Title: Disruptor in Data Engineering - Comprehensive Review of Apache Iceberg

Author: Dipankar Saha, Principal Solutions Architect/Independent Researcher, dipsaha.ias@gmail.com

Abstract
Apache Iceberg is an open source table format that has recently gained a lot of attention in the big data world. Managed under Apache Software Foundation, this project has fast become one of the industry favorites to solve some unique and complex big data problems. It is a disruptive and transformative technology which is redefining the landscape of large scale data management. Though this technology belongs in the realm of the data world, it provides unfamiliar capabilities compared to traditional data oriented technologies. As organizations continue to strive for more scalable, cost-effective, performant and reliable ways to handle data, Iceberg sits at the forefront of this to fulfill the expectation. In this paper, we begin by introducing Apache Iceberg and the numerous technical capabilities that make it unique. We discuss its architecture in detail to elucidate how it is designed to accomplish these unique capabilities. We deep dive into how this tool provides flexibility in solving different use cases, its interoperability, the cost savings opportunity it provides and why it has made its place as an attractive tool in the toolbox of organizations. We also discuss about a few other tools which perform similar jobs in this niche technology landscape and compare their features, strengths and weaknesses with Iceberg. We end the discussion with some future research opportunities that may address a few current limitations of Iceberg.

1. Introduction

It can be challenging to provide a proper definition of Iceberg to someone who has never encountered it before. While it most certainly is a technology about data, it is distinct from other traditional data-oriented technologies. Someone unfamiliar with Iceberg might ask, is it a database, is it a data warehouse, or is it some other form of database technology like the data lake of the big data world or is it the latest data buzzword in the industry “lakehouse”? The answer is it is neither of those.

Iceberg is actually just a “table format”. It doesn’t provide storage, it doesn’t provide a compute engine either. Its domain of use cases are two fold. First, it allows several compute engines like Spark [1], Flink [2], Hive [3] etc to work simultaneously with the Iceberg tables [4]. Secondly, together with a modern data processing engine, Iceberg helps in implementing the concept “data lakehouse”, a hybrid big data technology that leverages the best out of data warehouse and data lake.

Iceberg is a technology that relies on and works in conjunction with files of certain formats. Those file types that Iceberg depends on contain the actual data of a system or application. What Iceberg does is, it provides an abstraction layer to access the data efficiently stored in those data files. Iceberg accomplishes this by simply creating and maintaining metadata about the data files and intelligently creating, purging the data files behind the scene for efficient access. It provides a table like view of the data stored in the raw data files and hence the name "table format". The consumer of the data, whether human or machine, interacts with Iceberg and achieves performance, transactions and several other benefits of data access without knowing anything about the underlying functionalities and complexities of the implementation.

2. Open File Format

Before we delve deeper into Iceberg, we have to take a step back and discuss the data file formats leveraged by Iceberg under the hood. The data files in this context are not traditional file formats like CSV, Text or JSON that have been used historically for data extraction, ingestion and consumption purposes. These modern data files fall under a specific category called Open File Format. These Open File Format data types were invented almost a decade ago to support big data platforms. The motivation behind this open format was to support interoperability across multiple big data engines, avoid file copies, reduce storage cost and provide performance efficiency for data access.

Although several such file formats exist, they can primarily be categorized into two classes - row oriented file format and column oriented file format. The row oriented formats are the traditional types in the database-management-system (DBMS) world. The column oriented formats were introduced in big data systems primarily to achieve high performance [13]. The most common ones are Avro [8], Parquet [7], ORC [6] and Arrow [5]. Avro is a row oriented file format whereas Parquet and ORC are column oriented. Arrow is newer among all of these and it is an in-memory file format. Avro is the oldest in this lot and was discovered in 2009 as part of Hadoop. The ORC(Optimized Record Columnar) file was introduced by Meta in 2013. Parquet came out pretty much at the same time as the joint venture of Twitter and Cloudera [9]. Arrow was discovered in 2016.

Among these 4 formats, Parquet and ORC are more relevant in the context of Iceberg. Currently, Avro is less popular in big data solutions and its presence is mostly limited to Apache Kafka [11] streaming use cases. However Iceberg still supports Avro as one of its underlying file formats. Arrow on the other hand being an in memory is not a relevant file format for Iceberg yet. Outside of Iceberg, as general purpose file format, both Parquet and ORC are popular and used by multiple compute engines. Spark & Impala's favorite choice is Parquet, whereas ORC is favored by Hive [12].

Architecturally the file formats Parquet and ORC use compression(block compression by default) and encoding techniques to optimize storage as well as performance. They both use the standard online-analytical-processing(OLAP) compression techniques - dictionary encoding, run-length-encoding(RLE) and bitpacking [9].

The Open File Format was invented to solve several problems for big data analytics use cases which were not solvale using the traditional file formats such as CSV, TEXT, JSON or XML.

- **Efficient storage and compression** - big data applications typically contain enormous volumes of data that require huge storage space. These file formats (Parquet, ORC) store data efficiently by implementing compression and encoding techniques to reduce infrastructure overhead as well as cost.

- **Improved query performance** - due to columnar storage architecture, queries often require retrieving data for a subset of the attributes, improving query performance to a great extent.
- **Interoperability across systems** - the file formats are of open standard and consumable by multiple big data engines like Spark, Hive, Impala etc. By storing the file only in one format (Parquet or ORC), multiple engines can operate on the data in parallel thus eliminating the need for storing data in multiple formats, reducing cost as well as storage space.
- **Data integrity** - the file formats facilitate data validation to prevent errors and thus improving integrity of the data [15].

3. Open Table Format

Even though Open File Format such as Parquet/ORC solved a lot of problems in big data architecture as mentioned in the above section, there were additional problems that remained intractable [17]. As the data files were stored as independent units, they didn't provide a consolidated tabular view. As a result, query engines were unable to determine which files corresponded to a table. There were some other major problems as well.

- The files do not allow change in schema.
- Time travel over data is not possible.
- Updates are not well supported. If updates are made to multiple files, they are not atomic, ending up causing partial updates in case of failures which are difficult to rollback.

Open Table Format solves this problem by providing a metadata layer which is a set of files that contains information about the data files stored in Parquet/ORC/Avro etc formats. Query engine talks to the Open Table Format files which abstracts the data file layer and solves the limitations of the Open File Format architecture. Open Table Format provides the storage structure only by using the metadata layer, and is not a storage or a compute engine. The most common table-formats are Iceberg, Delta Lake and Hudi. The metadata implementation architecture differs for them but they all encapsulate the data files complexity and provide an abstraction layer for query engines. Iceberg implements metadata in a hierarchical manner while Delta Lake and Hudi implements tabular format metadata.

Hive is actually the first generation table-format which came before the three aforementioned table formats. Hive attempted to implement table format in HDFS without a metadata layer but ran into issues with atomicity problems and eventual consistency. Hudi and Iceberg was released in 2018, Delta Lake came a year later in 2019.

The key features provided by Open Table Format are CRUD (Create-Read-Update-Delete) operations, ACID(Atomicity-Consistency-Isolation-Durability) transactions, schema evolution, time travel and significant performance improvement in read as well as write operation, by employing various techniques under the hood [16, 17, 18]. As an example, the CRUD, ACID transactions are implemented by using the Open File Format data files (Parquet/ORC) as immutable file formats. The Open Table Format implementations create new data files when a change request is made and keeps the original file intact. This approach also allows time travel of the data as older versions of the data files exist in the storage system.

4. Data Warehouse, Data Lake and Data Lakehouse

Even though Iceberg or other table formats can be accessed by several processing engines for general purpose big data use cases, its usage is most prominent in the modern lakehouse pattern and is rapidly gaining popularity. As lakehouse itself is a fairly new concept in big data, this section discusses the evolving world of big data analytics that brought us into the era of table format based lakehouse implementation.

4.1 Data Warehouse

In the realm of traditional relational-database-management-system (RDBMS), two types of data platforms have been in use by the industry - online-transaction-processing (OLTP) systems and online-analytical-processing (OLAP) systems. The OLAP systems are typically called data warehouses, where vast amounts of data are brought in from different OLTP databases of the organization for consolidated reporting and analytics. Architecturally they have been a tightly coupled system where both storage and compute are provided by the same system. With the advent of cloud platforms and also columnar data storage format, we got another flavor of data warehouse where we were able to manage storage and compute separately, thus improving query performance and achieving scalability [10]. Data Warehouse provides atomicity-transaction-isolation-durability (ACID) properties, enforces schema and therefore ensures data quality, and typically provides good query performance through indexing and partitioning. However on the flip side, its strict enforcement of structured data format limits its flexibility. Warehouses are also not suited for advanced analytics and often lead to vendor lock-in.

4.2 Data Lake

The concept of data lake came around 2010 with the emergence of big data technologies. Data lake addressed the structured data format limitation of data warehouse as data lake was designed to support all different types of data formats - structured, semi-structured as well as raw unstructured. Data lake leverages distributed file systems like HDFS and Object store (e.g. Amazon S3) enabling storage of huge amounts of data at low cost. The principle of data lake has been to store raw data from source systems in unaltered format into the storage without an attempt to transform it or integrate with any other system before storage [19]. Having raw data from source systems allows processing and computation at a later stage as well as transforming it into different formats to support different use cases including advanced analytics and machine learning [20]. The emergence of open file formats such as Parquet and ORC brought in the benefits of interoperability, improved query performance in data lake. However on the down side, data lake was unable to enforce schema, couldn't guarantee quality of data, and didn't support ACID transactions like data warehouse.

4.3 Data Lakehouse

Data Lakehouse is the architectural concept which overcomes the limitations of data warehouse and data lake and provides a unified data solution. It brings together the ACID compliance, data quality of data warehouse and polyglot data format, low cost storage, interoperability via open file format of data lake. Open table format works as the lynchpin to accomplish this in the lakehouse pattern.

Data lakehouse is an open data architecture - combining open file format (Parquet, ORC, Avro) and open table format (Iceberg, Delta Lake, Hudi) with underlying storage in public cloud (AWS S3, Azure Blob Storage). A processing engine like Snowflake, Databricks, Spark, Trino uses the data through the open table format to provide the lakehouse capability. As an example, Apache Spark doesn't provide ACID property on its own but it achieves this using open table format [20]. Data lakehouse is not a single software component - a combination of open file format, open table format and a processing engine together forms a lakehouse. Apart from addressing the limitations of data warehouse and data lake, data lakehouse provides its own benefits. Leveraging the capabilities of open table format, lakehouse supports data versioning, time travel enabling historical, point in time reporting for audit, compliance purposes. The open nature of underlying table format and file format eliminates vendor locking, making it possible for multiple processing engines to leverage the same data [10]. Needless to say, single copy of data files and table format reduces storage costs and avoids data copy, thus allowing organizations to maintain a single source of truth by implementing a lakehouse pattern.

5. Iceberg In Depth

Iceberg is one of the most popular table formats. It brings SQL behavior into the big data world [4]. In the decade of 2010, as big data technologies grew in adoption, the data access pattern from big data platforms deviated away from traditional SQL flavor. Iceberg addresses that and makes it possible to leverage plain old simple SQL in big data. As a table format, it provides abstraction to end users and the end user doesn't need to know how it works under the hood. As a technology, it also provides a set of APIs and libraries to interact with the platform. As a table format, it allows multiple query engines and data platforms such as Spark, Trino, Impala and Snowflake work with the same table simultaneously [21].

5.1 History

Apache Hive is considered a first generation table format. Video streaming giant Netflix was using Hive and ran into issues with the Hive file format such as performance issues due to partitioning strategy, file rename, lack of atomic operations [22]. Netflix came up with Iceberg table format in 2017 to address the problem. Later in 2018 Netflix open sourced it to Apache Software Foundation. The project came out from incubator status in 2020 [23]. Since then, the software community is maintaining this project under Apache.

5.2 Architecture

Iceberg table format is a layered metadata architecture [32, 33]. It contains 3 layers [Figure 1] in the overall solution - a catalog layer, a metadata layer and finally the data layer which are the open file format Parquet, ORC, Avro files. The middle metadata layer is further broken down into 3 layers of files - metadata file, manifest-list file and manifest file.

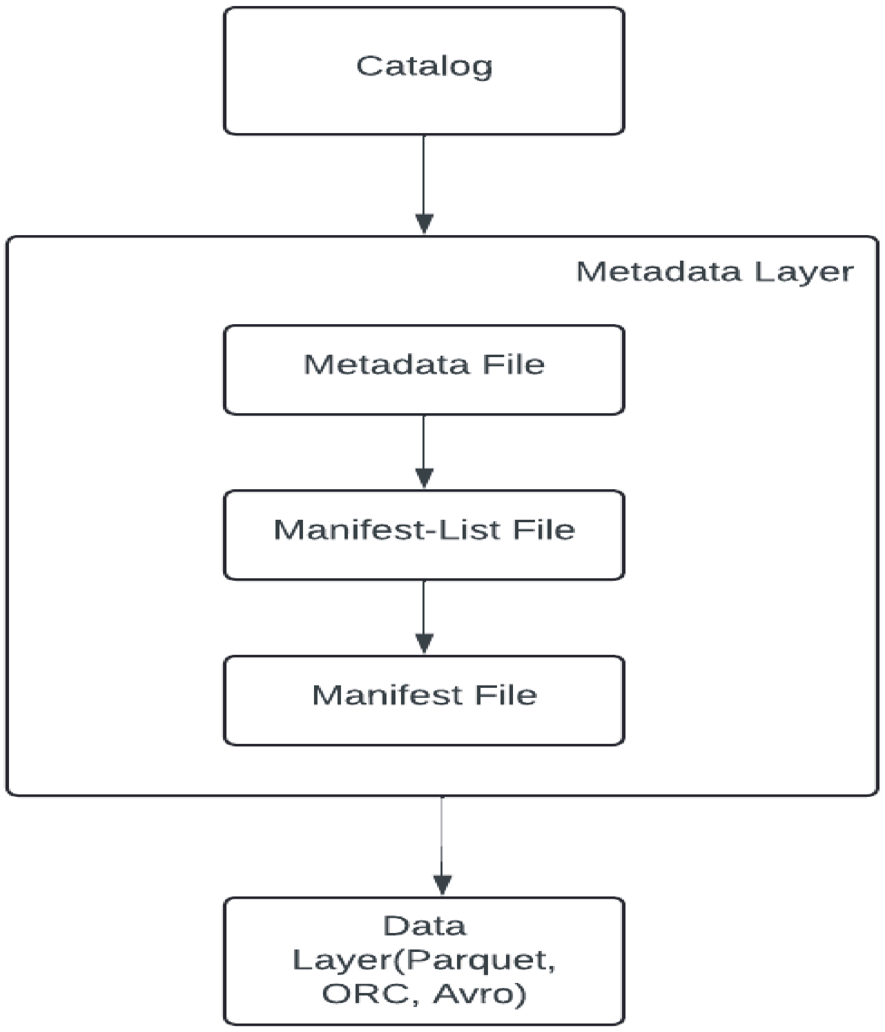

Figure-1 - Metadata architecture of Iceberg

In Iceberg table format architecture, the topmost layer is called Catalog. Catalog layer contains information about the latest metadata location of Iceberg tables. Since it is the source of truth for the latest data of a table, the metadata pointer change operation in the Iceberg catalog for an Iceberg table must support atomic operation. In a single catalog file, the metadata pointer for all tables of Iceberg exists.

Several technologies are available to build the Iceberg catalog. To start with, Apache Hive metastore can work as an Iceberg catalog and it is one of the oldest and most common catalog technologies. Nessie [35] is another option which provides a transactional metadata management system. Nessie is relatively new, but it supports version control like Git and is a cloud agnostic technology that makes it an interesting choice. There are a bunch of other options in the AWS ecosystem as well such as AWS Glue, DynamoDB etc, but while these choices come with the support of AWS, implementation with any of the AWS technology causes vendor lock-in which makes it a less favorable choice for catalog. Catalogs can be implemented in relational databases as well, using JDBC as well as REST APIs. In June of 2024, Snowflake brought in a new catalog format named Polaris [34]. It is currently in public preview and Snowflake plans to open source the technology which can be deployed within Snowflake platform or in an organization's own infrastructure.

The next layer below the Catalog layer is the Metadata layer which is broken into 3 sub-layers. These sub-layers have specific file format requirements - the topmost sub-layer is the metadata file which is in JSON format. The next two sub-layers in the hierarchy, manifest-list file and manifest file are in Avro format.

Iceberg architecture has a concept called snapshot. Snapshots represent a consistent, point-in-time view of a table. Each snapshot captures the complete state of the table, including all its data files and metadata, at a particular point in time [Figure 2]. A metadata file might contain one or multiple such snapshots. So, while one catalog file maps to a single metadata pointer for a table, that metadata pointer might map to many snapshots in the metadata file. Each such snapshot maps to a single manifest-list file which contains a list of manifest files, thus making the manifest-list to manifest file mapping, a one-to-many relationship. The manifest file is the bottommost sublayer of metadata and contains the list of data files where the actual data for the partition resides.In the metadata section of a manifest-list file, apart from the mapping of which snapshot a manifest file belongs to and partition information, the upper, lower boundary value of the partition columns are also maintained. Therefore a query when executed based on the lower, upper bound value of columns in the partition, Iceberg filters out the partitions which don't belong to the search criteria - thus narrowing down the scope of the search, resulting in faster query response. Figure 2, demonstrates the full metadata architecture along with all of the one-to-one and one-to-many relationships between the layers of Iceberg table format.

The data files are in Parquet, ORC or Avro format, Parquet being the most popular. So while a manifest file contains mapping of data files underneath it for data of a partition, having multiple data file means, the partition data is further subdivided into sub-partitions where each data file contains sub-partition data. This hierarchical structure of Iceberg metadata allows to narrow down the scope where the search query will run, and also facilitate parallel search across data files causing search query response time in single digit seconds in a dataset of size in petabytes. This metadata architecture apparently seems complex but significant benefit in query performance and interoperability outweighs the complexity of the design.

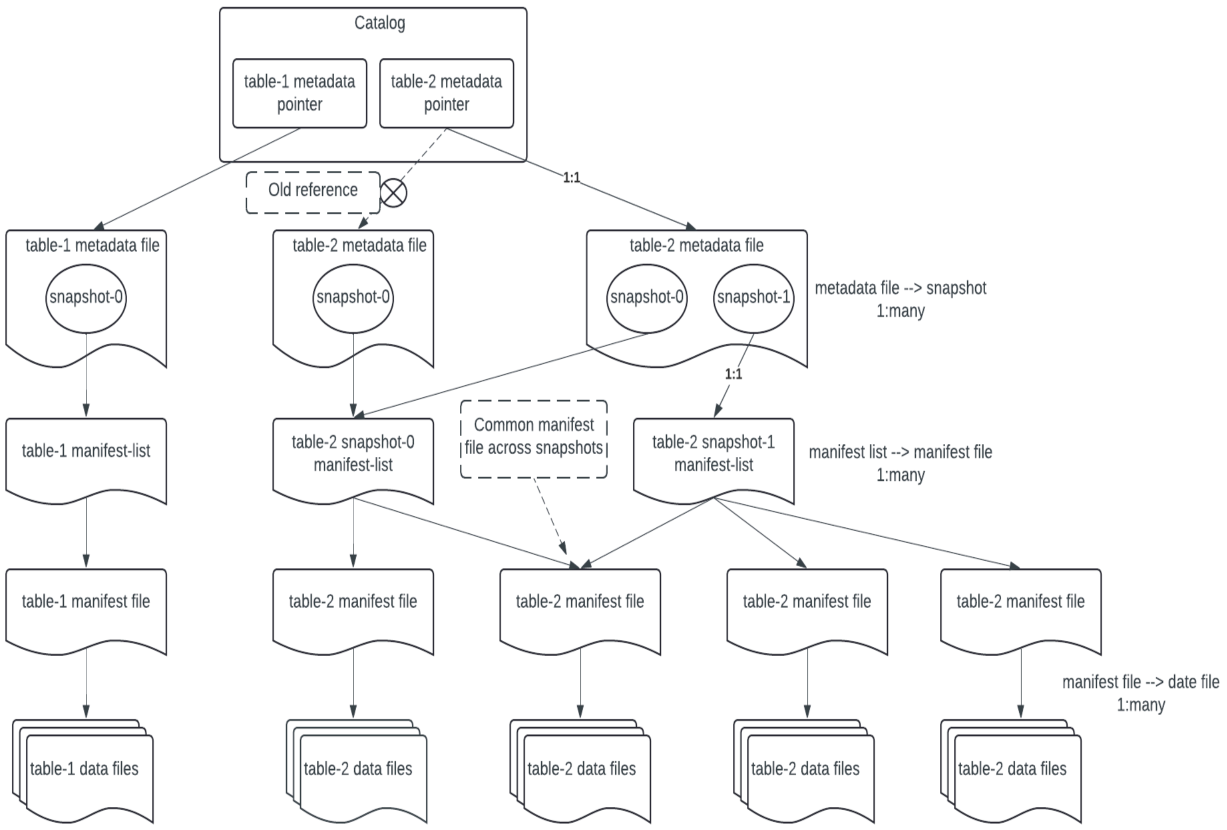

Figure-2 - Detailed metadata architecture with one-to-one and one-to-many relationships between layers

For write operations, Iceberg supports both copy-on-write (COW) as well as merge-on-read (MOR) [36]. For COW mode, Iceberg creates a new copy of the existing data file. The new data file contains added, updated records and omits deleted records. Manifest file updates its datafile list to point to the new data file. The old data file gets deleted asynchronously at a later point in time. Because COW performs this data file copy operation, write is slower compared to MOR since MOR doesn't perform a data file copy during write operation. If write operations impact one or handful number of rows for most of the use cases, then the COW approach is inefficient as too many files will be created from existing data files when the actual change impacts only a handful number of records in a file. Apart from increasing use of storage space, COW approach also has a latency impact for a system that is write heavy. If the number of record updates are usually large and volume of write operations are small or moderate, then COW is a good choice for writing in Iceberg. On the other hand, MOR creates a new file during write which contains records which are updated or deleted. It doesn't create any copy of the existing data file. As a result of that, write operation with MOR is faster than COW. However during read operation, the original data file is updated making read slower for MOR compared to COW. Iceberg implements MOR using a "delete file" technique. In this technique, a delete file is created with records to be updated or deleted. During read operation, the delete file together with the original data file are used for reconciliation to update the existing data file. As part of the compaction process, whether it is COW or MOR, older data files as well as "delete files" are cleaned up asynchronously to reduce storage space as well as maintenance overhead. While COW is a better choice for large but infrequent writes, MOR is just the opposite - it is better for small and frequent writes due to low latency writes during update/delete. If the system is ready heavy then MOR is not a good choice compared to COW as read operation takes longer in MOR due to reconciliation.

5.3 Capabilities

Iceberg as a table format provides several capabilities that's causing increased adoption of this technology at a rapid pace in the industry:

- **ACID compliance** - data manipulations are atomic providing transaction capability
- **Hidden partitioning** - Hive requires manual partition column definitions which is inefficient and complex. Iceberg addresses this limitation by providing a hidden partitioning concept. Iceberg manages partitioning internally by using data structure and metadata and users remain completely unaware of the partitioning scheme [24].
- **Partition Evolution** - if performance degrades over time, Iceberg changes the partitioning scheme on its own enabling partition evolution.
- **Schema evolution** - Schema evolution is straightforward in Iceberg as only metadata fields are updated. Standard operations such adding a column, dropping an existing column or renaming it are all possible in Iceberg using metadata file manipulation. The data files remain unchanged [26].
- **Time Travel** - Change in Iceberg causes creating a new version of metadata called snapshot. Old snapshot remains in the system for a while. This allows users to time travel over data using date range or version number of a snapshot [27].
- **Concurrency** - Iceberg allows concurrent reads and writes by multiple engines at the same time leveraging optimistic concurrency control[10, 28]. When there are multiple concurrent requests, Iceberg checks for conflicts at the file level, allowing multiple updates in a partition as long as there are no conflicts.
- **File filtering** - The metadata files contain min, max values for a column.This allows query that searches over petabytes of data, come back with result in single digit seconds producing significant fast performance.
- **Table Migration** - Iceberg provides a mechanism to create Iceberg metadata files using metadata files of other table formats such as Delta Lake using its table migration feature. It comes in 2 flavors - full data migration and in-place data migration. The full data migration creates a copy of the data files along with creating the Iceberg metadata files. The in-place migration on the other hand reuses the data files and only creates new Iceberg metadata files [29].

5.4 Performance

Iceberg has effectively addressed the performance limitations that traditional data lakes often encounter. The capabilities outlined in the previous sections form the foundation of how Iceberg delivers exceptional performance.

- Read query performance
  - Metadata & Partitioning: Search is the most essential use case in database management systems. Iceberg supports low latency search of data in tables of size in petabytes. It achieves that through its hidden partitioning and file filtering techniques controlled by its metadata architecture. The lower and upper bound of column values of a partition in the metadata files allow queries to skip files and narrow down search scope to return fast response.
  - File compaction - Data fragmentation and increasing number of data files are two key factors which if not managed efficiently and will slow down execution of query over time in a database management system. Iceberg periodically runs a compaction job to merge small files into large ones or by merging delete files with data files [51]. This helps in maintaining an optimal storage structure which in turn helps in fast query execution.
- Write Query Performance - Iceberg provides flexibility to implement two types of write strategy - copy-on-write(COW) and merge-on-read(MOR) [36]. Depending on the nature of the use cases - ready heavy system vs write heavy system, implementation can choose appropriate strategy to improve write query execution time by adopting MOR or compromise in write performance by adopting COW for read heavy systems.
- High throughput - Iceberg uses optimistic concurrency control(OCC) strategy and allows concurrent read/writes. Allowing concurrency improves the throughput of the overall system which enables not only execution of multiple requests from a single query engine but also execution of multiple requests from multiple query engines on the same dataset at the same time.
- Compression of data - Iceberg supports columnar file format like Parquet, ORC which inherently supports advanced compression. This allows data to be stored more efficiently in the underlying storage benefiting query performance when combined with the metadata architecture of Iceberg.

Iceberg doesn’t support native caching strategies like Delta Cache of Delta Lake [50]. Implementing a similar caching strategy will further boost ready query performance and is an opportunity for future research on performance optimization.

5.5 Implementation in Big Data Platforms

Iceberg is supported through most of the common query engines such as Spark, Trino, Presto and data platforms such as Snowflake, Dremio lakehouse. Due to its flexible table format, high performance, ACID compliance and overall robust architecture, the adoption of this technology has been attractive to these platforms. All three query engines, Spark, Trino and Presto have native integration with Iceberg. Dremio uses Iceberg to implement a data lakehouse model and takes advantage of its time travel capability [30]. Snowflake, the modern cloud data platform, allows Iceberg connectivity as external volume through its platform. Initially Snowflake supported 2 types of Iceberg implementation - native table and external tables, it has since then unified the two approaches and now offer a common solution of Iceberg through configuration [31]. However for Iceberg integration with Snowflake, the data files must be in Parquet file format even though Iceberg in general as a technology, supports other file formats such as ORC and Avro.

6. Iceberg & Other Table Formats - Comparative Analysis

Delta Lake and Apache Hudi are the other popular table formats apart from Iceberg. Apache Paimon, is another table format which is fairly new and more suitable for batch and stream operations [37]. In this section, we will briefly discuss the architecture and compare the architecture and capabilities between Iceberg, Delta Lake and Hudi [Table-1].

6.1 Delta Lake Architecture

Delta Lake also implements a metadata type of solution like Iceberg albeit in a different way. In Delta Lake architecture there are three types of files - transaction logs which are in JSON format, checkpoints using Parquet format and the underlying data table which also uses Parquet format [38]. The table itself is a directory which contains the data files, log transactions as well as the checkpoint files. The transaction log determines which data files are of which version of a table. Transaction log records contain something called “actions” - the most notable one is “Change Metadata” action, which like Iceberg metadata file contains schema and partition information. Log records are periodically compressed into checkpoints. A checkpoint file stores a compact, point-in-time snapshot of the table's metadata, reducing the need to replay the entire log from scratch. In the event of a failure or restart, Delta Lake can recover the latest consistent state of the table using the checkpoint files.

6.2 Hudi Architecture

Apache Hudi’s [39] metadata architecture revolves around timestamps. Metadata is implemented using more than one file type - partition metadata and timeline metadata. A single Hudi table uses several directories to organize metadata. Metadata contains many indexes for performance optimization. The actual data files are of two types - base file that contain the actual data and log files which keep track of changes in base file. The base files use Parquet or ORC format and the log files use Avro format.

6.3 Architecture & Capability Comparison

Iceberg	Delta Lake	Hudi
Supports Parquet, ORC and Avro data file format	Supports only Parquet format	Supports Parquet, ORC and Avro format
Metadata implementation hierarchical in nature - catalog, metadata layer(metadata file,	Metadata implementation tabular in nature - transaction log file and checkpoint file	Metadata implementation tabular in nature - partition metadata and timeline metadata

manifest-list and manifest file)		
No caching support for performance optimization	Delta Cache support for performance optimization	No caching support for performance optimization
Supports Copy-On-Write(COW) and Merge-On-Read(MOR) for write operations	Supports only Copy-On-Write(COW)	Supports both Copy-On-Write(COW) and Merge-On-Read(MOR)
Hidden partitioning and partition evolution supported	Explicit partitioning required, partition pruning supported	Explicit partitioning required and partition evolution supported
Snapshot versions are used for time travel	Transaction log based versioning for time travel	Time travel based on incremental commits and versions
Schema evolution supported including addition and rename of fields	Supports schema evolution without requiring schema migration	Schema evolution supported with automatic handling of schema migration
Concurrency writes are supported using Optimistic Concurrency Control and snapshots	Transaction log based Optimistic Concurrency Control	Optimistic concurrency control using multi-version concurrency control

Table-1

6.4 Interoperability - Metadata Conversion

Even though the table formats help building interoperable solutions and avoid data copies, the challenge with interoperability still remains to some degree if certain query engines are compatible with specific table formats only. Snowflake, the modern data cloud platform works with only Iceberg table format. Therefore even if data files are in Parquet but metadata formats are in Delta Lake or Hudi, Snowflake will not be able to process the data. In order to solve the problem, table format technologies are also implementing solutions to generate metadata in multiple formats to truly achieve interoperability.
Iceberg can generate metadata format from other table formats such as Delta Lake or Hudi into Iceberg format [41]. There are two different techniques Iceberg supports - full data migration and in-place data migration. Full data migration not only creates Iceberg metadata but also creates a copy of the data files as well. On the other hand, in-place data migration creates only metadata and the original data files are used. In-place is preferable as the purpose of open table format and open file formats are to not duplicate data and allow access to the same data through multiple engines. Delta Lake supports a similar process using Delta Uniform [42]. Unlike Iceberg table migration which creates Iceberg compatible metadata from other table formats, Delta Uniform does the opposite . Delta Uniform creates Iceberg or Hudi compatible metadata from

the original Delta Lake table format. Hudi on the other hand doesn't support creating other table formats.
Apache XTable is a new open source technology that supports cross table format interoperability [40]. While Iceberg and Delta Lake support only one way conversion of metadata (Delta Lake/Hudi to Iceberg by Iceberg, Delta Lake to Iceberg/Hudi by Delta Uniform), XTable creates metadata from any format to any format. As an example, if the existing table format is in Iceberg, XTable can generate Delta Lake and Hudi metadata. Likewise if the existing table format is in Delta Lake, XTable can generate Iceberg and Hudi metadata. XTable also supports the emerging table format Apache Paimon [37].

6.5 Table Format Ecosystem

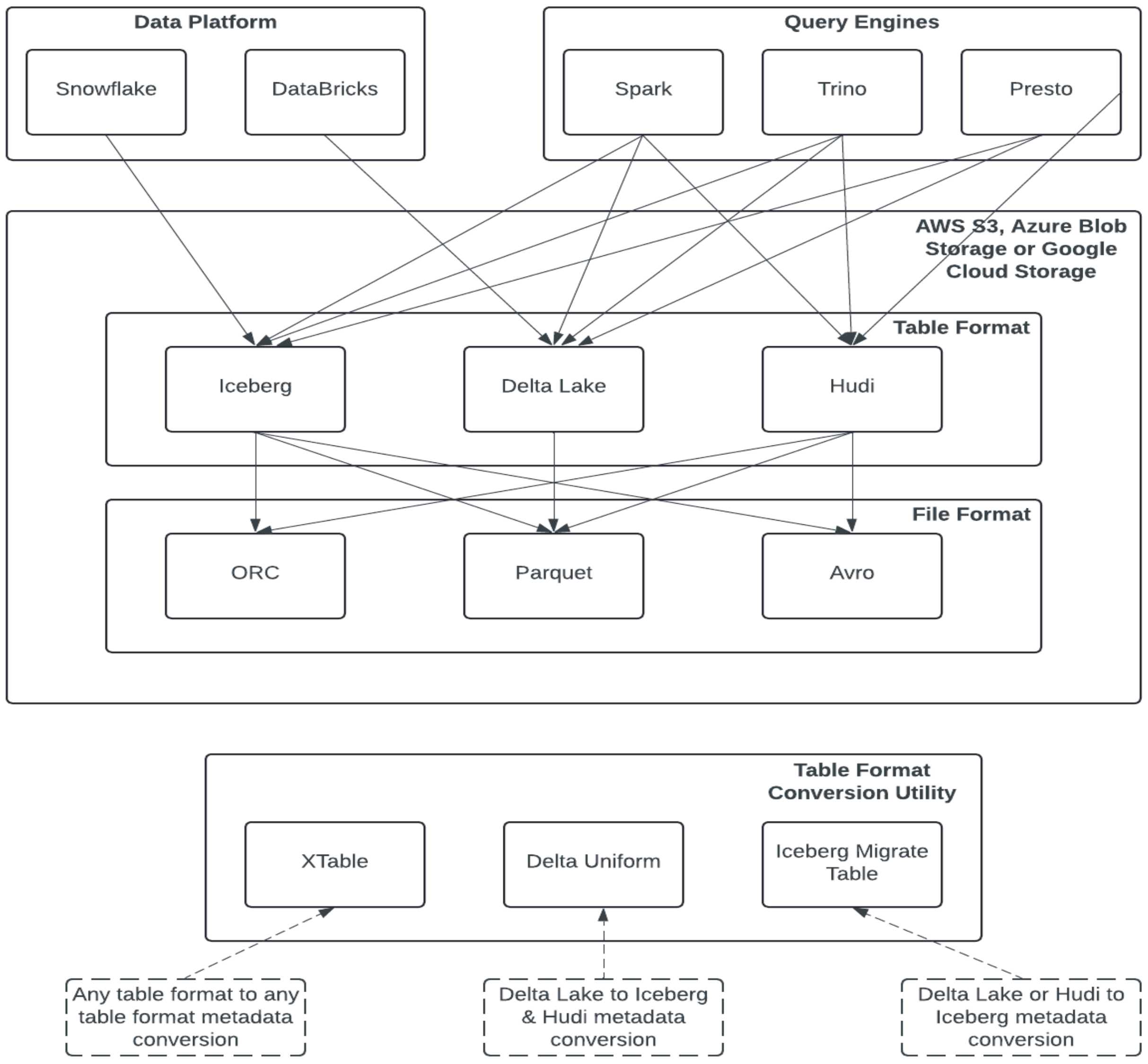
Data Platform
Snowflake
DataBricks
Query Engines
Spark
Trino
Presto
AWS S3, Azure Blob Storage or Google Cloud Storage
Table Format
Iceberg
Delta Lake
Hudi
File Format
ORC
Parquet
Avro
Table Format Conversion Utility
XTable
Delta Uniform
Iceberg Migrate Table
Any table format to any table format metadata conversion
Delta Lake to Iceberg & Hudi metadata conversion
Delta Lake or Hudi to Iceberg metadata conversion

Figure-3: The overall landscape of table format and it's adoption by various data platforms and query engines

All three table formats have wide industry adoption - while Iceberg was founded by Netflix and adopted by Adobe, Apple, Hudi is used at companies like Uber, Walmart, Robinhood. Delta Lake on the other hand is used by prominent organizations such as Instacart and Comcast. Due to its flexible catalog approach, integration of Iceberg in this data space seems to be making more progress than Delta Lake and Hudi [20].

6.6 Iceberg Specialities

Iceberg supports Git-like capabilities such as branching, tagging through integration with Project Nessie [35, 43]. Nessie in the big data world is synonymous to Git in source code repositories [35]. Using the Nessie extension in Iceberg, the catalog table of Iceberg can be simultaneously updated by multiple users across different branches and later commit the changes from the individual branches to the main branch. This is a technique to accomplish multi-table transactions in Iceberg which is not natively supported otherwise. Because Nessie works like the Git version control system, it allows listing commit history and even cherry picking commits across branches if needed [43]. This capability is not available in other table formats at this moment.

7. Disruption & Adoption

Iceberg is a disruptive technology which is being broadly adopted by top organizations and having tangible impacts across many sectors- finance, retail, healthcare and entertainment and even within the software industry itself [Table-2]. It is an already matured technology that is solving decade old complex problems of the big data world, in a stable and reliable manner.

- Data Integrity and version control - this technology brings versioning and atomic operations in big data. Together with its time travel capability, organizations can access historical, point in time views of data without the risk of data corruption which was not achievable in data lake. This particularly benefits industries like finance to support compliance, audit and reporting.
- Performance in large dataset - Organizations are generating and ingesting massive volumes of data at a rapid pace which often requires near real-time data analysis at low latency. This was difficult in traditional data lake. Iceberg as a transformative technology makes this possible through its dynamic partitioning and file filterng strategy.
- Implement complex data strategy with ease - Iceberg as a technology supports schema evolution - forward and backward compatible schema management, quite similar to Kafka [44]. This allows organizations to modify their data model due to evolving business needs, avoiding breakage in their data architecture. In the retail sector, where due to the variety and veracity of data, data models evolve quickly, schema evolution of Iceberg allows such changes to happen without disruption in the data pipeline of the organization.
- Cost efficiency - the interoperability of the table format combined with adaptability across multiple data engines enables organizations to avoid data copies and vendor lock in. This reduces storage costs as well as relieves organizations from being forced to choose more than one data engine for their use cases, thus reducing cost on technology investments.

- Future-proof data architecture - Historically organizations had to overhaul their data strategy with the emergence of new technologies which is costly as well as time consuming. Iceberg's open table format architecture allows adaptability across existing technologies as well as emerging technologies in the data field.

Adoption	Use Cases
Adobe	Data integrity, version control and scalability of data platform [46]
Netflix	Scalability and performance for streaming [45]
Apple	Time travel for ML use cases, ACID for GDPR, improvement in batch reliability using Iceberg with Spark [1, 47]
Shopify	Scaling an interoperability of data across multiple engines in the organization [48]
Pinterest	Cost reduction of infrastructure by cutting down cloud compute resources for recommendation, content delivery use cases [49]
Snowflake	Allow customer leverage their external storage having Iceberg data and provide rich capabilities and governance of Snowflake [17]

Table-2: Adoption of Iceberg across organizations of different sectors

In summary, Iceberg addresses key data management challenges by providing fast query performance, establishing single source-of-truth by avoiding data copies, reduction of storage cost and maintenance overhead. By providing ACID, concurrency, interoperability, and performance for polyglot data types, this technology opens up data democratization and solves key issues of legacy data systems, thus establishing itself as a truly disruptive technology.

8. Iceberg Limitations & Future Scope

There are a few limitations of Apache Iceberg which gives an opportunity for future research and improvement.

1. Iceberg table migration only allows creating Iceberg metadata from other formats. It doesn't allow creating metadata of other table formats from Iceberg metadata. Adding this capability in Iceberg to seamlessly create Delta Lake or Hudi metadata through its library or APIs would be a welcome addition towards achieving interoperability.
2. Iceberg, like other table formats, is suited for use in cloud platforms. We should have Iceberg available for use in on-premise infrastructure as well since many organizations

have legal, compliance and regulatory restrictions (like finance and healthcare) to store data within the organization datacenter. This restricts organizations who host data lake in their infrastructure from being able to take advantage of Iceberg open table format.

3. While the table format in general is interoperable and engine agnostic, the metadata layer requires files to be in specific formats - catalog file should be JSON and metadata, manifest-list and manifest-file should be in Avro. While this is not a major problem but certainly can be considered for future research opportunities to make the metadata files format agnostic.
4. Metadata architecture simplification - while Iceberg shines with it's hierarchical metadata layout, the overall metadata architecture though comprehensive, is complicated. In the event when there are many small files, the metadata size can grow significantly causing slow query performance for large datasets.
5. Unlike other table formats like Delta Lake which supports query performance improvement via Delta Cache, Iceberg doesn't have such a feature. A future enhancement to add caching ability to improve query performance is desirable.

9. Conclusion

Apache Iceberg is an open table format gaining fast popularity across big data technologies and organizations. In this paper we deeply reviewed what Iceberg is, why it is important and how it works with open file formats. We explained the importance of interoperability where table formats like Iceberg play a key role in reducing storage cost, avoiding data copies to help establish single-source-of-truth and thus reducing data management complexities in the organization. We reviewed in detail about its architecture and how the architecture helps in implementing its unique capabilities. We also highlighted how open table formats like Iceberg are rapidly gaining ground in the space of big data, and lakehouse. We also reviewed other similar technologies like Iceberg in the form of Delta Lake and Hudi and made a comparative analysis with Iceberg in terms of architecture and capabilities. We reviewed Iceberg's adoption across various data platforms and query engines. We ended with discussing its current limitations and future improvement opportunities which may broaden its adoption across industry and developer communities. This paper highlighted not only the technical advancements brought by Iceberg in the data world but also established why it should be recognized as a disruptive technology having profound impacts by solving today's pressing problems of big data management.

10. References:

1. Apache Spark https://spark.apache.org/
2. Apache Flink https://flink.apache.org/
3. Apache Hive https://hive.apache.org/
4. Apache Iceberg https://iceberg.apache.org
5. Apache ORC https://orc.apache.org
6. Apache Arrow https://arrow.apache.org
7. Apache Parquet https://parquet.apache.org

8. Apache Avro https://avro.apache.org
9. An Empirical Evolution of Columnar Storage Formats - Xinyu Zeng, Yulong Hui, Jiahong Shen, Andrew Pavlo, Wes McKinney, Huanchen Zhang https://www.vldb.org/pvldb/vol17/p148-zeng.pdf
10. Dipankar Majuzmar Jason Hughes JB Onofre https://arxiv.org/pdf/2310.08697
11. Apache Kafka https://kafka.apache.org
12. Study on ORC & Parquet https://onlinelibrary.wiley.com/doi/full/10.1002/cpe.5523
13. File formats for Big Data Storage Systems - Samiya Khan, Jamia Millia Islamia Mansaf Alam https://www.ijeat.org/portfolio-item/a1196109119/
14. https://www.vldb.org/pvldb/vol16/p3044-liu.pdf
15. Dremio File Format https://www.dremio.com/wiki/file-format/
16. Teradata https://www.teradata.com/insights/data-platform/what-are-open-table-formats
17. Snowflake https://www.snowflake.com/en/blog/5-reasons-apache-iceberg/
18. Kyle Weller Hudi vs Delta Lake vs Iceberg https://www.onehouse.ai/blog/apache-hudi-vs-delta-lake-vs-apache-iceberg-lakehouse-feature-comparison
19. Storage Structure in The Era of Big Data: From Data Warehouse to Lakehouse - Mohssine Bentaib, Abdelaaziz Ettaoufik, Abderrahim Tragha, Mohamed Azzouazi https://www.jatit.org/volumes/Vol102No6/16Vol102No6.pdf
20. The Lakehouse: State of the Art on Concepts and Technologies - Jan Schneider, Christoph Groger, Arnold Lutsch, Holger Schwarz, Bernhard Mitschang https://link.springer.com/article/10.1007/s42979-024-02737-0
21. Dell Technologies https://infohub.delltechnologies.com/en-us/l/white-paper-modern-data-stack-with-red-hat-openshift-container-platform-on-intel/apache-iceberg-4/
22. Netflix Iceberg Github https://github.com/Netflix/iceberg
23. Apache Iceberg: A Different Table Design for Big Data - Susan Hall https://thenewstack.io/apache-iceberg-a-different-table-design-for-big-data/
24. Iceberg hidden partitioning https://iceberg.apache.org/docs/latest/partioning
25. Apache Impala with Iceberg Tables https://impala.apache.org/docs/build/html/topics/impala_iceberg.html
26. AWS - Evolve Iceberg Table Schema https://docs.aws.amazon.com/athena/latest/ug/querying-iceberg-evolving-table-schema.html
27. Tabular.io - Time Travel Queries Iceberg https://tabular.io/apache-iceberg-cookbook/basics-time-travel-queries/
28. Optimistic Concurrency Control https://blogs.oracle.com/maa/post/from-chaos-to-order-the-importance-of-concurrency-control-within-the-database-2-of-6
29. Apache Iceberg Table Migration https://iceberg.apache.org/docs/1.3.0/table-migration/

30. Dremio Iceberg - https://docs.dremio.com/current/sonar/query-manage/data-formats/apache-iceberg/
31. Snowflake - Unifying Iceberg Tables https://www.snowflake.com/blog/unifying-iceberg-tables
32. Apache Iceberg - An Architectural Look Under The Covers - Jason Hughes https://www.dremio.com/resources/guides/apache-iceberg-an-architectural-look-under-the-covers/
33. Apache Iceberg - The Definitive Guide - Timer Shiran, Jason Hughes, Alex Merced https://learning.oreilly.com/library/view/apache-iceberg-the/9781098148614/ch02.html
34. Iceberg Polaris Catalog - https://www.snowflake.com/en/blog/introducing-polaris-catalog/
35. Nessie https://projectnessie.org/
36. Row Level Changes on the Lakehouse: Copy on Write vs Merge on Read in Apache Iceberg - Alex Merced https://www.dremio.com/blog/row-level-changes-on-the-lakehouse-copy-on-write-vs-merge-on-read-in-apache-iceberg/
37. Apache Paimon https://paimon.apache.org/
38. Delta Lake: High Performance ACID Table Storage over Cloud Object Store - https://www.vldb.org/pvldb/vol13/p3411-armbrust.pdf
39. Apache Hudi https://hudi.apache.org/
40. Apache XTable https://xtable.apache.org/
41. Iceberg Table Migration https://iceberg.apache.org/docs/1.4.0/table-migration/
42. Delta UniForm: a universal format for lakehouse interoperability - Bilal Obeidat, Sirui Sun, Adam Wasserman, Susan Pierce, Fred Liu, Ryan Johnson, Himanshu Raja https://www.databricks.com/blog/delta-uniform-universal-format-lakehouse-interoperability
43. Iceberg Nessie https://iceberg.apache.org/docs/1.5.1/nessie/
44. Confluent Apache Kafka Schema Evolution - https://docs.confluent.io/platform/current/schema-registry/fundamentals/schema-evolution.html
45. Incremental Processing using Netflix Maestro and Apache Iceberg - Jun He, Yingyi Zhang, Pawan Dixit - https://netflixtechblog.com/incremental-processing-using-netflix-maestro-and-apache-iceberg-b8ba072ddeeb
46. Iceberg at Adobe - Jaemi Bremner - https://blog.developer.adobe.com/iceberg-at-adobe-88cf1950e866
47. Iceberg at Apple - https://www.youtube.com/watch?v=r7KJf8F585Q
48. Rewriting History: Migrating petabytes of data to Apache Iceberg using Trino - Marc Laforet, Cole Bowden - https://trino.io/blog/2022/12/09/trino-summit-2022-shopify-recap.html

49. Scaling row level data deletions at Pinterest - https://www.youtube.com/watch?v=OpXOtIrIuM4
50. Delta Lake Delta Cache - https://docs.databricks.com/en/optimizations/disk-cache.html
51. Maintaining tables by using compassion - https://docs.aws.amazon.com/prescriptive-guidance/latest/apache-iceberg-on-aws/best-practices-compaction.html