads from different organizations	Yes	Depends on company size
Simplify operation and increase utilization via resource virtualization	Yes	No

Figure 2: Public Vs Private Cloud

1.3.3 Hybrid Cloud

A Hybrid cloud [22] is a mixed computing environment which utilizes the functionality of both public and private clouds. This allows for data interoperability between applications in both environments. The organizations implementing hybrid cloud utilizes the benefits of both environments that contains features ranging from security to scalability.

Example: Hybrid cloud concept is synonymous to a hybrid vehicle that has the ability to run on both gas and electricity. In this kind of model, the vehicle utilizes the features of the system based on the need.

1.3.4 Multi Cloud

A Multi Cloud [10] deployment model is an environment, which integrates two or more public cloud platforms to run any specific applications instead of using a single vendor. This deployment model helps organizations to be fool proof.

1.3.5 Can private clouds be called a cloud platform?

In the early days of the community there was contention about whether to consider private cloud as a cloud platform, because it was considered as a data center run by an organization. However, when the community designed and finalized its core principles, then it was included as separate entity.

1.4 Cloud Native Applications

Applications that are specifically designed [8] and developed to scale dynamically over a cloud are called cloud native applications. Cloud native applications [8] prioritize scalability, flexibility and resiliency to meet the demands of modern cloud architectures. Nowadays many organizations are developing products that are cloud native applications.

An example of a cloud native applications is Snowflake a Cloud native Data warehouse offered as DBaaS (Database as a service) and scales elastically whenever the demand for compute is needed.

1.5 View of Cloud Computing

The study conducted by Armbrust et al. [17] aimed to define cloud computing and eliminate confusion by providing clear terminology. They distinguished between cloud and conventional computing using simple diagrams and briefly explained the classes of utility computing and the economics of cloud computing, emphasizing its importance when considering any platform. Additionally, the authors identified and compared the top technical and non-technical challenges and opportunities in Cloud Computing, providing a basis for further improvement.

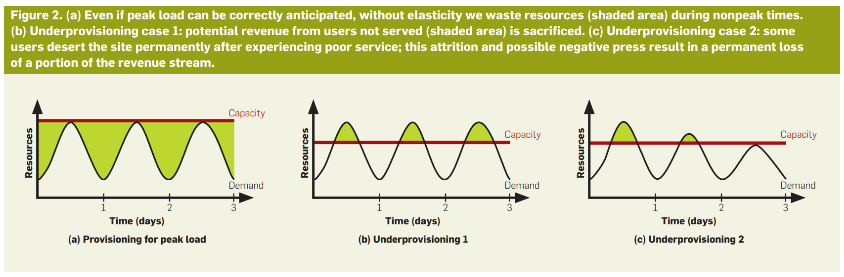


Figure 2. (a) Even if peak load can be correctly anticipated, without elasticity we waste resources (shaded area) during nonpeak times. (b) Underprovisioning case 1: potential revenue from users not served (shaded area) is sacrificed. (c) Underprovisioning case 2: some users desert the site permanently after experiencing poor service; this attrition and possible negative press result in a permanent loss of a portion of the revenue stream.

Figure 3: Depiction of Rate of Provisioning in Cloud

The authors defined cloud computing as the delivery of system software, applications, and hardware over the Internet, for providing services to users. They also subclassified cloud services into three models, PAAS (Platform as a Service), SAAS (Software as a Service), and IAAS (Infrastructure as a Service). Furthermore, the authors mentioned that computation, storage, and network communication are the three fundamental classes of utility computing needed for building applications. They highlighted some products like EC2 (Compute), S3 (Storage), and App engine from various public cloud vendors such as AWS (Amazon Web Services), GCP (Google Cloud Platform), and Azure. They also presented the concept of cloud economics with necessary use cases, discussing unexpected spikes from users and the provisioning of resources by managing cost associativity. Later, the authors succinctly explained the difference between cloud and conventional computing, citing features such as infinite compute resources on demand, elimination of upfront commitment by cloud users, and the ability to pay for use on demand. This simplifies operations and improves utilization using resource virtualization. Private data centers or conventional computing environments do not support the above principles; instead, public cloud handles the problem of under-provisioning and over-provisioning. To support these principles, the authors provided a use case from Animoto, a company that started offering its services on Facebook and experienced a spike in users. This led them to launch compute from 50 to 3500 servers in three days, and later reduced the number of servers when the demand subsided. Therefore, this problem correlates with identifying under-provisioning, scaling up based on demand, and also connects with elasticity and pay-as-you-use, addressing the operational requirement.

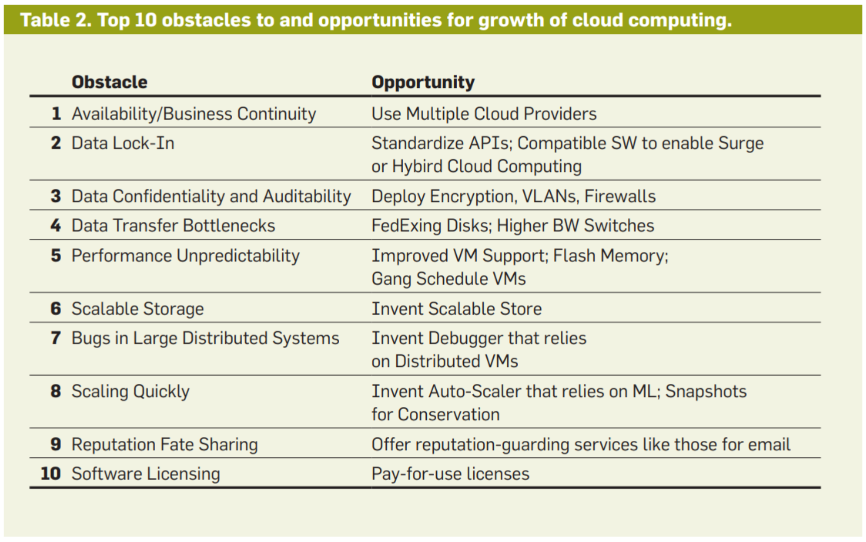


Table 2. Top 10 obstacles to and opportunities for growth of cloud computing.

	Obstacle	Opportunity
1	Availability/Business Continuity	Use Multiple Cloud Providers
2	Data Lock-In	Standardize APIs; Compatible SW to enable Surge or Hybird Cloud Computing
3	Data Confidentiality and Auditability	Deploy Encryption, VLANs, Firewalls
4	Data Transfer Bottlenecks	FedExing Disks; Higher BW Switches
5	Performance Unpredictability	Improved VM Support; Flash Memory; Gang Schedule VMs
6	Scalable Storage	Invent Scalable Store
7	Bugs in Large Distributed Systems	Invent Debugger that relies on Distributed VMs
8	Scaling Quickly	Invent Auto-Scaler that relies on ML; Snapshots for Conservation
9	Reputation Fate Sharing	Offer reputation-guarding services like those for email
10	Software Licensing	Pay-for-use licenses

Figure 4: Obstacles and Opportunities in Cloud Computing

Furthermore, the authors identified ten obstacles and opportunities in the field of cloud computing, where the first three are around adoption, the next five affect growth, and the last two are about policy. Some of the obstacles and opportunities mentioned are regarding business continuity and availability, where everyone expects 24/7 availability. The authors suggested that having a multi-cloud strategy is the solution. Data lock-in is another issue when using the cloud because every vendor has its own formats and standards. To solve this problem, the authors suggested standardizing APIs across platforms to enable rapid adoption in hybrid cloud environments. Other obstacles include data confidentiality, transfer bottlenecks, and performance unpredictability, which can be resolved by maintaining strong firewalls, encrypting data, increasing the bandwidth of disks, and improving VM (Virtual Machine) support. Additionally, scalable storage, dynamic scaling, and bugs in large-scale distributed systems can be rectified by inventing scalable storage and creating an auto scaler technique that uses machine learning for dynamic scaling, and by developing a debugger specifically for distributed VMs. Lastly, for reputation fate sharing and software licensing, the authors advocated having reputation-guarding services and pay-for-use licenses.

2 Introduction to Cloud Databases

In this section, we will briefly examine what a cloud database is and how it differs from an on-premises database. Later, we will also discuss different types of deployments and conclude this section by mentioning the model that is frequently used.

2.1 Definitions

A database is organized collection of structured information or data, that is stored electronically on a computer, and managed by a database management system (DBMS).

2.2 Cloud Database

A cloud database, is special type of database that is developed, built, deployed and accessed specifically over all cloud environments, such as private, public and hybrid.

2.3 Deployment models of Databases in the Cloud

In order to run a database on a cloud environment, there must exist specific models that a system can smoothly execute with out any interruption. Therefore it can be done primarily, in two ways and those are ?

1. Traditional Deployment
2. Database as Service (DBaaS)

2.3.1 Traditional Deployment

Running DBMS software on on-premises systems or on a cloud virtual machine is considered a traditional deployment. This kind of deployment involves manual backup of data by dedicated administrators and must require a resource to continuously monitor the database instance. Cloud Users can typically get machine-optimized images from a cloud service provider or can install them on their own.

Example:

1. Mysql installed on AWS(Amazon Web Services) is a traditional deployment where the data is stored and processed on a single instance.
2. Its similar to Mysql database running on a local desktop.

2.3.2 Database as a Service (DBaaS)

In the DBaaS model, organizations usually sign up for a subscription model with a cloud service provider. Activities such as scaling, backup and maintenance are all taken care by service provider and involves significant automation.

Example:

1. Snowflake cloud data warehouse is a SaaS (Software as a Service) product that eliminates manual administration of scaling, backup and replication.
2. It also provides unlimited compute for the data stored and supports scalable analytics.

2.4 Data Models

There are multiple data models in the database community. Some of them include:

Hierarchical Data Model, Network Data Model, Object oriented Data Model, Relational Data Model, Entity Relationship Data Model, Dimensional Data Model, Document Data Model, Column-family Data Model, Graph Data Model, Spatial Data Model, Time Series Data Model

Among all the above mentioned models, the relational data model is the most widely used one in the database community. These databases are used for handling high throughput transaction processing, in the context of order management and customer service engagement. The document data model and graph based models are the next highly used, among the community in industrial settings.

2.5 Types of Database Systems

There are mainly two types of database systems

1. OLTP (Online Transaction Processing) : A database system that is highly optimized for processing transactions in real time, and performing high throughput updates.
2. OLAP (Online Analytical Processing): Online Analytical Processing systems are designed to perform huge analytical workloads.

2.6 Evolution of Cloud Databases

Initially,the database community was completely focused on developing database products that are dominant in the area of unary machine processing, but due to adoption of internet, many companies started to innovate based on their rising needs, to handle the incoming data into the system. Therefore it all started in 2003, with the creation of distributed file system, GFS [26], where the demand for fault tolerance of files was need of time. Later the, MapReduce [25] paper was released which laid the foundation for multiple papers in the area of big data domain, but for the scope of this report, we will stick to the topic of databases and file systems which are core in understanding cloud databases and the internals that regard to compute and distribution of data across multiple areas.

3 Distributed Computing

In this section, we will briefly talk about core distributed computing principles and the programming model that helps us in understanding big data systems and cloud based databases. The topics that we discuss are the following:

1. MapReduce
2. CAP Theorem

3.1 MapReduce

The authors, Ghemawat et al. [25], proposed the MapReduce programming model. The model consists of a map function that process a key/value pair and generates a set of intermediary key/value pairs, and a reduce function that merges all intermediate key/value pairs as aggregated result set. MapReduce programming model uses distributed computing and has master and slave components. The programs written in this model utilize masters ability to hold multiple data structures and are by default parallelized and executed on a large cluster of commodity machines [25]. Partitioning of data, scheduling and handling machine failures are taken care by run-time system.

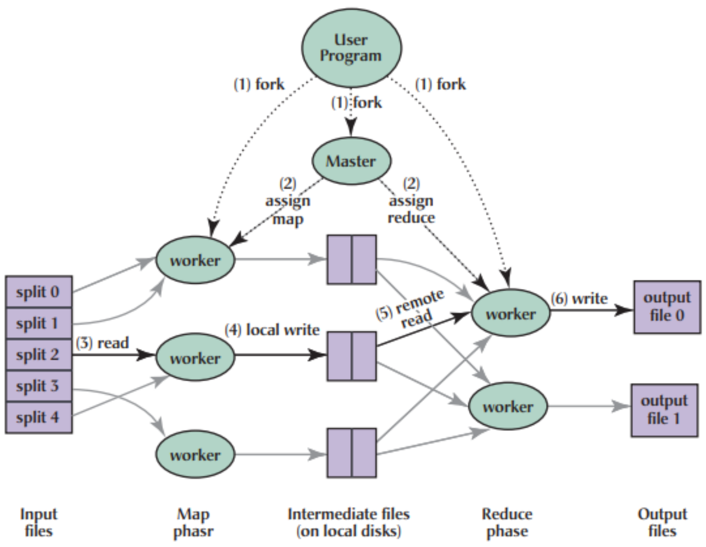


Figure 5: Map Reduce Data Flow

The authors [25], explained the implementation of the MapReduce model over a distributed file system and highlighted its key features regarding execution of jobs and fault tolerance. Later the authors compared the performance of map reduce jobs on 1 TB of data using varied strategies, and found that the memory consumption increased in map operation and decreased in reduce operation. Overall the MapReduce paper presents the programming model and delves into the implementation and its internal workings of the cluster.

3.2 CAP Theorem

The Brewer [20], proposed a fundamental theorem that highlights the trade-off between the properties of consistency, availability, and partition tolerance [20]. The CAP theorem states that it is impossible to simultaneously achieve all three properties in any distributed system.

Consistency refers to a system state that has the most updated data in all of its nodes. Availability is the process of ensuring that the system is continuously available to the users in the event of a failure. Partition tolerance refers to the ability of the system to continue functioning despite network/communication failure between the nodes.

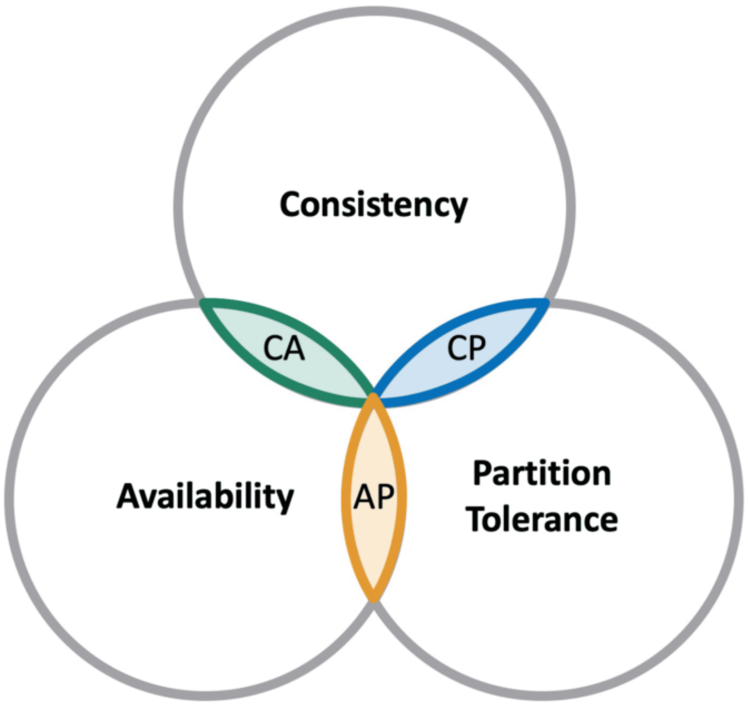


Figure 6: CAP Theorem Image taken from Online Sources

In the event of a communication failure, a distributed system must choose between consistency and availability. The author mentions that if the system chooses consistency, it will be unavailable to the users until the network partition is resolved. Similarly, if the system chooses availability over consistency, then the the service continues to run uninterrupted, but will result in data inconsistency. Overall, the author states the trade-off between necessary properties and establishes a correlation between them in a distributed system environment.

4 Distributed File Systems

In this section, we will discuss the definition of distributed file systems and their importance. Later, we will summarize the internal workings of the Google File System and briefly explore the Hadoop File System by introducing its core components.

4.1 Definitions

Distributed File Systems are the file systems that span across multiple file servers or multiple locations. These systems makes accessibility easy and provides high fault tolerance towards user data.

Example: There are many distributed file systems out in the literature, some of them include - Network File System, Andrew File System, Google File System, HDFS (Hadoop File System). These file systems when modified and deployed on a cloud, they are called cloud object stores. A notable example is AWS S3, it uses a distributed file system to replicate its user data and provides fault tolerance and high availability.

4.2 Google File System

The work done by authors Ghemawat et al.[26] resulted in the creation of a novel distributed file system, which was specially designed for rising workloads at Google. For this, they used commodity hardware to build a working model, discussed its internal functionality, and explained its features, such as fault tolerance and replication strategies. In addition, they compared and contrasted workload breakdown between the research and development cluster and the production cluster. Finally, they also documented their experiences in building the above product.

As per the authors [26], existing distributed file systems like MGFS (Minnesota Google File System), GPFS (General Parallel File System), NASD (Network Attached Storage Device), and AFS (Andrew File System) use different design techniques and algorithms to store the data in an optimized way. As a result, every system has its own performance limitations due to these design choices. Therefore, the authors state that a standardized distributed system with high reliability and performance is much needed.

The authors proposed that Google File System contains 3 main components: a) master, b) clients and c) chunkservers

Characteristics of each component:

Master: A commodity server that stores metadata and files mapping information in it

1. The job of the master is to redirect all the requests from the clients to their respective chunkservers.
2. It chooses a primary chunkserver initiating lease by following the system protocol.
3. The master maintains 2 tables in memory for quick access.
4. One maps the file names to an array of chunk ids, while the other monitors the list of chunkservers, version numbers, and lease expiration parameters.

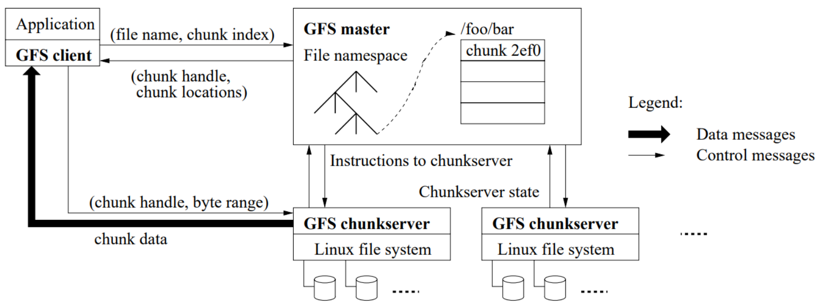


Figure 7: Google File System Architecture

5. A log file is also kept on the master local disk which maintains the checkpoints of all the operations including state management, version tracking, and chunk handle management.
6. Chunk handle management involves assigning a unique 64-bit identification number to every block.

Client: The end user applications that dump data into a distributed file system

1. The task of a client is to generate a file write request and partition the incoming file into 64 MB chunks when it needs to be saved on disk.

Chunkservers: Following are the Chunkserver functionalities,

1. Chunkservers communicate with clients after a lease agreement is initiated by the master.
2. It writes the partitioned chunks to its disks as 64 MB blocks.
3. Each block is replicated by a standard replication factor of three to ensure high availability of a particular file.
4. When an update is made to an existing chunk, first the client enquires with the master regarding which chunkserver holds the current lease for the requested chunk. Then the master checks, if the lease is acquired or not. If not it gets one and sends information regarding primary and secondary chunkservers to the client.
5. Thereafter, the client caches this information for future needs and contacts the master again only when the primary chunkserver becomes unreachable.
6. The client pushes the data to its primary and secondary chunkservers, which store data in LRU buffer cache until it is utilized or expired.
7. Once all the replicas send an acknowledgement saying that data is received, now the client generates a write signal to the primary chunkserver.
8. Later the primary assigns serial numbers to all the changes it receives, and asks secondary chunkservers to perform a write request in a specified order.

9. Once the write operation is performed the secondary server notifies the primary server saying that it has completed the write, and the primary responds to the client, stating it has successfully executed its task.

10. In this way, if there are any write issues at the downstream servers the operation is retried until it is completed.

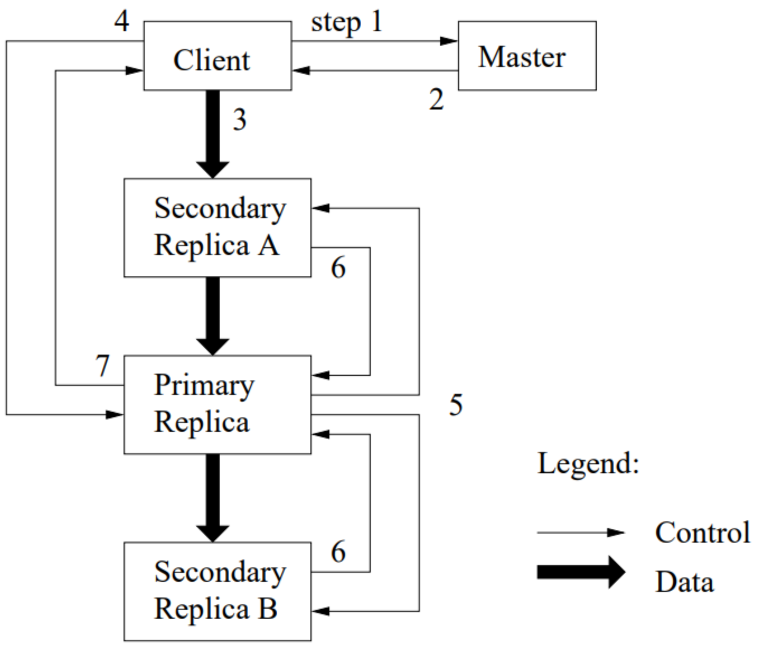


Figure 8: Read and Write flow of Data in Google File System

Evaluation: The authors evaluated the cluster by deploying it at a data center. They collected metrics regarding the real-time usage of the system with 19 servers (which include chunkservers, master, master replica) connected to one switch and 16 clients connected to another switch with a capacity of 1 GBPS. Overall for research and development cluster they proposed that the theoretical and pragmatic efficiency of a read operation, for single and multiple clients are commendable with 80 % and 75 %. This exemplifies that the read operation supports the cluster to operate effectively at peak loads with single and multiple clients. Similarly, the practical working write operation has almost 50 % efficiency in concurrent clients when compared to theoretical a measure due to the chance of high collision while writing data to the chunkservers at the same time. For record appends, for a single client it starts at 6.0 MB/s and drops to 4.8 MB/s for concurrent clients due to congestion and variation of transfer rates by multiple clients in the network.

The authors provided multiple research directions from the standpoint of distributed computing. These include building a fault-tolerant and highly available system, by citing early sources like AFS (Andrew file System), XFS, Intermezzo, and Minnesota GFS. Also, they provided more information regarding the differences between the existing architectures and laid a solid foundation by providing novel algorithms that regards to fault tolerance and handling huge volumes of data reliably.

4.3 Hadoop File System

Shvachko et al. [30] proposed the Hadoop File System (HDFS) as a distributed storage system that provides a framework for analysis and transformation of very large datasets. They mentioned that incapability of existing systems to process large volumes of data gave rise to this project. Hadoop uses MapReduce programming model to execute jobs over a cluster of machines. It also leverages distributed computing capabilities, such as high fault tolerance, reliability and cluster computation. The main characteristic of Hadoop is its partitioning of data, and computation across thousands of hosts and executing application jobs in parallel close to data storage.

The Hadoop architecture consists of a name node, a data node and a HDFS client, which enables the cluster to write data efficiently. The process of file writing, begins with a client application interacting with the HDFS cluster and it also contains these core components:

1. Hbase: Column-oriented Service
2. Pig: Dataflow Language and Parallel execution framework
3. Hive: Data warehouse Infrastructure
4. ZooKeeper: Distributed Coordination service that is widely used, in coordinating and managing distributed applications.
5. Chukwa: Data Collection System for monitoring large-scale distributed systems.
6. Avro: Data Serialization System

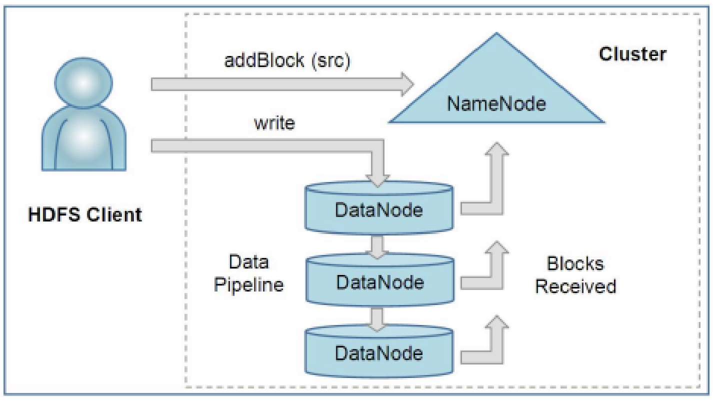


Figure 9: Hadoop Architecture

Overall the authors focused on describing and discussing the architecture of HDFS (Hadoop Distributed File System) by detailing its internal working of various components.

5 Column Oriented Databases and File Formats

In this section, we will first define and explore the advantages of a column-oriented database. We will then introduce the need for having such a database. Later, we will provide a concise summary of the C-Store implementation and proceed to discuss Open file formats and their types in more detail.

Definitions and Advantages: The name column oriented implies that data is stored as columns instead of rows. This kind of data organization is helpful in specific scenarios such as:

1. Query performance: Column oriented databases are useful while performing data mining operations such as filtering and aggregation that involves reading data from complex data sets.
2. Data compression: In column oriented databases, data compression can be tailored according to the data type of objects present in the column. This results in higher compression, rates, in contrast with row oriented databases.
3. Data Aggregation: In column oriented databases, aggregation operations are very efficient due to its columnar structure. Therefore, finding the minimum value, maximum value and aggregation operations such as sum, count, and avg are optimal.
4. Horizontal and Vertically Scalability: Columnar databases can handle large datasets using horizontal and vertical scaling mechanisms. Horizontal scaling can increase the compute power of a cluster by adding extra nodes to the existing ones. In vertical scaling the existing machine is replaced with a new configuration.
5. Compression and Inmemory processing: To achieve faster processing, modern day systems leverage the usage of main memory to perform in memory compression and processing.

5.1 Need for Column Oriented Databases

The demand to handle and analyze large volumes of data has triggered the need for powerful OLAP engines that aid in faster analysis result. This has led to creation of various columnar engines like MonetDB, Sybase, and C-Store, InfinitiDB. Among them we are going to look into C-Store for this section. In respect to context of cloud data warehouses such as Snowflake, C-Store database is taken as reference.

5.2 C-store

The authors Stonebraker et al. [31] studied that existing write optimized database systems are highly efficient for OLTP (Online Transaction processing) style applications, and discovered that read efficient systems are needed for the adhoc querying of large amounts of data. Therefore, they proposed creating a new system named C-Store, which is a column oriented database system that stores every column data as a series of values using effective compression techniques.

The authors discussed the internal architecture of C-Store, with a focus on the implementation of various layers. In this process, they discussed more regarding the data model implementation, and query processing algorithms that are covered, in implementing the database. They also compared its performance to various commercial databases. Overall C-store performed exceptionally superior than existing commercial databases like Sybase.

5.3 Open File Formats

In this section, we will define and propose the aims of the Open File Format. Later, we will look into summaries of Apache Parquet [6] and Apache ORC [5]. The reason for including this section here is that Open File Formats are the foundation of Big Data Processing (Section 7) and Data Lake Table Formats (Section 8.2), which are, in turn, used in the Lakehouse architecture (Section 8.3). So, understanding these systems is the basis for comprehending further systems discussed in other sections.

Definition: Open file formats are designed to store and process large volumes of data efficiently. They provide a structured and organized way to access data by providing high order compression. The primary aims of file formats are:

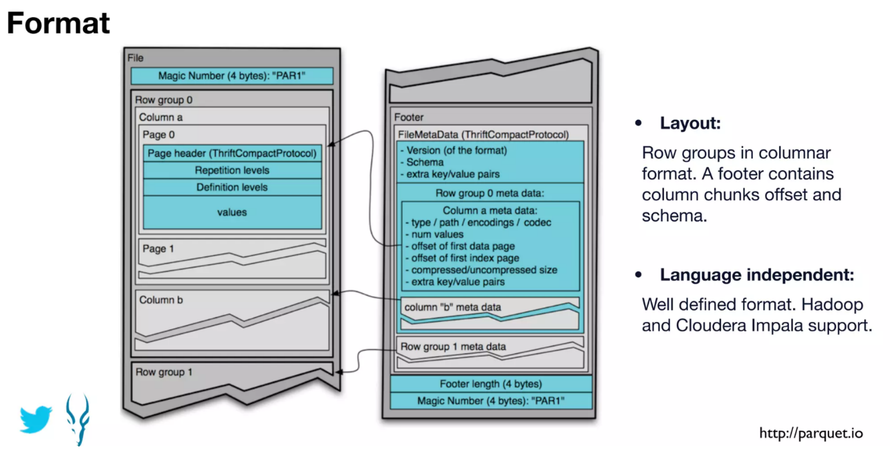
1. Data Organization: File format aims to maintain data in a structured manner, defining how its elements are represented and stored.
2. Data Interoperability: File formats provide a standard way of communication with various systems, by providing a common language for data representation.
3. Efficiency: File formats aims to reduce file sizes using various compression techniques following a hybrid storage strategy. Which helps in effective query processing and reduced bulk load times.
4. Data Integrity: File formats include effective mechanisms like checksum and error detection that ensures data integrity over file transmission.
5. Schema Management: Many file formats support specifying schema for type validation and type checking.
6. Accessibility and Portability: File formats aim to be widely accessible and portable across different platforms, software applications, and programming languages. They provide a standardized way to read, write, and manipulate data, ensuring that data can be seamlessly exchanged and processed across various environment.

5.3.1 Apache Parquet

Apache Parquet [6] is an open-source, structured, column-oriented file format specifically designed for efficient data retrieval and storage. It offers enhanced performance, efficient data compression, and encoding schemes to handle complex bulk data. Parquet utilizes a hybrid storage approach to effectively group information by rows and columns.

Parquet [6] implements the record shredding and assembly algorithm proposed by Dremel [29]. It also provides support for schema encoding on each column and offers flexibility to incorporate new encoding techniques as they are developed. Parquet has emerged as the de facto standard for Big Data file formats and is widely adopted in major Big Data tools such as Apache Spark [34] and Open Table Formats like Apache Hudi [2], Apache Iceberg [3], and Delta Lake [16].

The structure of Parquet [6] is comprised of row groups, where rows are grouped as a single entity, and columns are horizontally partitioned and stored. Additionally, Parquet [6] utilizes column chunks that are stored contiguously in the file, enabling faster read times. In summary, the documentation discusses the use cases for Apache Parquet and provides crucial information and context to developers regarding optimization strategies for its usage.



- **Layout:**
  Row groups in columnar format. A footer contains column chunks offset and schema.
- **Language independent:**
  Well defined format. Hadoop and Cloudera Impala support.

Figure 10: Parquet File Format Storage Organization

5.3.2 Apache ORC (Optimized Row Columnar)

Apache ORC (Optimized Row Columnar) [5] is a self-describing, type-aware column oriented file format that speeds up the performance of Apache Hive [32]. It enables high-speed processing that aims to reduce file sizes on Hadoop [30]. ORC is optimized for huge streaming reads and builds an internal index when the file is written. The data in the file format is stored as stripes with a default size of 64 MB. Each stripe contains the column information separately so that the column information can read the data independently. ORC [5] also uses predicate pushdown to determine what data to be read from an underlying file format and supports all the data types of Hive [32]. In addition, it includes complex data types such as structs, maps, lists, and unions.

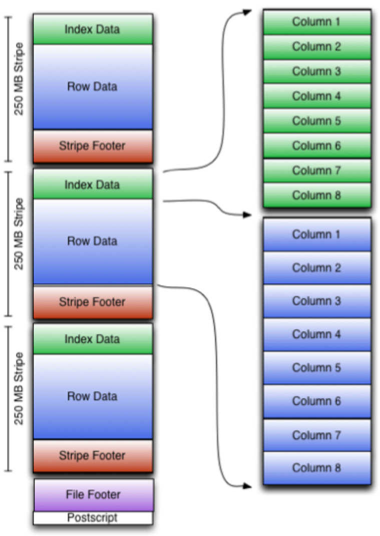


Figure 11: ORC File Format Storage Organization

As of now, ORC supports various distributed file systems and data processing systems such as Apache Hive [32], Apache Spark [34], Apache Hadoop [30], and Apache Arrow [1]. It also maintains three levels of indexes:

1. File level - statistics about the values in each column across the entire file
2. Stripe level - statistics about the values in each column for each stripe
3. Row level - statistics about the values in each column for each set of 10,000 rows within a stripe

However, ORC also supports ACID properties in conjunction with other frameworks like Apache Hive [32]. Overall, the ORC file format helps in reducing the bulk load times and query execution times using its optimized file layout strategies.

6 OLTP Cloud Databases

In this section, we will look into the cloud based OLTP database systems by describing its data organizing and processing mechanisms. After going through this section the reader understands the working of foundational cloud based OLTP systems such as Google BigTable, Spanner and Amazon Aurora.

6.1 Big Table

With the widespread adoption of the Internet, users are utilizing the features of various products developed by Internet companies such as Google. This has led to the demand for a resilient, fault-tolerant storage system to handle incoming structured data from applications. The authors Ghemawat et al. studied the problem of structuring and versioning huge volumes of data that accumulates in Google data centers from various applications.

The authors mentioned that projects like Boxwood aim to provide a distributed infrastructure for building higher-level services, such as databases and file systems, but that the protocols for handling structured data were not present. This made the existing system not highly beneficial. Similarly, several vendors created parallel databases such as the Oracle Real Application Cluster Database, which uses shared disks and distributed lock managers to handle data. In the same way, IBM DB2 Parallel Edition also uses a shared-nothing architecture similar to Bigtable, but they handle data in the context of relational databases. All the discussed databases and file systems do not directly support client applications that wish to store huge data or they must be tailored as per Google's growing data needs.

Working of Big table: The authors proposed BigTable [21], a distributed structured storage system that meets the capacity and scaling needs of Google. It is a sparse, distributed, persistent, and multi-dimensional sorted map that is indexed using a timestamp, column key, and row key as a unique string of an uninterrupted array of bytes. Initially at Google, it was used to perform web page indexing, and referred to as a web table. Each row has a size of 64 KB, and all the unique strings identified in a table are lexicographically sorted and stored in a tablet. A Tablet is data structure that holds multiple rows as a group for better row locality of reference. Similarly, columns are grouped into sets called column families where access control and memory accounting are performed, and are denoted by the following syntax (family: quantifier). Timestamps are used for indexing to create multiple versions of the same data.

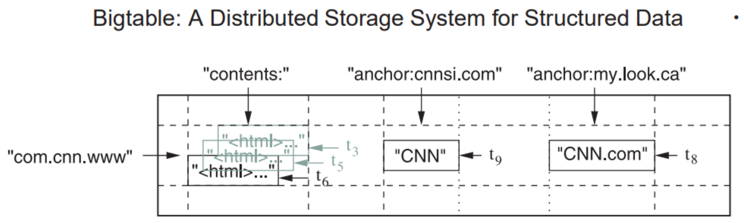


Figure 12: Big Table Row Structure

BigTable uses the Google File System (GFS) [26], a distributed file system that stores data across multiple clusters, in the form of log and data files. Typically Big Table relies on cluster management system software for scheduling jobs and monitoring machine status. Internally, every row is stored in the Google SSTable file format. A persistent ordered immutable map of keys and values with each block of size 64 KB (which is configurable) that loads Block Index into memory to search for other blocks. A Block Index is a special index in Bigtable which contains metadata that maps data blocks to their locations in a distributed file system.

BigTable relies on Chubby, which is a highly available distributed lock service that has five active replicas and uses the Paxos algorithm to maintain the consistency of its replicas. One server is elected as master among the pool and other nodes are annotated as tablet servers that have tablets.

BigTable maintains a three-level hierarchy similar to a B+ tree as a) Root tablet, b) Metadata tablets, and c) User tables to store tablet location information. The first level is a file stored in Chubby that contains the location of the root tablet. The root tablet maintains the location of all tablets in a special metadata table. Each metadata tablet contains the location of user tablets, and the root tablet itself is the first tablet in a metadata table. Similarly, all the tablets that are tracked are present in the user table (where all the data rows that are inserted are present) with reference to the metadata table.

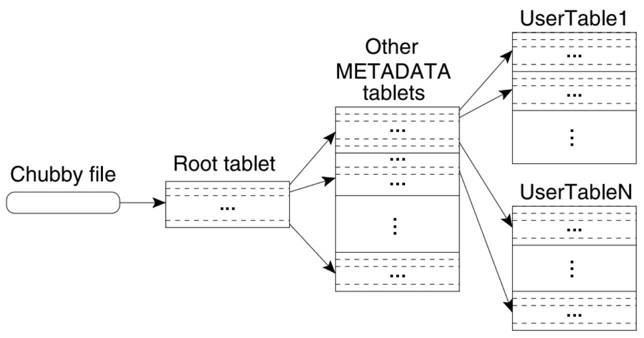


Figure 13: Tablet Location Hierarchy

Data records are persistently stored in GFS using the SSTable file format, which stores redo records in a commit log. Recently committed ones are stored in the primary memory of a tablet server and is referred to as a memtable (An in-memorysorted buffer).

When a client initiates a read request, the tablet servers that are provisioned by the master node acquires an exclusive lock on a uniquely named file in a specific chubby directory. If there are any network connectivity issues, it retries the connection until it reobtains the lock, and later on, it will terminate itself if the file path does not exist in Chubby.

Similarly, when the tablet server does not respond to master messages, it tries to acquire a lock on the chubby file and if it succeeds then it is live. Otherwise it implies the tablet server is dead. Therefore it will be replaced by the master and reassigns all of its tablets back to unassigned state. Similarly, all the write operations also follow the sequence of steps. Big table is currently running in production and used by many products internally in Google.

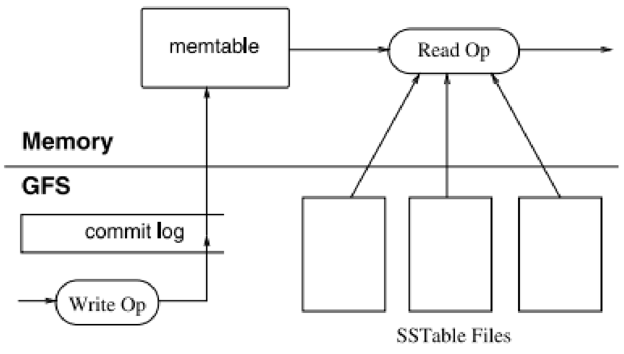


Figure 14: Tablet Representation

The authors conducted several tests on Bigtable using a cluster of N tablet servers. Each tablet server was configured to use 1 GB of memory and a GFS [26] cell consisting of 1786 cells, with two 400 GB IDE hard drives each. The tests were conducted to evaluate the performance of Bigtable and its ability to handle a large volume of data. The results showed that Bigtable was able to handle petabytes of data and provided high availability, scalability, and fault tolerance.

The authors proposed various research directions, including novel distributed file system methodologies that utilize shared-nothing architecture to provide fault tolerance and high availability.

6.2 Spanner Globally Distributed Database

Google products are some of the most popular and widely used applications worldwide, generating vast amounts of data that need to be stored and processed using highly fault-tolerant systems without compromising performance. As a result, this need has led to multiple innovations in the field of data management, including tools such as Google File System (a highly fault-tolerant distributed file system), MapReduce (a highly efficient programming model that performs aggregation computing on large datasets) [25], Bigtable (a distributed storage system for structured data) [21], and Megastore (a scalable and highly available storage for interactive services). All the above products provide exceptional data management capabilities at a regional level, but they lack at providing a unified consistent data a global level. Therefore, the authors Corbett et al. [23] studied the problem and utilized all the above mentioned services and built a novel product to address the problem of providing a globally distributed, highly available, and consistent database system for Google's applications and services.

According to the authors [23], traditional database systems have limitations in terms of scalability, availability, and latency, particularly in a globally distributed environment. Therefore, existing systems such as MySQL, Oracle, Megastore, and DynamoDB have issues with replication as they are pre-built solutions and are unable to globally distribute data due to their architecture. Additionally, these solutions require manual effort that results in high operational costs and potential anomalies.

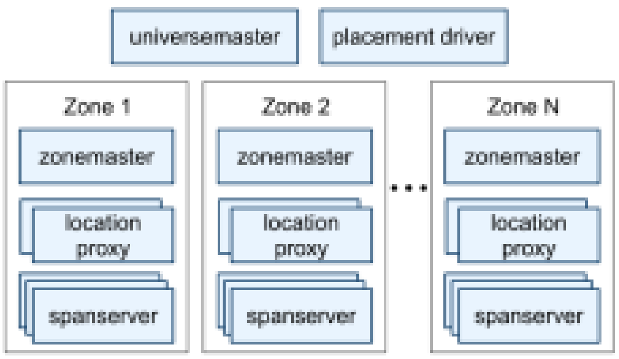


Figure 15: Spanner Organization

The authors [23] proposed Spanner, a globally distributed database system developed by Google that provides both strong consistency and high availability. It combines several techniques, including a distributed versioned storage system, a distributed consensus protocol, a TrueTime API for clock synchronization across data centers, and uses the concept of a global clock to achieve external consistency. Spanner is designed to scale horizontally and handle large amounts of data, and it provides an SQL-based interface for querying and updating data. Spanner also supports distributed transactions across multiple data centers, making it a powerful tool for building globally distributed applications.

The authors present several experiments that evaluate the performance and scalability of Spanner. The experiments demonstrate that Spanner can scale to hundreds of nodes and can handle millions of transactions per second with low latency. The experiments also show that Spanner can provide strong external consistency and high availability even in the presence of network partitions and other failures. Overall, the results demonstrate that Spanner is a highly scalable and reliable database system.

The authors conclude by discussing future research directions for Spanner, including improving its performance and scalability, supporting new data types and query languages, and integrating

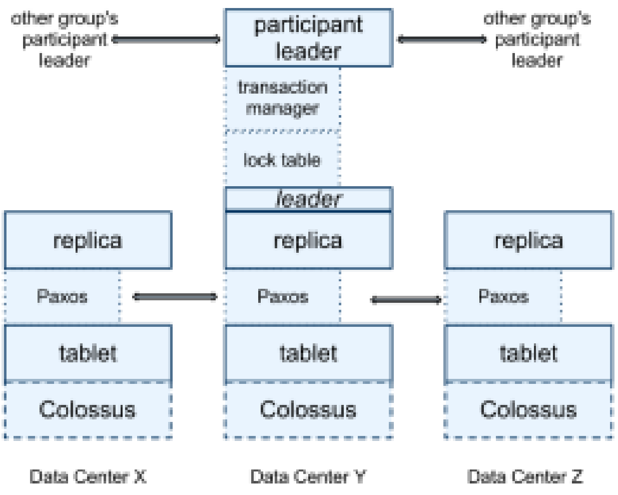


Figure 16: Spanner Software Stack

it with other Google services. The authors also suggest that the techniques used in Spanner could be applied to other distributed systems beyond databases, such as distributed file systems and key-value stores. Overall, the paper highlights the importance of developing new techniques for managing data at scale in a globally distributed environment.

6.3 Amazon Aurora

The authors Verbitski et al. [33] presents the design principles and considerations behind Amazon Aurora, an advanced cloud-native relational database service offered by Amazon Web Services (AWS). The primary objective of Aurora's design is to deliver exceptional throughput and low latency for demanding workloads. The authors highlights several key architectural elements of Aurora, including a distributed storage system that scales automatically, a fault-tolerant design that ensures high availability, and an innovative approach to database replication that eliminates the need for traditional, synchronous replication.

The authors also explore various performance optimizations employed by Aurora. It discusses how Aurora utilizes parallel query processing techniques to distribute query execution across multiple nodes, enabling faster data retrieval and analysis. Additionally, intelligent caching mechanisms are implemented to reduce disk I/O and improve overall query performance. Aurora also adopts a log-structured storage approach, which optimizes write operations by efficiently organizing data on disk.

In conclusion, the authors emphasizes the advantages of adopting Amazon Aurora for applications that require high throughput and scalability without sacrificing the familiar SQL interface of traditional relational databases.

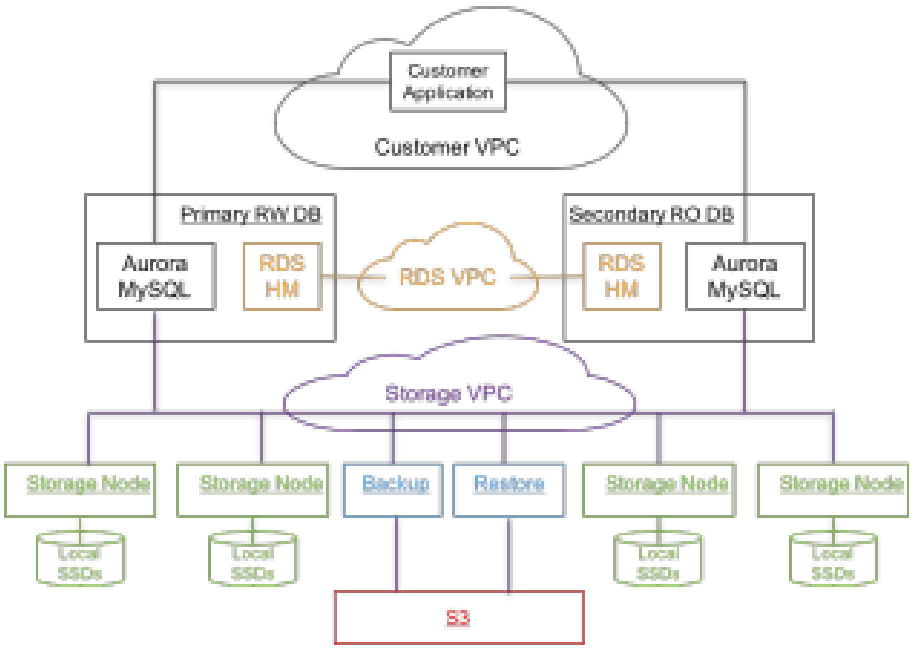


Figure 17: Aurora Architecture

7 Big Data Engines

In this section we are briefly going to define what is big data and will summarize the characteristics of big data by discussing more on how Apache Spark [34] works from the perspective of authors .

Definition: Big data engines are software packages designed to process and analyze large volumes of data in a distributed and scalable manner. These engines are essential for deriving knowledge, patterns, and value from huge datasets, that are commonly known as big data.

A few common characteristics of big data engines are:

1. Scalable Distributed Computing: Ability to scale existing system on high user demand.
2. Fault Tolerance: Ability of being resilient towards system failures.
3. Varied Data Processing Paradigms: Ability to provide batch and stream processing systems in a single framework.
4. Distributed file systems: Ability to support various file systems that enables access to files without any interruption.
5. Flexibility and Extensibility: Ability to add various libraries to the existing system and ease of integration.

7.1 Apache Spark

Through constant innovation and implementation, many organizations have introduced multiple tools to handle big data. However, every tool has its own limitations, specifically for their use cases. Therefore, a unified tool that can handle all the problems in a big data domain is a much-needed requirement. The authors Zaharia et al. [34] proposed and presented Apache Spark, a unified big data processing tool developed to solve multiple problems that originate in a big data domain.

As per the authors, Spark was created to process varied data using a single engine with the capabilities of multiple engines. It uses a programming model similar to MapReduce, known as RDD (Resilient Distributed Datasets). Similarly, it employs distributed computing to process jobs. However, the core difference lies in the computation part of RDD. Apache Spark performs compute operations using lazy evaluation. It leverages the lineage graph mechanism to process the submitted query and optimizes on run time.

Spark supports various file formats such as Parquet [6], ORC (Optimized Row Columnar) [5], and AVRO (A row-oriented remote procedure call) [7], CSV, and JSON Hive Tables. Parquet [6], in particular, is widely used as an underlying file format within the community. Additionally, Spark [34] can use any storage system, such as Hadoop file system (HDFS) [30], cloud object storage from any public vendor (S3, Google Object Store) [13, 9], and can query data based on file format [6, 5, 7]. Furthermore, various organizations are utilizing Spark for their big data environments, ranging from biotechnology to the finance industry.

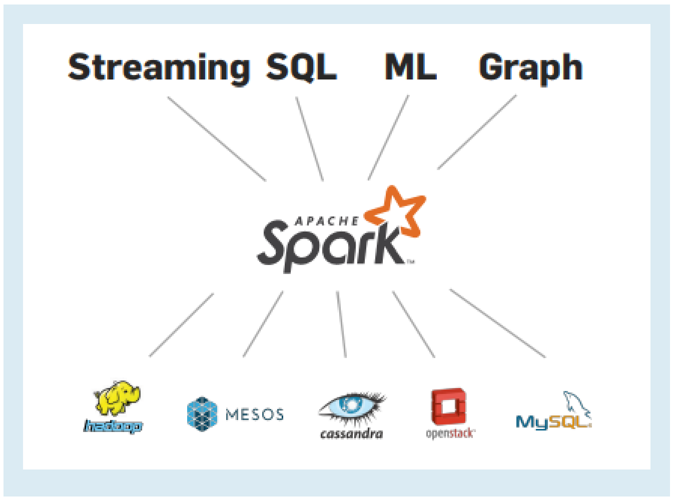


Figure 18: Apache Spark Architecture

Overall, the authors touched upon many research areas that range from distributed computing, big data technologies, and cluster computing programming models in the context of RDD (Resilient Distributed Dataset), reflecting the core ideas of their paper.

8 OLAP Based Cloud Databases

In this section, we discuss the evolution of cloud data warehouses and present the various generations of them, including the limitations of the systems. This entire section is divided into 3 parts

1. In section 8.1 will discuss about the various generations, of cloud data warehouses (CDW) and will contrast the Redshift and Snowflake architectures.
2. In section 8.2 we discuss, Open table formats and its evolution history.
3. Finally in section 8.3 we discuss, the Lakehouse Architecture.

8.1 Cloud Data Warehouses

Cloud data warehouses [14] are modern OLAP database platforms, that are deployed on a public cloud. They are optimized for providing scalable BI (Business Intelligence) and Data analytics Insight for users. Also, it enables organizations to unify and store data in the cloud, offering various advantages over on-premise systems.

8.1.1 Generations in Cloud Data Warehouses

First Generation CDW The first-generation Cloud Data Warehouses [14] were the first data warehouse types to deploy and run on public clouds. It utilized the features of cloud services, and provided support to consumers using the MPP (massively parallel processing) paradigm. Data warehouses such as Redshift and Big query are the examples in this category. Even though the first generation data warehouses did provide some flexibility for the data practitioners, it had various limitations such as:

1. Scalability and Performance: First generation Data warehouse engines frequently faced scalability issues when performing complex data processing tasks.
2. Separation of Storage and Compute: First generation systems, maintained data and computing together which caused resource under utilization issues.
3. Schema and Data Model Rigidity: Flexibility of schema and support for semi structured data are not highly supported.
4. Vendor Lock-In: The problem of vendor lock on a few cloud datawarehouses hindered organizations to interoperate data freely between one system to another.

Second Generation CDW Second generation systems such as Snowflake [14] resolved the limitations of first generation data warehouses by developing cloud native products and innovated in areas such as:

1. Separation of Storage and Compute: Snowflake created a decoupled storage and compute architecture that resulted in scaling of individual components.
2. Elasticity and On-Demand Scaling: Second-generation systems provided the potential to scale resources dynamically based on workload demands.
3. Flexibility and Schema Evolution: Second-generation systems offered substantial flexibility in data modeling, supporting semi-structured and unstructured data natively. In addition it allowed schema evolution, enabling users to adapt the data model as requirements changed without significant disruption.
4. Multi-Cloud and Hybrid Cloud Support: Second generation systems offered multi cloud approach to store the data in various platforms apart from the deployed cloud service provider.

8.1.2 First Generation - Amazon Redshift

The authors Gupta et al. [27] discusses how traditional data warehousing solutions require complex configurations and hardware provisioning for compute-intensive analysis, resulting in time-consuming processes. To address these challenges, authors introduced Redshift [27], a managed data warehousing solution featuring a Massively Parallel Processing (MPP) engine and a column-oriented, scale-out architecture. Since its launch, Redshift has gained widespread adoption in the data warehousing community due to its cost-effectiveness, reduced experimentation time, and faster generation of initial reports.

The authors, provides an overview on the system architecture, by detailing its internal components, such as storage, compute and query optimization that aid in performing very large analysis of datasets. Later, the authors mentioned dependencies of various AWS services linked with Redshift that leverages the power of existing products availability and fault tolerance. It, also emphasizes the cost effectiveness of Redshift's pay-as-you-go pricing model. The authors also presents multiple use cases of various customers, and also mentions upon the team experiences in managing thousands of database instances.

8.1.3 Second Generation - Snowflake Elastic Data Warehouse

The rise of cloud computing has provided a plethora of opportunities for handling huge computing workloads. This has triggered many organizations to innovate more SaaS (Software as a Service) products, which enabled the usability of enterprise-class systems for all ranges of customers. Similarly, Data Warehousing community was in need of a SaaS product that helps data developers to seamlessly integrate data and use it for analysis. Therefore the authors Dageville et al. [24] studied the problem of classical on-premises Data Warehousing solutions and proposed an alternative to it.

According to the authors, the integration of multiple sources and systems has led to varied data formats, which triggered huge computing needs to process the incoming data into the system. Traditional data warehousing solutions were not able to support and scale as per the rising demands due to hardware limitations. While some parts of the data warehousing community pivoted to big data platforms such as Hadoop [30] and Spark [34] to create Data Warehouses, these solutions required significant engineering effort to roll out and handle a cluster. In addition the existing systems did not provided support for varied data types. All of these existing problems demonstrate the need for a new system.

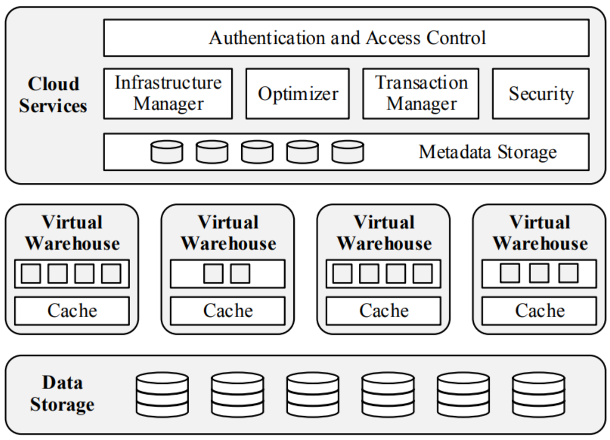


Figure 19: Multi-Cluster, Shared Data Architecture

The authors developed a novel columnar database engine named Snowflake, which consists of 3 layers : a) cloud services layer, b) the virtual warehouse layer, and c) data storage layer.

Cloud Services layer:

1. The cloud services layer is considered as the brain of a unary snowflake environment, because it contains crucial software services such as a query optimizer, a transaction manager, an infrastructure manager, a metadata storage, and security services.
2. A query optimizer, uses a cascade-style approach, following top-down cost-based optimization. It optimizes the submitted query to the engine by reducing its search space due to lack of indexing and uses delayed evaluation. Similarly, once the optimizer completes its

work, the resulting execution plan is distributed across worker nodes, and the cloud services continuously monitor the state of the query.

3. A transaction manager handles concurrency control, and ensures ACID transactions by using Snapshot isolation. It internally uses MVCC (Multi-Version concurrency control, a protocol where a copy of every amended object is stored until a certain period of time) based engine, that enables time travel and cloning whenever required.

4. The infrastructure manager monitors and provisions virtual warehouses as a typical compute cluster in Snowflake which has primary memory and disk storage to process the incoming query.

5. The metadata component tracks the statistics of all tables, and stores them as min-max ranges.

6. The security services layer, oversees the security aspects of the objects stored in Snowflake.

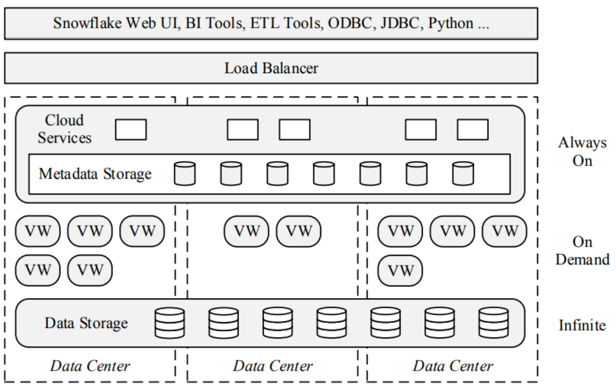


Figure 20: Snowflake Multi Data Center Deployment

Virtual Warehouse Layer:

1. The virtual warehouse layer leverages the power of compute and storage in a loosely coupled manner, and uses the convention of t-shirt size such as X-small to XX-Large.

2. A proprietary shared nothing layer caches query results data onto local disks.

3. The query results are frequently accessed, over the network, and this is defined as a multi-cluster shared data environment (when a compute cluster has multiple nodes in it).

4. The query results are accessible until the virtual warehouse is up and running. Once a virtual warehouse is suspended then the query results are flushed out from the virtual warehouse.

Data Storage Layer:

1. For persistent table storage Snowflake uses existing cloud object stores like S3 (Simple Storage Service) [13].

The authors mentioned that Snowflake is purely offered as a SaaS product where users are not needed to tune databases or perform physical backups of data because all of these activities are taken care by the engine itself. This not only enhances the user experience but also improves usability. Also, storage services are highly available with a guarantee between 99.99% and 99.999999999% of data durability. In addition, Snowflake extensively supports semi-structured data using the variant data type. It consists of many helper functions that can potentially transform the data in a structured way. Overall the authors found that, the test performance of 22 standard queries on Snowflake using relational and semi-relational data is quite acceptable, but for few queries it took significantly longer time due to sub-optimal join order.

The authors provided various research directions, which include novel query processing methodologies that utilize multi-cluster shared data architecture. Also, they leveraged the power of cloud computing and provided more details regarding MVCC architecture and time travel, cloning. Overall the authors presented a cloud-native based data warehousing solution that has the ability to scale on demand, and briefly discussed its replication and query processing techniques.

8.2 Data Lake Table Format

The Data Lake table format is an open table format [11]. It bundles all the files stored in a data lake repository as a table over cloud object stores. Some of the examples of open table formats include Apache Hudi [2], Apache Iceberg [3] and Delta Lake [16].

8.2.1 Apache Hudi

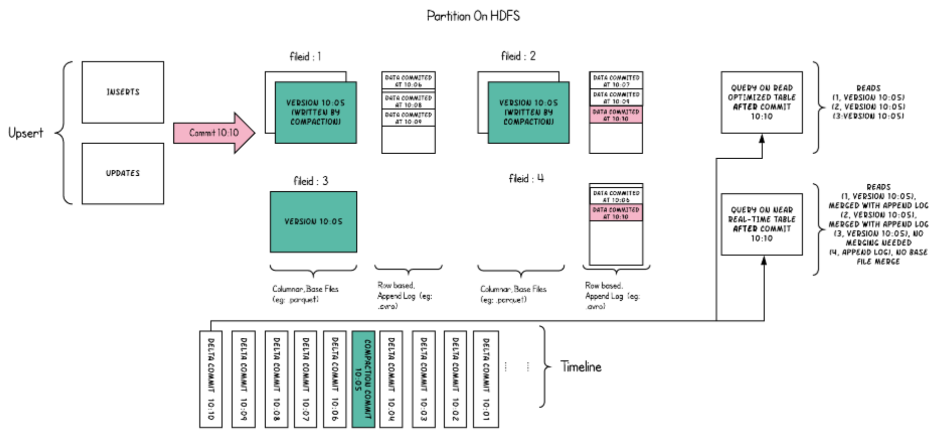

Partition On HDFS
Upsert
INSERTS
UPDATES
Commit 10:10
fileid : 1
VERSION 10:05 (WRITTEN BY COMPACTION)
DATA COMMITED AT 10:06
DATA COMMITED AT 10:08
DATA COMMITED AT 10:09
fileid : 2
VERSION 10:05 (WRITTEN BY COMPACTION)
DATA COMMITED AT 10:07
DATA COMMITED AT 10:09
DATA COMMITED AT 10:10
fileid : 3
VERSION 10:05
fileid : 4
DATA COMMITED AT 10:06
DATA COMMITED AT 10:10
Columnar, Base Files (eg: .parquet)
Row based, Append Log (eg: .avro)
Columnar, Base Files (eg: .parquet)
Row based, Append Log (eg: .avro)
QUERY ON READ OPTIMIZED TABLE AFTER COMMIT 10:10
READS (1, VERSION 10:05) (2, VERSION 10:05) (3:VERSION 10:05)
QUERY ON NEAR REAL-TIME TABLE AFTER COMMIT 10:10
READS (1, VERSION 10:05), MERGED WITH APPEND LOG (2, VERSION 10:05), MERGED WITH APPEND LOG (3, VERSION 10:05), NO MERGING NEEDED (4, APPEND LOG), NO BASE FILE MERGE
DELTA COMMIT 10:10
DELTA COMMIT 10:09
DELTA COMMIT 10:08
DELTA COMMIT 10:07
DELTA COMMIT 10:06
COMPACTION COMMIT 10:05
DELTA COMMIT 10:04
DELTA COMMIT 10:03
DELTA COMMIT 10:02
DELTA COMMIT 10:01
Timeline

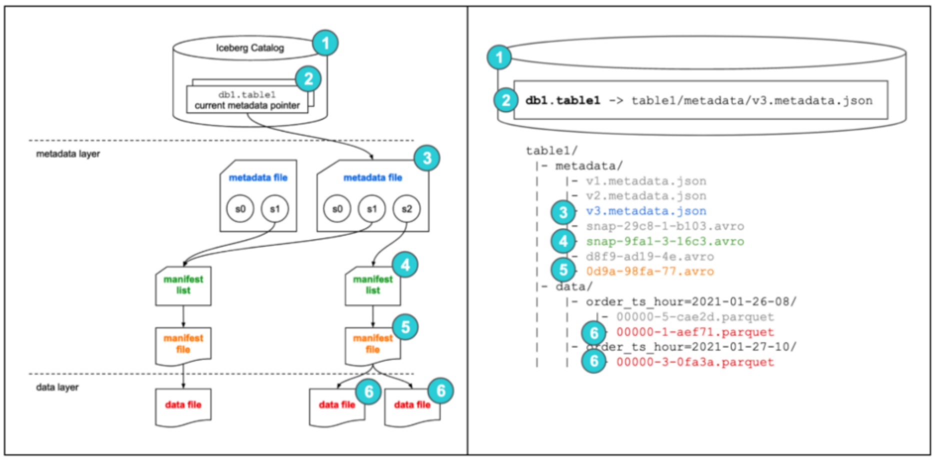
Figure 21: Apache Hudi File Format

Apache Hudi [2] (Hadoop Update Deletes and Inserts) is an Open Table Format developed by Vinoth Chandar et al. to solve Uber's internal data infrastructure issues. Initially, the project's prime focus was creating incremental data updates on existing files over HDFS. Later the authors created a library that handles incremental updates over data files on Hadoop in the first iteration. Further after the release of Deltalake [16] protocol for enforcing ACID properties on object stores the community expanded the support to all major cloud object stores that help in building Lakehouses [18].

As of now Hudi, supports schema management, Time travel, ACID transactions, and efficient upserts and deletes. More importantly, it supports Data Lake performance optimizations such as clustering, compaction indexing and Z-ordering by storing its data in Parquet [6] files in a compressed manner.

Overall the authors proposed that the developed system supports near real-time ingestion and incremental processing of pipelines using "copy on write" and "merge on read" techniques. Also Hudi can convert any file on a distributed file system (Hadoop) [30] or cloud object store [13, 9] to a table with ACID properties.

8.2.2 Apache Iceberg



```
db1.table1 -> table1/metadata/v3.metadata.json

table1/
 |- metadata/
 |    |- v1.metadata.json
 |    |- v2.metadata.json
 |    |- v3.metadata.json
 |    |- snap-29c8-1-b103.avro
 |    |- snap-9fa1-3-16c3.avro
 |    |- d8f9-ad19-4e.avro
 |    |- 0d9a-98fa-77.avro
 |- data/
 |    |- order_ts_hour=2021-01-26-08/
 |    |    |- 00000-5-cae2d.parquet
 |    |    |- 00000-1-aef71.parquet
 |    |- order_ts_hour=2021-01-27-10/
 |    |    |- 00000-3-0fa3a.parquet
```

Figure 22: Apache Iceberg File Format

Apache Iceberg [3] is an open table format project developed by the Netflix data teams initially, as a part of its business infrastructure needs. The project was later donated to the Apache Software Foundation, which currently oversees the development and maintenance. Initially, this project aimed to resolve Hive [32] related issues that were encountered, during frequent partition updates. Hive is a data warehouse built on top of HDFS (Hadoop File System) [30]. Also the other issues like stale statistics of a table, and user comprehension of physical layout of data, triggered the need for handling the problem at table level.

Therefore the authors proposed a new architecture that supports schema evolution, data versioning and proposed a new architectural pattern named the Iceberg table architecture. It contains 3 main components:

1. Iceberg Catalog: A store that maintains the metadata pointers for iceberg tables.

2. Metadata Layer: Tracks metadata information about tables, at a certain point of time.
3. Data Layer: Stores the data in the form of Parquet files.

Overall, Apache Iceberg [3] project resolved the existing problems by providing faster query evaluation and execution, and delivering consistent view of table in minimal response time. As of now the project also supports writing data to multiple cloud object stores, and follows the ACID based transactional protocol.

8.2.3 Delta Lake

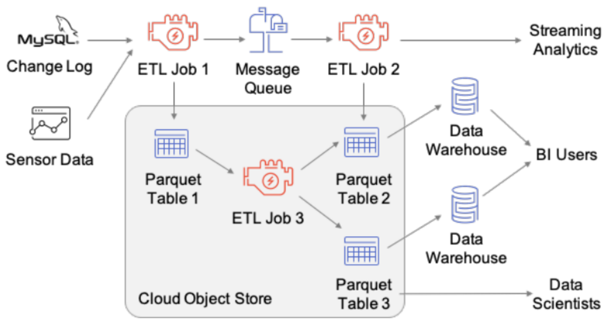
The widespread adoption of cloud computing led various organizations to use cheap storage services like S3 [13] across all domains, which triggered the need to track and version data accrued over the cloud. Since then, the concept of a Data Lake was proposed. A Data Lake is a centralized repository on cloud object stores that is used to store structured, semi-structured, and unstructured data. Regardless of various innovative architectures, however there was huge need to manage data in place over object stores. Therefore, the authors Armbrust et al. [16] studied the problem, of managing data over object stores by enforcing ACID (Atomicity, Consistency, Isolation, Durability) properties, which eliminates the need of having multiple layers in data architecture.

The authors stated that current data pipeline architectures contain separate layers for Data Lake and Data Warehouse to manage and report data. A Data Warehouse is a centralized system containing facts and dimension tables, and is highly used to generate business insights. This not only increases the level of isolation but also requires huge computing resources to build Data Warehouses. In this process, the Data Lake is used as a storage component that consists of multiple files. Therefore, the authors mention that having multiple layers not only increases the computing cost, but also increases the network cost to move the data from one layer to another. Overall they propose a new architectural pattern that can achieve the functionality of both Data Warehousing and Data Lake embedded inside a single layer.

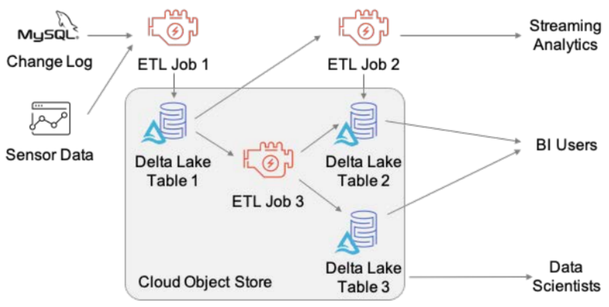
The authors proposed a system named Delta Lake that uses a novel Lakehouse architecture. It combines the functionalities of both the Data Lake and Data Warehouse by leveraging the benefits of the MVCC (Multi Version Concurrency Control) protocol and ACID properties on cloud object stores. Multi Version Concurrency Control is a technique used in database systems to manage concurrent access to data. The existing big data systems such as Spark [34] store the table using the Parquet format [6] as a bunch of objects, without maintaining strong consistency.

Delta Lake Architecture: Every table on a Delta Lake has 3 underlying files - a log file, and a data file and a checkpoint file.

1. A log file is annotated with numerical values and is stored in JSON format. It is used to manage metadata of table and every key value entry contains addition and remove operations of a particular table. The remove operation does not immediately remove the data instead it flags the record and removes it after particular time threshold.
2. A Data file columns are partitioned and organized in Parquet [6].
3. A checkpoint file is stored in Parquet [6] and is periodically compressed for optimized performance.

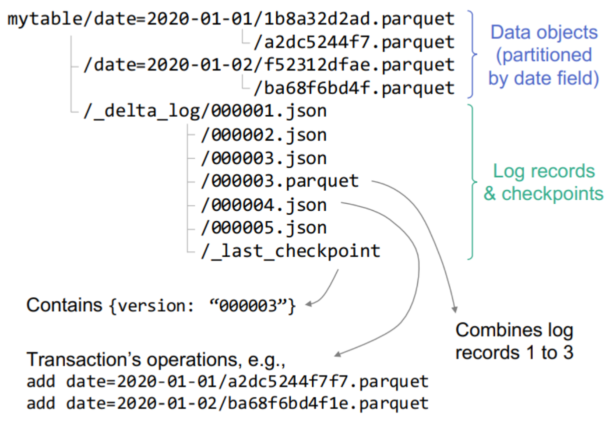


(a) Pipeline using separate storage systems.



(b) Using Delta Lake for both stream and table storage.

Figure 23: Need for Delta Lake



```
mytable/date=2020-01-01/1b8a32d2ad.parquet
                       /a2dc5244f7.parquet
       /date=2020-01-02/f52312dfae.parquet
                       /ba68f6bd4f.parquet
       /_delta_log/000001.json
                  /000002.json
                  /000003.json
                  /000003.parquet
                  /000004.json
                  /000005.json
                  /_last_checkpoint
```

Data objects (partitioned by date field)
Log records & checkpoints
Contains {version: "000003"}
Combines log records 1 to 3
Transaction's operations, e.g.,

```
add date=2020-01-01/a2dc5244f7f7.parquet
add date=2020-01-02/ba68f6bd4f1e.parquet
```

Figure 24: Table on Delta Lake

Read Operation:

1. When a read operation is initiated, the query engine pulls the last checkpoint ID from the tables log directory,
2. Later, from the above-obtained checkpoint ID the engine tries to perform a LIST operation and reconstructs the table state from most recent checkpoint.
3. Subsequently, it identifies all the add records in data objects excluding remove label records,
4. Finally it understands the objects that are relevant for read query and will parallelly query object store across a cluster using a compute engine.

Write Operations:

1. The write operations are performed to eliminate any data corruption errors that happen via parallel writes.
2. Delta Lake write transaction rate varies depending on implementation of cloud object stores, and the authors explains if a higher rate is required, then having custom log store might alleviate the problem.

Time Travel:

1. Delta Lake uses MVCC mechanism, to fetch the most recent version of data, and provides a time travel feature, to query back data in time.

Performance Evaluation: The authors evaluated the performance of Delta Lake by conducting various tests on the following parameters: (a) measuring the impact of multiple objects or partitions and (b) read and write performance of tables. For parameter (a), the authors used a file with 33 million records and partitioned data into 1000 to 1 Million partitions.

Delta Lake outperforms all the existing engines, with minimal processing latency when compared to standard Presto (a query engine for object stores and RDBMS engines) [12] and Hive (a query engine for Hadoop and object stores) [32]. Similarly, for write performance, huge tables with statistics collection do not add any significant overhead to the ingestion and are similar to other engines.

Overall, the authors provided various research directions, which include novel query processing methodologies, over cloud objects stores. Also, they mentioned that Delta Lake does not support secondary indexes and has performance issues in streaming data for sub-millisecond workloads. Therefore this could be an interesting problem to look into for implementing and optimizing the performance of Delta Lake.

8.3 Lakehouse Architecture

Lakehouse architecture is a novel data architectural pattern that combines the best features of data lakes and data warehouses. It aims to alleviate the challenges faced from existing data lakes and traditional data warehouses.

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

The rise of data has pushed innovation in the field, of data management. Companies like Uber, Netflix, and Databricks has solved the limitations of existing tools in the area of big data management, by creating new table formats like Apache hudi [2], Apache Iceberg [3], and Delta Lake [16]. This has given inspiration for organizations to revisit the existing architecture that resulted in a new optimized system named Lakehouse [18].

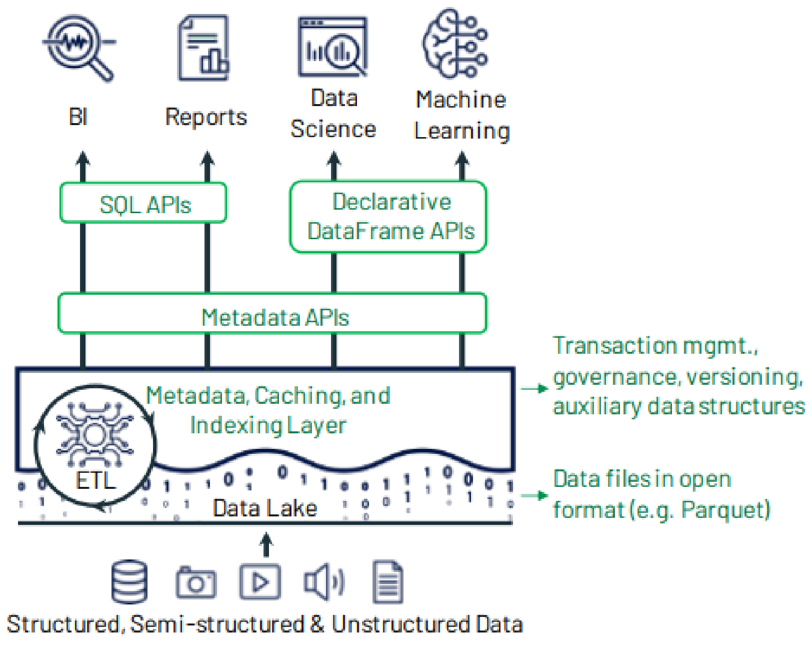

BI
Reports
Data Science
Machine Learning
SQL APIs
Declarative DataFrame APIs
Metadata APIs
Metadata, Caching, and Indexing Layer
Transaction mgmt., governance, versioning, auxiliary data structures
ETL
Data Lake
Data files in open format (e.g. Parquet)
Structured, Semi-structured & Unstructured Data

Figure 25: Apache Hudi File Format

The authors [18] proposed that the existing data warehousing architecture will become obsolete in the upcoming years. They argue that the Lakehouse architecture, with its support for machine learning, data science, and business intelligence, offers multiple advantages. The Lakehouse architecture consolidates the features of data warehouses and data lakes into a single layer, eliminating the need for multiple layers as depicted in the figure [point to the figure]. This consolidation not only simplifies the architecture but also reduces the time required for data ingestion from data lakes to data warehouses. Lakehouse enables the construction of data warehouses on object stores by performing Extract, Load, and Transform (ELT) operations from one schema to another schema. Additionally, machine learning models can be seamlessly integrated into the tables present in the Lakehouse. Data can be accessed using DataFrame APIs, allowing for efficient data reading and model training.

Overall, the authors provide a comparison between existing cloud data warehouse platforms and the features and performance of the Lakehouse system. They conclude that Lakehouse queries outperformed other systems in terms of time to execute and credits consumed (cost of query executions billed in dollars).

8.3.2 Photon Query Engine for Lakehouse Systems

The authors behm et al. [19] at databricks developed a query engine, for the workloads on Lakehouse [18] systems. Initially they addressed the need of having a separate engine for lakehouse systems, instead of directly plugging in spark query engine over deltalake. So they considered, rewriting the existing Apache Spark [34] code base that is compatible for Delta Lake [16] and has chosen vectorized execution approach that uses columnar execution.

Then after the authors, discussed various design decisions that went into developing photon and they mentioned the internal implementations of vectorized query processing over the system. Every table that is migrated on delta lake [16] is stored as bunch of Parquet [6] files. When a query is submitted, Photon engine parses and prepares an execution plan for data retrieval from deltalake [16], and it converts the Spark sql plan to a Photon plan.

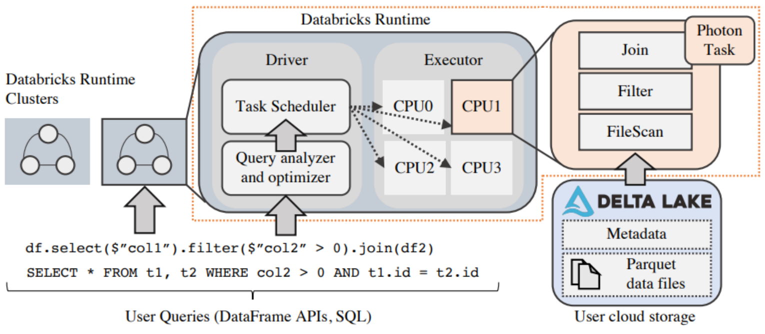

Databricks Runtime
Databricks Runtime Clusters
Driver
Executor
Task Scheduler
Query analyzer and optimizer
CPU0
CPU1
CPU2
CPU3
Join
Filter
FileScan
Photon Task
DELTA LAKE
Metadata
Parquet data files
df.select($"col1").filter($"col2" > 0).join(df2)
SELECT * FROM t1, t2 WHERE col2 > 0 AND t1.id = t2.id
User Queries (DataFrame APIs, SQL)
User cloud storage

Figure 26: Photon Query Engine

Later they compared the DBR (Databricks runtime execution) with Photons execution and published results. In conclusion, the Photon query engine outperforms the existing DBR (Databricks run time) engine. Overall the authors describe and illustrate design decisions and internal workings of Photon query engine.

8.3.3 Analyzing and Comparing Lakehouse Storage Systems

The authors studied the performance of various Lakehouse table formats, namely Apache Hudi [2], Apache Iceberg [3], and Delta Lake [16]. They began by providing an overview of Lakehouse [18] systems and contrasting the transaction coordination approaches employed by each format. The authors delved into the transaction management of different open table formats, shedding light on the intricacies involved.

Furthermore, the authors listed the metadata and update strategies associated with each format and discussed their limitations. They proceeded to describe the experimental setup used to evaluate these systems. By conducting tests on diverse data sizes that reflected real-world use cases, they generated valuable insights and published their findings.

Overall, the paper comprehensively covered key aspects of all the table formats and presented the results derived from their experiments. The conclusion reached was that Delta Lake outperforms all other systems. Please let me know if you require any further assistance or clarification.

References

[1] Apache arrow. [Online]. Available: https://arrow.apache.org/

[2] Apache hudi. [Online]. Available: https://hudi.apache.org/docs/overview/

[3] Apache iceberg. [Online]. Available: https://iceberg.apache.org/blogs/

[4] Apache iceberg issues and architecture review. [Online]. Available: https://www.dremio.com/resources/guides/apache-iceberg-an-architectural-look-under-the-covers/

[5] Apache orc. [Online]. Available: https://orc.apache.org/docs/

[6] Apache parquet. [Online]. Available: https://parquet.apache.org/docs/

[7] Avro. [Online]. Available: https://avro.apache.org/docs/

[8] Cloud native definition. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/definition

[9] Googles object store. [Online]. Available: https://cloud.google.com/storage

[10] Multicloud definition. [Online]. Available: https://cloud.google.com/learn/what-is-multicloud

[11] Opentableformat. [Online]. Available: https://airbyte.com/blog/data-lake-lakehouse-guide-powered-by-table-formats-delta-lake-iceberg-hudi

[12] Prestodb. [Online]. Available: https://prestodb.io/blog/

[13] Simple storage service. [Online]. Available: https://aws.amazon.com/s3/

[14] Snowflake definition. [Online]. Available: https://www.qlik.com/us/cloud-data-migration/cloud-data-warehouse#:~:text=A%20cloud%20data%20warehouse%20is,for%20scalable%20BI%20and%20analytics.

[15] Uber lakehouse. [Online]. Available: https://www.uber.com/en-CA/blog/ubers-lakehouse-architecture/

[16] M. Armbrust, T. Das, L. Sun, B. Yavuz, S. Zhu, M. Murthy, J. Torres, H. van Hovell, A. Ionescu, A. Łuszczak, *et al.*, "Delta lake: high-performance acid table storage over cloud object stores," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 3411–3424, 2020.

[17] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, *et al.*, "A view of computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[18] M. Armbrust, A. Ghodsi, R. Xin, and M. Zaharia, "Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics," in *Proceedings of CIDR*, 2021, p. 8.

[19] A. Behm, S. Palkar, U. Agarwal, T. Armstrong, D. Cashman, A. Dave, T. Greenstein, S. Hovsepian, R. Johnson, A. Sai Krishnan, *et al.*, "Photon: A fast query engine for lakehouse systems," in *Proceedings of the 2022 International Conference on Management of Data*, 2022, pp. 2326–2339.

[20] E. A. Brewer, "Towards robust distributed systems," in *PODC*, vol. 7, no. 10.1145. Portland, OR, 2000, pp. 343 477–343 502. [Online]. Available: https://sites.cs.ucsb.edu/~rich/class/cs293b-/papers/Brewer_podc_keynote_2000.pdf

[21] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, pp. 1–26, 2008.

[22] G. Cloud, "What is cloud computing." [Online]. Available: https://cloud.google.com/learn/what-is-cloud-computing

[23] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford, "Spanner: Google's globally distributed database," *ACM Trans. Comput. Syst.*, vol. 31, no. 3, aug 2013. [Online]. Available: https://doi.org/10.1145/2491245

[24] B. Dageville, T. Cruanes, M. Zukowski, V. Antonov, A. Avanes, J. Bock, J. Claybaugh, D. Engovatov, M. Hentschel, J. Huang, A. W. Lee, A. Motivala, A. Q. Munir, S. Pelley, P. Povinec, G. Rahn, S. Triantafyllis, and P. Unterbrunner, "The snowflake elastic data warehouse," in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 215–226. [Online]. Available: https://doi.org/10.1145/2882903.2903741

[25] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[26] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003, pp. 29–43.

[27] A. Gupta, D. Agarwal, D. Tan, J. Kulesza, R. Pathak, S. Stefani, and V. Srinivasan, "Amazon redshift and the case for simpler data warehouses," in *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, 2015, pp. 1917–1923.

[28] P. Jain, P. Kraft, C. Power, T. Das, I. Stoica, and M. Zaharia, "Analyzing and comparing lakehouse storage systems." CIDR, 2023.

[29] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: interactive analysis of web-scale datasets," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 330–339, 2010.

[30] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. Ieee, 2010, pp. 1–10.

[31] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, *et al.*, "C-store: a column-oriented dbms," in *Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker*, 2018, pp. 491–518.

[32] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive: a warehousing solution over a map-reduce framework," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1626–1629, 2009.

[33] A. Verbitski, A. Gupta, D. Saha, M. Brahmadesam, K. Gupta, R. Mittal, S. Krishnamurthy, S. Maurice, T. Kharatishvili, and X. Bao, “Amazon aurora: Design considerations for high throughput cloud-native relational databases,” in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 1041–1052.

[34] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, *et al.*, “Apache spark: a unified engine for big data processing,” *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.