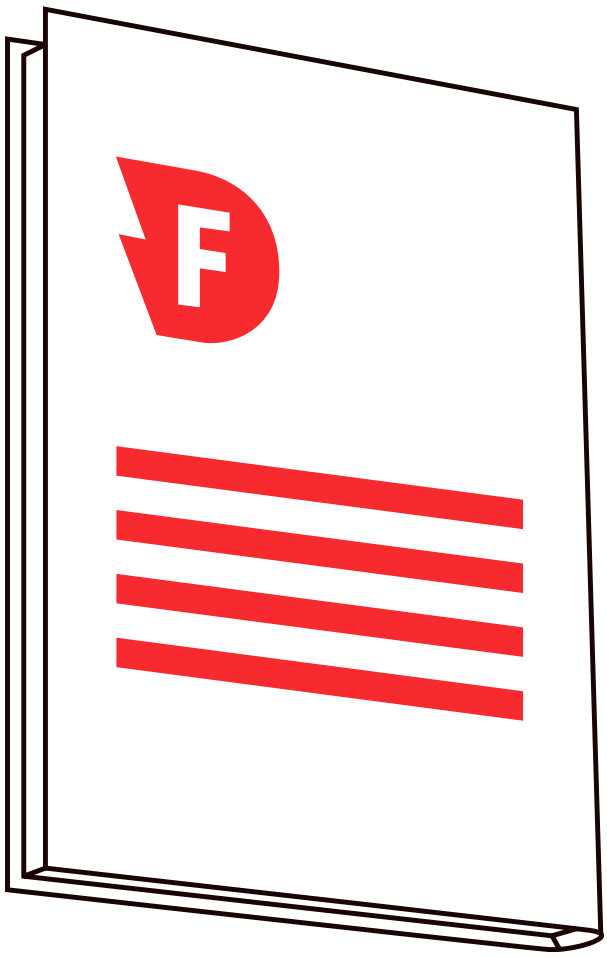

The Firebolt Cloud Data Warehouse Whitepaper

FIREBOLT

Abstract

Companies have usually adopted new cloud technologies for two reasons: to get the latest innovations, and to lower costs. It has been nearly a decade since cloud data warehouses were first released. In the case of cloud data warehouses, most companies adopted them to migrate traditional reporting and dashboard-based analytics to the cloud. Cloud data warehouse vendors did innovate technically in ways that helped with this migration. Decoupled storage and compute architectures gave us elastic scalability and helped simplify administration.

But cloud adoption was mostly to help modernize IT. They have not supported the new needs of the business. Operational and customer-facing interactive analytics have become a top priority to help employees make better decisions faster, and provide greater self-service to customers. These newer use cases require 10x or faster performance for interactive and ad hoc queries; unprecedented data, query, and user scale; and a much lower cost of ownership per user.

During the last decade since cloud data warehouses were first released, they have not delivered the order-of-magnitude improvements in speed, scale and costs required for these new use cases. In fact, cloud data warehouses inherited many of the same limitations that prevented premises data warehouses from supporting these new use cases. For example, most cloud data warehouses are still mostly batch-centric. They do not support streaming ingestion at scale or provide low-latency visibility, both of which are critical for many operational and customer-facing use cases. Those that do not decouple storage and compute have limited data, query, and user scalability. Even those that decouple storage and compute, while they improved scalability, have slower query performance than the older specialized data warehouses. Support for semi-structured data, which is now the dominant form of data, is either incomplete, too slow, or inefficient.

Cloud data warehouses have also often raised costs, not lowered them. There are too many stories of companies blowing through their monthly credits, or allotted

budget. The main solution has been to throttle queries. But limiting the use of analytics goes against the main goal of putting analytics and data into the hands of everyone to make faster, better decisions.

The Firebolt cloud data warehouse was designed to deliver the combined improvements in speed, scale, and efficiency needed to cost-effectively deliver fast, interactive analytics for ad hoc, interactive operational and customer-facing analytics; in other words, fast analytics to everyone. This whitepaper explains the overall architecture, the key differences compared to the previous generations of cloud data warehouses, and provides some numbers to show how much of an improvement in speed, scale, cost, and price-performance is possible.

Requirements for the modern cloud data warehouse

Today's cloud data warehouse has to support more than traditional reporting and dashboards, and the analyst teams behind them. It has to support ad hoc and interactive analytics against both batch and streaming data for 100-1000x more users as companies push operational analytics directly to their employees, and offer self-service analytics to their customers. It also needs to support data engineers and the data engineering lifecycle.

Because of this, cloud data warehouses need to provide much more than the elastic scalability and simplicity that first-generation cloud data warehouses introduced. They also have to deliver an order of magnitude improvement in performance, scalability, cost efficiency, and agility to support these newer users and their analytics. Today, the requirements are:

- **100% SQL** - SQL is the de facto language of data, especially for analysts and data engineers. Every task, from ELT to queries of any data, should be achievable in SQL.

- **Sub-seconds query performance** - Ad hoc, interactive analytics by employees, and self-service analytics by customers all require queries that run in a few seconds or less.

- **Gigabyte-petabyte scale** - The newer types of data - from customer interactions, connected devices or newer applications - are massive in size compared to the data from traditional applications and transactions, and are growing much faster. Most companies have terabytes of data; some have petabytes.

- **Elastic scale** - Ingestion and query workloads are less predictable, which makes elastic scale much more important both for efficiency and service level agreements (SLA).

- **Native support for JSON and other semi-structured data** - While some products do support handling JSON, it is not complete, performant, cost-efficient, or easy. Part of the challenge is that the JSON is stored as text, not as a native data type. But data engineers also need to be able to easily manipulate JSON in SQL.

- **Native ELT support with SQL** - For data engineers to get new analytics out in hours or days, they need to be able to extract, load, and transform (ELT) their own data entirely in SQL, without having to wait on a separate team.

- **Streaming analytics support** - Companies need real-time visibility, which means they need to be able to support continuous ingestion and fast query times.

- **High user and query concurrency** - Analytics is now being delivered to much larger groups of employees and end customers than the traditional analyst teams. It can require supporting 100s to 1000s of concurrent users and queries.

- **Workload isolation** - Unlike reporting, which can be done in batch at any time, multiple workloads and users require isolation to help ensure high priority, near real-time SLAs are met, and to protect workloads from each other.

- **Simplicity for data engineers and DataOps** - Data warehouse deployments can no longer be controlled in ways that slow down changes to support new data needs. Cloud data warehouses need to support DataOps in ways that provide more control to data engineers and enable faster data analytics cycle times.

- **Cost efficiency** - Costs need to drop 10x or more to support analytics by 10-100x more users who consume 10x or more data.

The earlier generations of cloud data warehouses do not fulfill all these requirements. Those with decoupled storage and compute are SQL-native and provide elastic scale. They also provide adequate performance for reporting and other repetitive query workloads through caching. But they do not deliver the type of performance or efficiency needed for ad hoc, interactive, operational or customer-facing analytics where caching does not help as much.

This is clear from independent benchmarks like the one by Fivetran. It found the price-performance of Snowflake, Redshift and Google BigQuery to be roughly the same, with average query times for uncached data at 8-11 seconds across queries and $150,000 or greater annual 24×7 costs for a cluster with 1TB of data. Even the most expensive specialized data warehouses, like Teradata, only cost $100,000 a terabyte annually.

The implied costs in these benchmarks are too expensive, especially as companies reach petabytes of data, and too slow. .

Employee and customer-facing analytics, as well as automation, requires much faster and much more cost-effective analytics. Most people expect data in a few seconds or less when they need to make near real-time decisions. Even 50% of people expect mobile applications to return in 2 seconds or less.

Satisfying all of these requirements requires a redesign of the modern decoupled

storage-compute architecture to address three major performance and efficiency limitations.

- **Data access** - Most cloud data warehouses fetch entire segments or partitions of data over the network despite the network being the biggest bottleneck. In AWS, for example, the 10, 25 or 100 Gbps (gigabits per second) networks transport roughly 1, 2.5 or 10 gigabytes (GB) per second at most. When working with terabytes of data, data access takes seconds or more. Fetching exact data ranges instead of larger segments can cut access times 10x or more.

- **Query execution** - Query optimization makes a huge difference in performance. It is one of the reasons Amazon chose to use ParAccel originally; because building all the optimization takes time. And yet most cloud data warehouses lack a lot of proven optimization techniques - from indexing to cost-based optimization.

- **Compute efficiency** - Decoupled storage and compute architectures have been a blessing and a curse. They allowed vendors to use nearly unlimited scale to improve performance instead of improving efficiency.

Firebolt cloud data warehouse

The Firebolt cloud data warehouse was rearchitected from the ground up to deliver the speed and efficiency at scale needed for ad hoc, interactive operational and customer-facing analytics. Like some of its predecessors, it is built on a decoupled storage and compute architecture. But it adds to the previous generations of cloud data warehouses by optimizing decoupled storage and compute together to improve speed, scale, and efficiency.

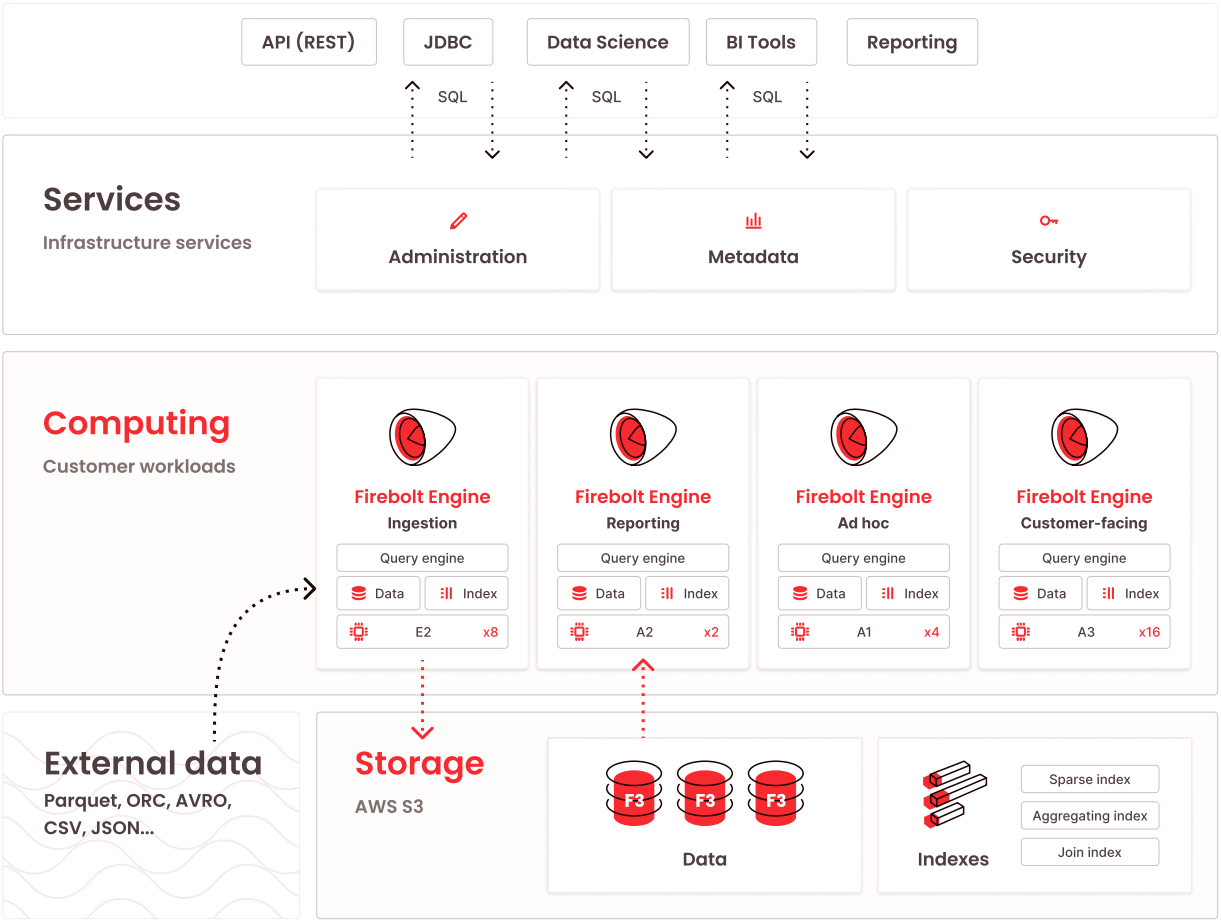

Figure 1. Firebolt decoupled storage-compute architecture

Like other modern cloud data warehouses, Firebolt is a multi-tenant SaaS cloud data warehouse. It supports ANSI SQL at a similar level to other cloud data warehouses via JDBC drivers and REST APIs.

Firebolt is also built on a decoupled storage and compute architecture that enables horizontal, linear data and compute scalability, with the ability to both scale and isolate compute workloads and users. Storage, like others, is built on S3. In the case of Firebolt, the compute clusters are called engines. They are used both for ingestion and all types of analytics. Workload isolation is simple. Whenever you need to support a new isolated workload, you start up a new engine with a few clicks. You can also scale up and down with a few clicks the same way. Once users connect to an engine and send queries, data is fetched and cached locally in each node of the engine.

Firebolt has added features to help with performance and efficiency that other data warehouses do not have. The biggest differences are how decoupled compute and storage are optimized together for performance, scale and efficiency:

- Storage is managed across tiers - While storage is decoupled from compute, all data is managed through a common data access layer as a single logical data set across remote storage and SSD data caches on nodes. While remote storage has segments, only much smaller data ranges are fetched and cached in each node's SSD.

- Indexes accelerate data access and operations - Firebolt uses indexing extensively to improve performance. Each table has a sparse (primary) index consisting of any number of columns in any order. In addition, there are indexes that help with aggregations, string searches, or joins. Indexes dramatically improve performance and overcome some of the limitations of decoupled storage and compute. They are also updated during ingestion, which supports real-time streaming analytics use cases.

- Query optimization is compute- and storage-aware - Firebolt optimizes queries to take advantage of the existing location of data across nodes and drive future data distribution to improve query performance on each engine.

- Distributed, dynamic query execution built for performance and efficiency - The Firebolt query execution engine builds on the storage, indexing, and query

optimization, and then uses LLVM (Low Level Virtual Machine) with pushdown optimization, vectorized processing, and a host of other execution techniques to maximize performance and price-performance.

This optimization of storage and compute with indexing reduces both remote data access and the amount of data cached by 10x or more, which also leads to 10x less data to scan and process. Query optimization and indexing reduces the amount of data and computing needed even more. The combination has helped customers improve query performance 4-6000x while improving price performance 10x or more.

Firebolt Storage

Firebolt's performance and efficiency improvements start with storage, which is built on the Firebolt File Format (F3), pronounced "Triple F". Firebolt stores and manages data differently on each tier - ranging from remote storage, to local SSD data caches in each node. All data is accessed via a common data access layer that manages the entire data lifecycle from ingestion, data storage and indexes, to data access and caching across these tiers.

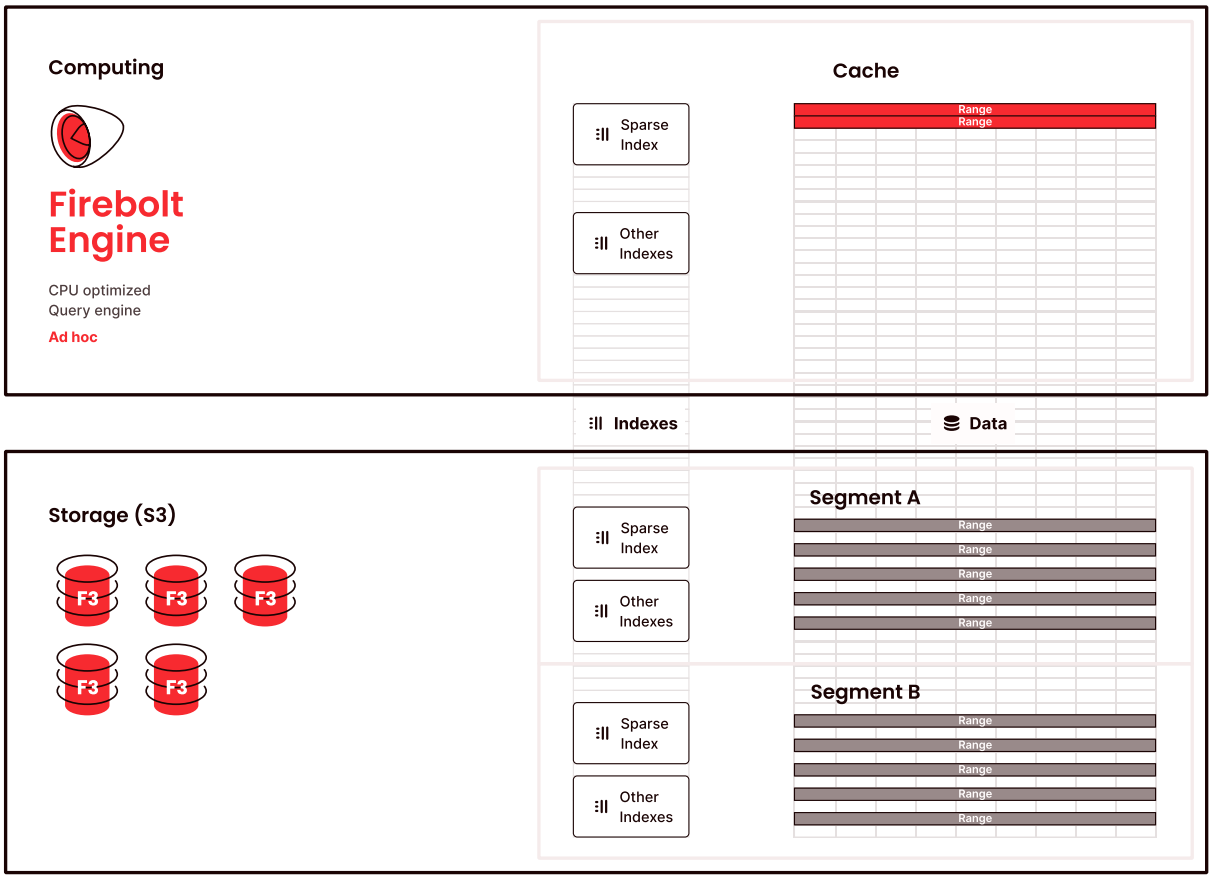

Figure 2. Firebolt File Format (F3) tiered data architecture

The first big challenge with most decoupled storage today is data access latency, or the amount of time it takes to fetch all the data needed by a given query. In this case, the network bandwidth is the main bottleneck. AWS offers 10Gbps-100Gbps network bandwidths, which can transfer roughly 1-10 GB of data per second, at most. Even working with 100GB of data could easily lead to seconds of data access

times for a query. You would need to wait a few minutes for 1TB to load into a cluster.

The biggest way to lower access times is to reduce the amount of data fetched from storage. Most cloud data warehouses and query engines have very inefficient data access because they fetch entire partitions or segments of data from decoupled storage. Many implement partition-level pruning and sorting to reduce the number of partitions fetched. But fetching entire partitions makes no sense if you only need a fraction of the data in each partition. For many queries, especially lookups, you only need 1% or less of the partition.

Firebolt accesses much smaller ranges through the use of indexes. For example, Firebolt only fetches the data ranges it needs, not entire partitions or segments. What makes this possible is sparse indexing. In Firebolt, the unit of remote storage is a segment, and it can be any size (see the Data ingestion section below for more.) Within each segment are much smaller units called ranges. Firebolt only accesses much smaller ranges of data instead of larger segments. Storing only the ranges not only dramatically reduces data access times. It reduces the amount of data scanned, and scan times as well. Indexes can help in many other ways as well.

Indexes

Despite the extensive use of indexes in relational databases (RDBMS), indexes have not been used extensively in most data warehouses. But indexes have been shown to improve both performance and efficiency by 1-3 orders of magnitude when used in columnar databases.

Firebolt built F3 with native support for a broad range of indexes to improve performance, scalability, and price-performance, and new ones are constantly being added. This ever-growing list of indexes includes:

- Sparse (primary) indexes
- Aggregating indexes
- Join indexes
- Metadata indexes
- String indexes

You can think of indexes as improving performance and cost in two ways. A sparse, or primary index for each table can reduce the amount of data fetched, stored, and processed by compute 10x or more. Other indexes improve performance and cost even more by replacing data and compute with cached, precomputed results.

Firebolt automatically updates indexes during each atomic data ingestion operation, which ensures indexes are always consistent with the data.

Sparse indexes

A sparse index dramatically reduces the amount of data fetched, stored, and processed for each query. It is the primary index of a table, composed of any number of columns listed in any order of a table, and is declared with a single line of SQL alongside the table definition.

As data is ingested, Firebolt automatically sorts and compresses data based on the sparse index, and incrementally updates each index. The data and indexes are then committed as a new segment. Administrators do not need to do any maintenance, such as vacuuming or rebalancing, after the initial declaration. Firebolt automatically optimizes and merges segments over time for performance.

Data is always accessed via F3 using the sparse index to find and fetch an exact range of data instead of fetching entire segments, partitions, or blocks, which is what other cloud data warehouses do. Only fetching these much smaller ranges can result in 10x or less data fetched over the network the first time that data is needed. This reduces data access times 10x or more. Sparse indexes also improve query performance and require fewer resources because Firebolt only stores, scans, and processes the much smaller data ranges locally in SSD on the compute nodes, not the entire segments.

Aggregating indexes

Aggregations - ranging from COUNT or SUM to GROUP BY operations - are at the heart of any interactive analytics. But they can require a lot of compute. Some companies use materialized views to help pre-compute and store aggregations, but materialized views can be costly to maintain and limited in function. For example, several materialized views do not support COUNT DISTINCT. You will need to rewrite your query to use the materialized view as well, unless the query optimizer is designed to detect and use them.

Because of these limitations and costs, while many companies do use materialized view capabilities in the warehouse, many others have resorted to calculating their aggregations outside of the data warehouse using Spark or various ETL technologies. It can not only become costly to maintain. It can take weeks to get any changes done since you have to coordinate with a different team that then has to develop, test, and deploy the changes.

Firebolt added aggregating indexes as a way for data engineers to do all this by adding one line of SQL when you create the fact table, or any time after. You can have any number of aggregating indexes associated with a table. Firebolt automatically populates and maintains each aggregating index - which can include raw data, aggregations, and other operators - during ingestion. Whenever the query optimizer sees an operation that can be performed faster using an aggregating index, it automatically adds the index into the query plan. A data engineer does not need to rewrite the query.

Aggregating indexes are implemented as tables stored alongside the fact table, with their own sparse index to make queries faster. In many cases, they can replace fact tables as the main source for a query. For example, they can maintain many levels of granularity for GROUP BY and other aggregation operators automatically, such as COUNT DISTINCT.

Whenever an aggregating index is used, it replaces the need to access, store and compute data with cached, pre-computed results. This improves price-performance by cutting out related data access times, cache and compute resources.

Join indexes

Firebolt has been extensively optimized to minimize the cost of joins, which is one of the most expensive operations in analytics.

A join index helps replace full table scans with lookups and other operators that are much faster, and much less expensive. Because indexes themselves are small, join

indexes are stored in RAM to improve performance. For example, a join index can enable the query optimizer to turn multiple table scans into a single list of lookup operations.

Firebolt can also “pushdown” predicates to happen before joins to reduce the size of the fact table data set before the lookups as well. The query optimizer will find the best combination of predicate pushdowns and indexes based on the location of data across nodes to maximize performance. This includes optimization of nested joins and multiple levels of predicate pushdowns.

The combination of query optimization and join indexes help deliver sub-second query performance for complex, multi-terabyte table joins.

Metadata indexes

A metadata index helps improve query performance for operators or predicates dependent on variables other than the primary index. For example, some metadata stores values like COUNT. Other metadata tracks the MIN, MAX, and other values to help further prune ranges. It can be very effective with columns that have high cardinality. Firebolt uses metadata and other indexes both for cost-based query optimization. It also runs queries whenever possible using just the indexes to avoid accessing data, which helps improve performance.

String indexes

Firebolt has several built-in indexes to support string-based operations. By default, the sparse index supports strings and LIKE operations. But you can use other string indexes as well, such as a hash index to support faster EQUAL or GROUP BY operations.

Query optimization

Firebolt's extensive query optimization is one of the big reasons Firebolt is able to deliver sub-second query times. Each time a new query is received, the Firebolt query optimizer starts to build the best query plan just-in-time. It first looks to see where the data ranges are that it may need for the query across remote storage and the local cache within each node of the engine.

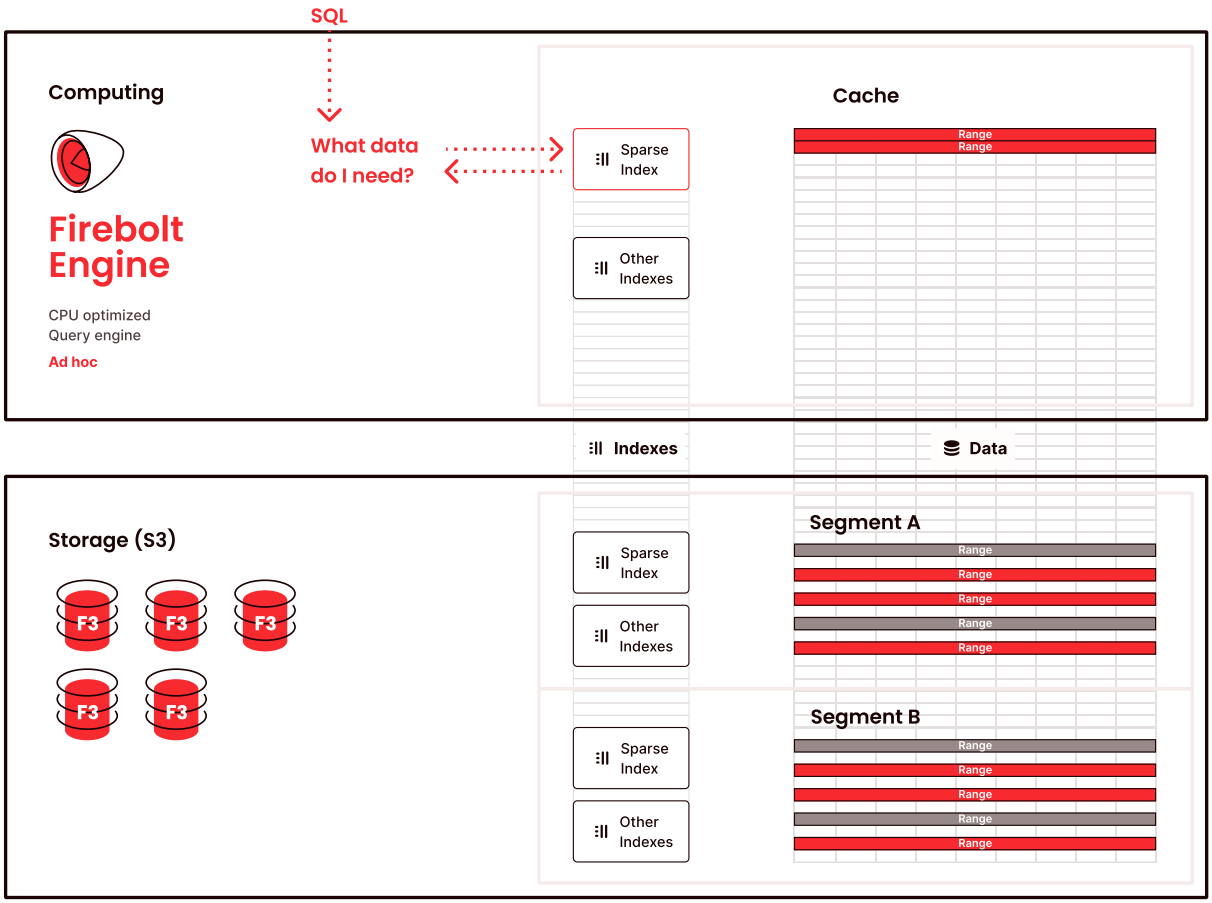

Figure 3a. Query optimizer identifies ranges needed in cache and storage

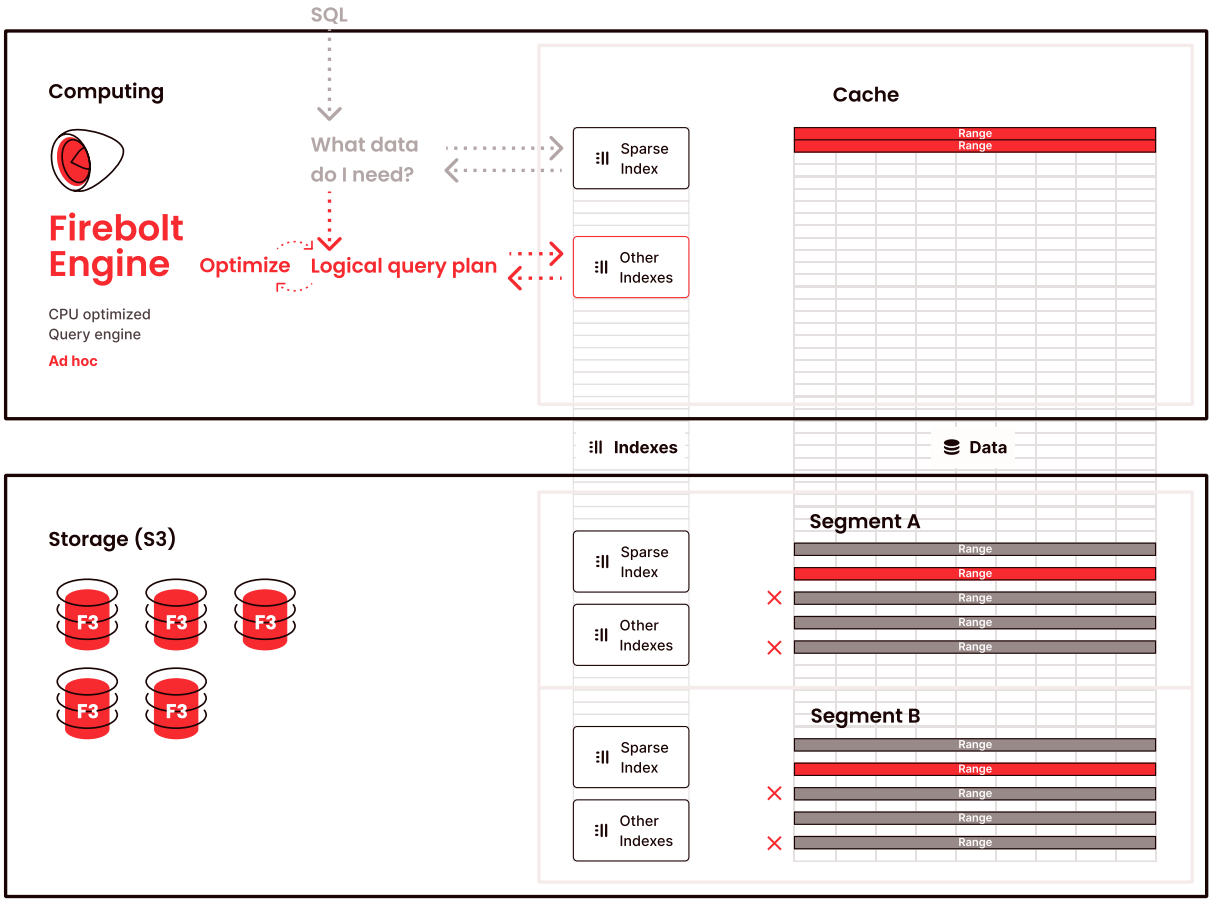

Figure 3b. Query optimizer optimizes query plan to reduce data needed from storage

Like several other data warehouses, the Firebolt Query optimizer parses each query, builds a logical query plan, and then uses cost-based and other optimization techniques to derive the best physical query plan.

But there are several key differences with Firebolt that help improve performance and efficiency. One key difference is Firebolt's use of sparse indexing. For each table, sparse indexing enables the F3 data access layer to reduce the amount of data fetched over the network, sometimes by 10x or more (Figure 3a.)

Another key difference is the level of optimization that is implemented. The query optimizer evaluates whether query performance can be improved by reordering query operations, or by using indexes in place of operations (Figure 3b.) It reorders operations to take advantage of predicate pushdowns, and uses aggregating and join indexes to further reduce both data access and scans There are several other

data warehouses that do predicate pushdowns, but they do just one level. For example, you can “pushdown” a predicate as a filter prior to doing a join. This helps reduce data sets before accessing data or performing joins. Firebolt can pushdown operations to many levels. This makes a big difference with both star and snowflake schemas where multiple fact or dimension tables are involved. Firebolt can also pushdown GROUP BY and other operations multiple levels, which is only supported to one level by just a few other databases.

```
SELECT
            dt.d_year,
            Item.i_brand_id,
            Item.i_brand,
            sum(ss_ext_discount_amt) sum_agg
FROM
            date_dim dt,
            Store_sales,
            item
WHERE
            dt.d_date_sk = store_sales.ss_sold_date_sk
            AND item.i_item_sk = store_sales.ss_item_sk
            AND item.i_manufact_id = 427
            AND dt.d_moy = 11
GROUP BY
            dt.d_year,
            Item.i_brand,
            item.i_brand_id
ORDER BY
            dt.d_year,
            sum_agg DESC,
            Brand_id
LIMIT 100;
```

Figure 4a. TPC-DS SQL query 3 before optimization

```
[0] [Alias] LOOKUP('idx_join_date_dim','d_year',NODE_1.ss_sold_date_sk) AS d_year, LOOKUP('idx_join_item','i_brand_id',NODE_1.ss_item_sk) AS brand_id, LOOKUP('i
 \_[1] [Projection] LOOKUP('idx_join_date_dim','d_year',NODE_1.ss_sold_date_sk), LOOKUP('idx_join_item','i_brand_id',NODE_1.ss_item_sk), LOOKUP('idx_join_item',
    \_[2] [Limit] 100 @ NODE_4
       \_[3] [Sort] LOOKUP('idx_join_date_dim','d_year',NODE_1.ss_sold_date_sk) (Ascending), SUM(NODE_1.SUM(ss_ext_discount_amt)) (Descending), LOOKUP('idx_join
          \_[4] [Aggregate] GroupBy: [LOOKUP('idx_join_date_dim','d_year',NODE_1.ss_sold_date_sk), LOOKUP('idx_join_item','i_brand',NODE_1.ss_item_sk), LOOKUP('
             \_[5] [Projection] LOOKUP('idx_join_item','i_brand_id',NODE_1.ss_item_sk), LOOKUP('idx_join_item','i_brand',NODE_1.ss_item_sk), LOOKUP('idx_join_da
                \_[6] [Aggregate] GroupBy: [LOOKUP('idx_join_date_dim','d_year',NODE_1.ss_sold_date_sk), LOOKUP('idx_join_item','i_brand',NODE_1.ss_item_sk), LO
                   \_[7] [Join] Mode: Semi [NODE_9.d_date_sk = NODE_1.ss_sold_date_sk] @ NODE_10
                      \_[8] [Join] Mode: Semi [NODE_11.i_item_sk = NODE_1.ss_item_sk] @ NODE_12
                      |  \_[9] [StoredTable] Name: 'idx_agg_store_sales' pruned: 0/3 column(s) AGGREGATING_INDEX_TABLE @ NODE_1
                      |  \_[10] [Predicate] NODE_11.i_manufact_id = 427 @ NODE_13
                      |     \_[11] [StoredTable] Name: 'item' pruned: 20/22 column(s) DIMENSION @ NODE_11
                      \_[12] [Predicate] NODE_9.d_moy = 11 @ NODE_14
                         \_[13] [StoredTable] Name: 'date_dim' pruned: 26/28 column(s) DIMENSION @ NODE_9
```

Figure 4b. TPC-DS Firebolt logical query plan for query 3, from the Firebolt explain plan

What might start as a query with complex nested joins along with several WHERE (predicate), GROUP BY, and ORDER BY operations (Figure 4a) can be replaced with several levels of pushdown predicates and LOOKUP operations (Figure 4b) - all of which can dramatically improve performance by reducing the amount of data needed from storage and the number of scans. The final result is a physical distributed query plan assembled just-in-time to deliver the best performance based on the available indexes and the location of all the data needed across nodes for the query.

Query execution

The final step is the actual query execution. Many of the innovations with query execution come from the integration with query optimization, storage, and indexing to maximize performance, scale, and efficiency. Federated query engines such as Presto are not as coupled to storage. Others such as Snowflake do not distinguish between storage and SSD, and store entire partitions in cache, not more specific ranges. Firebolt tracks which data is in storage and cache, and then builds a query plan to maximize performance.

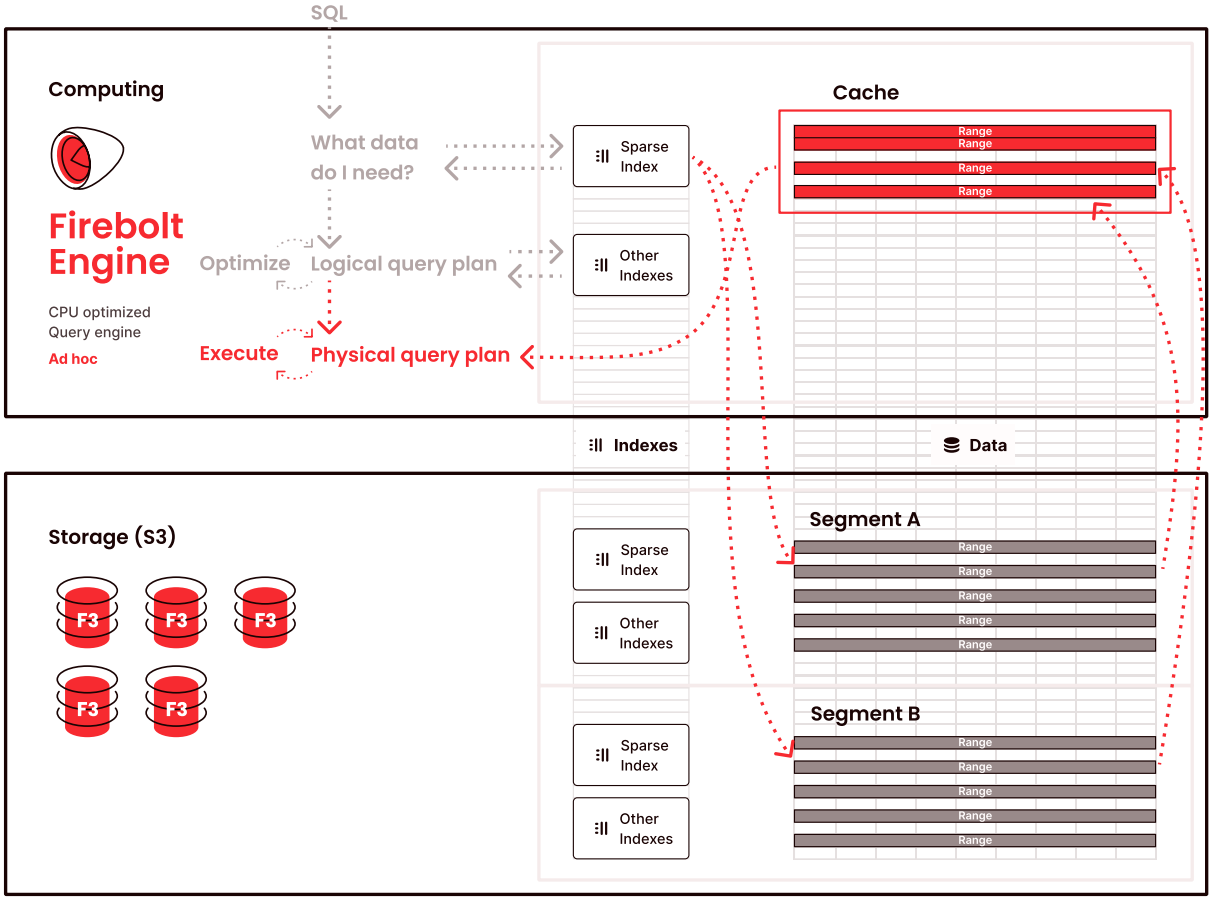

Figure 5. Firebolt execution of the physical query plan

Within the query execution engine there are several innovations inherited from public research and previous generations of on premises and cloud data warehouses. Two of the more common ones are:

- **LLVM (Low Level Virtual Machine):** Firebolt is built in C/C++, and uses LLVM to dynamically and quickly optimize query plans prior to execution.

- **Vectorized processing:** Firebolt leverages query vectorization to process entire columns and data ranges instead of just one row at a time. Grouping entire columns into single operations dramatically accelerates performance by reducing the number of I/O and compute operations.

There are also several innovations that are unique to Firebolt, discovered and implemented based on internal research and work with customers. Beyond the pushdown and query optimization techniques mentioned, Firebolt optimizes data and compute collocation, with more improvements expected through the use of machine and deep learning techniques.

- **Index-based optimization:** Each physical query plan is built in part using the indexes to build the fastest queries based on the current location of data ranges. An initial set of ranges is gathered as an index "set" which gets reduced through the query optimization process. This final index set is used to execute the query.

- **Data and compute collocation**: Unlike other cloud data warehouses, Firebolt distributes data ranges differently in each engine's cache based on each query's needs. The query optimizer will distribute queries across nodes for each engine in a way that helps maximize data collocation and minimize remote data access times. For example, it will collocate data by foreign key to improve join performance. Every engine can have completely different collocation of the same data. You can also explicitly shard data on read by a column or expression that works together with the primary index.

- **Automatic data rebalancing:** During ingestion, Firebolt will look to rebalance, or combine segments and their indexes in S3 as needed. There is no need to

"vacuum" and do other types of manual rebalancing or maintenance.

- **Reuse of compute sub-trees and operators:** Beyond just reusing data in caches, Firebolt also reuses parts of query plans that it has already compiled. The collocation of data and compute increases reuse, further improving end-to-end performance.

- **Explain plans:** Firebolt exposes the logical query plans (Figure 4b) to help users understand the optimizations. It also collects statistics on performance. Firebolt is increasingly using this information to help make recommendations.

Scalability

There have been two major advancements in data warehouse scalability over the last two decades. The first was a shared nothing architecture, which started with partitioning data and queries across nodes. It helped deliver linear horizontal "scale-out" scalability for the first time. But you still had to have all data on each cluster.

The second was a decoupled storage and compute architecture, which added the ability to have different compute clusters running different queries and retrieve data subsets "on demand" as needed from remote storage. It improved scalability by allowing different compute on different clusters. It made scaling more elastic as well since you could easily provision and resize new compute clusters.

Modern cloud data warehouses with decoupled storage and compute and a shared nothing architecture are able to support petabyte-scale data by storing any size data in remote storage separate from any cluster. They have been able to support large, complex queries by scaling up node sizes, adding more nodes, and running different queries on different clusters. They have also been able to support high user concurrency by replicating clusters and partitioning users across the clusters.

But other scalability bottlenecks remain even in the most modern cloud data warehouses. These include:

- **Data ingestion:** most data warehouses are still limited to batch-centric ingestion. They do not support low latency, streaming ingestion at scale. This is in large part because most data warehouse storage is columnar and requires rewriting entire partitions for a single row update.

- **Data access:** most decoupled storage and compute architectures move entire partitions from storage into the compute cluster. This is a major problem because it makes the network the biggest bottleneck.

- **Query scalability:** Queries with large joins or complex nesting can require a lot of SSD to store partitions, a lot of RAM to hold data while processing, and a lot of compute for scanning and complex processing. In many cases the only solution is to use very large node types.

- **Semi-structured data:** Most cloud data warehouses either store semi-structured data as flattened strings, or don't support formats like JSON at all and require you to "flatten" or "unnest" it into columns in tables. But processing strings ends up taking a lot of compute and RAM to hold all the JSON for scanning, which limits scalability.

Data ingestion

One of the oldest bottlenecks in data warehouses including modern cloud data warehouses, is ingestion. Most are still only optimized for batch-based ingestion. Historically, the best way to extract data from most legacy applications was batch. Over time, as businesses wanted more real-time visibility, real-time data pipelines started to be built, using a host of technologies from Change Data Capture (CDC) to messaging. But most data warehouses remained batch-centric, which became more evident as streaming data volumes grew. Most cloud data warehouses do not support continuous ingestion at any reasonable scale. They rely on micro-batch at

best behind the scenes, which leads to a minute or longer delay between ingestion and when you see data in queries. The main reason is that ingestion is constrained by storage.

Most cloud data warehouses rely on columnar storage with relatively large partitions that are immutable. Each column in each partition is usually compressed as a set of separate files. In order to add or update a row, you have to rewrite all the columnar files associated with the row. With continuous ingestion, you either rewrite partitions with each new row, or you batch up changes. You can batch up new data into new partitions and eventually combine and resort partitions later. But updates still require rewriting.

Firebolt re-architected ingestion and storage to support not just batch, but continuous ingestion and low latency analytics as well.

The first change was to act a little like a federated query engine. Any Firebolt engine can ingest data from external files by exposing them as a relational table. Anyone can then use SQL to select the relevant data from the external files or other data sources, perform any operations, and then insert into target tables into

F3. This enables data engineers to do their own ELT and build dashboards without depending on others to complete their work first.

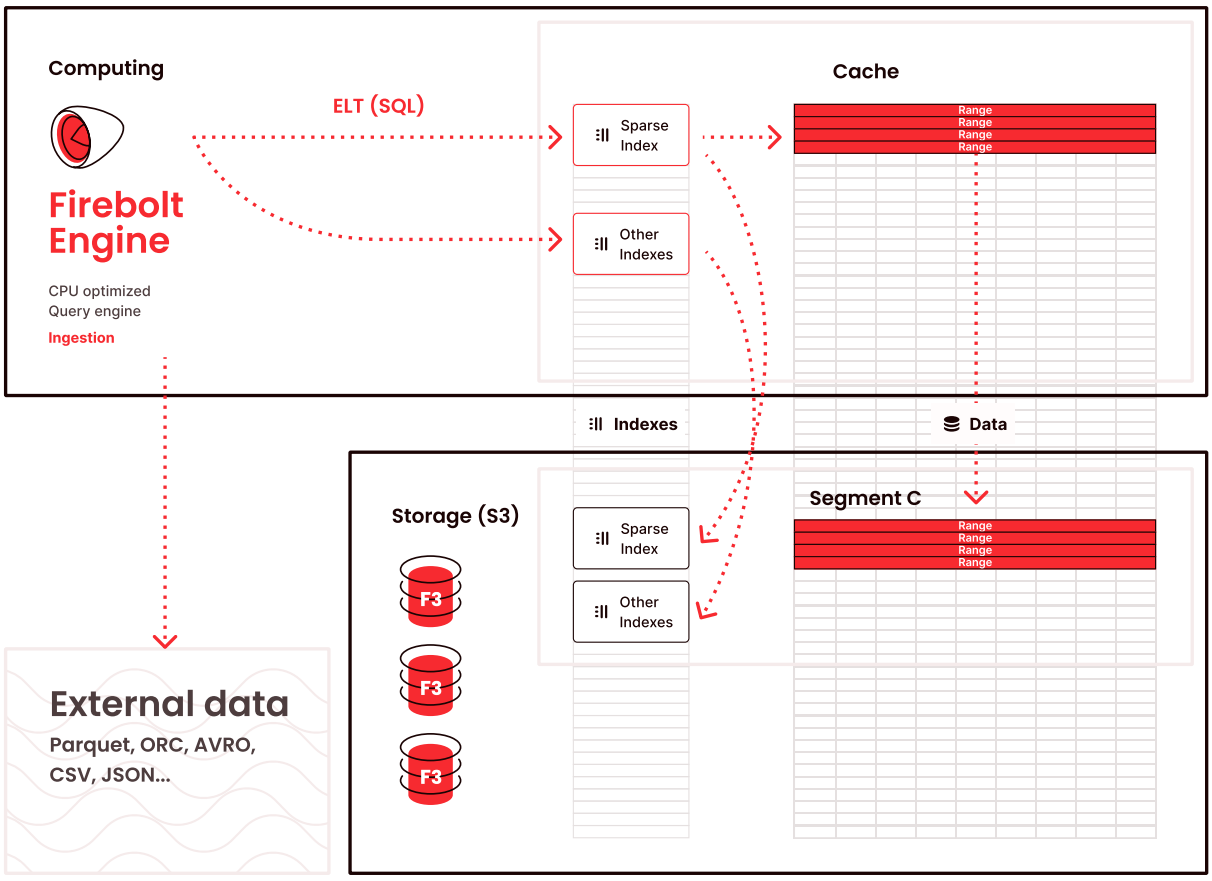

Figure 6. Firebolt data ingestion process

In Firebolt, any engine can be used for ingestion. Each node in the engine performs non-blocking, immediate ingestion of any number of rows whether it is 1 or 1 million rows per write as a new segment. If you want to scale out ingestion, you can add more nodes to perform parallel ingestion. The nodes will work together to optimize the ingestion process. Data is sorted and compressed as it is written as data ranges within the segment. Segments and their ranges are also combined as needed by the ingestion engine to optimize query performance.

This works because each ingestion or rewritten segment operation is transactional, meaning the data is committed to the cache and guaranteed to commit to S3. Each index involved with the table write along with the segment is part of the same

transaction. This makes each read consistent because each read is executed via the F3 data access layer and the sparse index.

With the sparse index, segments do not matter to the query engine; they are only for remote storage optimization. Each engine only requests and stores data ranges in cache. This allows new segments of any size to be written.

Semi-structured data performance and scalability

Today there is much more “Big Data”, including semi-structured data in formats like JSON, than ‘transactional” data. But support for semi-structured data is limited or nonexistent in most data warehouses. The traditional approach has been to flatten, or unnest data into tables. But that often results in needing nested queries to analyze the data, which cripples performance. A few data warehouses added support for storing JSON as text. But that requires the query engine to load all the JSON across all the rows into RAM and perform full text scans for processing. This results in the need for a lot of RAM, and a lot of compute power.

Firebolt ingests, stores, and processes JSON as native JSON. You ingest and query JSON using SQL with array functions and Lambda expressions. During ingestion, you can load JSON directly, or partially UNNEST the JSON as columns. JSON can be automatically stored as a nested array type. The array functions and Lambda

expressions traverse the nested arrays directly instead of having to load all the data into RAM. The result is much better scalability, with much less RAM and CPU needed, which leads to lower costs as well as much faster query times.

```
{
        “id”: 1,
        “StartTime”: “2020-01-06 17:00:00”,
        “Duration”: 450,
        “tags”: [“summer-sale”,”sports”],
        “user_agent”:{
            “agent”: “Mozilla/5.0”,
            “platform”: “Windows NT 6.1”,
            “resolution”: “1024x4069”
        },
        “events”:[
            {
                “EventId”: 547,
                “type”:”login”,
                “Timestamp”:”2020-01-06 17:00:10”,
                “EventProperties” :
                {
                    “UserName”:”John Doe”,
                    “Successful”: true
                }
            },
            {
                “EventId”: 548,
                “type”:”add-to-cart”,
                “Timestamp”:”2020-01-06 17:01:15”,
                “EventProperties” :
                {
                    “ProductID”:”xy123”,
                    “items”: 2
                }
            }
        ]
    }
```

Figure 7a. Sample JSON.

Visit

Id: 1

Start_time: 2020-01-06 17:00:00

Duration: 450

Tags
“summer-sale”
“sports”

user_agent_properties

property_name	property_value
“agent”	"Mozilla/5.0"
"platform"	"Windows NT 6.1"

event

id	type	Timestamp	properties	
574	“login”	"2020-01-06 17:00:10"	**property_name**	**property_value**
			“agent”	"Mozilla/5.0"
			"platform"	"Windows NT 6.1"
548	“add-to-cart”	"2020-01-06 17:01:15"	**property_name**	**property_value**
			“ProductID”	**“xy123”**
			“items”	**2**

Figure 7b. Resultant row after JSON ingestion.

For example, consider the above JSON (Figure 7a) and how it is stored in Firebolt (Figure 7b.) While you could nest the entire JSON, you can also create a fact table (Figure 7c) and partially UNNEST some data that can then be used with a sparse index to optimize performance. You can also use any combination of UNNEST and ARRAY functions within queries (Figure 7d), as well as Lambda expressions

as a function. In all cases, Firebolt is directly traversing the native nested array structure, not doing text scans, which improves performance and requires less cache and compute.

```
CREATE FACT TABLE Visits (
   Id INT,
   StartTime TIMESTAMP,
   Duration INT,

   -- zero or more tags
   tags Array(TEXT),

   -- user agent properties will consist of two arrays,
   -- one for names and one for values,
   -- they will alway in the same length per specific row

   user_agent_properties_names Array(TEXT),
   user_agent_properties_values Array(TEXT),

   -- Events consist of an array for IDs, array for types,
   -- and two nested arrays for event property names and event property
      values

   events_ids Array(INT),
   events_type Array(TEXT),
   events_properties_names Array(Array(TEXT)),
   events_properties_values Array(Array(TEXT))
)
```

Figure 7c. SQL (DDL) for creating a fact table with partially unnested JSON

Using UNNEST

```
SELECT
   id
FROM
   visits
UNNEST (tags)
WHERE
      tags = 'sale'
   OR tags = 'fashion'
GROUP BY
   id
ORDER BY
   duration DESC
```

Using array functions

```
SELECT
   id
FROM
   visits
WHERE
   has(tags, 'sale')
   OR has (tags,'fashion')
ORDER BY
   duration DESC
```

Using Lambda expressions

```
SELECT
   id
FROM
   visits
WHERE
   has(tags, 'sale')
   OR has (tags,'fashion')
ORDER BY
   duration DESC
```

Figure 7d.UNNEST, array function, and lambda expression examples.

Efficiency

There has been one major efficiency gain from decoupled storage and compute; you can start and stop compute clusters at any time. This can dramatically reduce the cost of compute for “bursty” workloads such as batch-based ingestion or widely varying query workloads.

But cloud data warehouses can still become expensive, especially when they allow any query or are used as a data lake for raw data. This is mostly due to the cost of cloud data warehouse compute, and a lack of compute efficiency. All too often scale-out databases have relied on a shared-nothing architecture to improve performance by adding more compute instead of making each node and cluster more efficient. This has led to companies throttling their analytics in order to contain costs, which goes against the business goal of opening up analytics to more people, to enable better, faster data-driven decision making.

Firebolt is designed not just for performance, but for performance, scalability and efficiency combined. It includes a host of features that enable companies to scale and choose the optimal price-performance, or performance at a given cost. The combination of the greater efficiency across the query pipeline, from more efficient storage and indexing to query optimization and execution, combined with greater choice of the best resources, is what has delivered 10x or greater price-performance compared to the alternatives when running on similar infrastructure.

Choice of node types

Part of the way to get the best price-performance is to choose the best combination of compute, cache, and scale-up or scale-out configurations for each workload or query. Node types can be more I/O intensive to support ingestion, compute intensive to support queries, or balanced for both. Depending on the types of queries, scaling up or out might make more sense to improve price-

performance. Firebolt allows administrators to choose any size and type of node for each engine, and any number of nodes (up to 128) for each engine to achieve the optimal price-performance. Segmenting workloads such as ingestion and queries, or different users with different SLAs, enables even greater price-performance optimization.

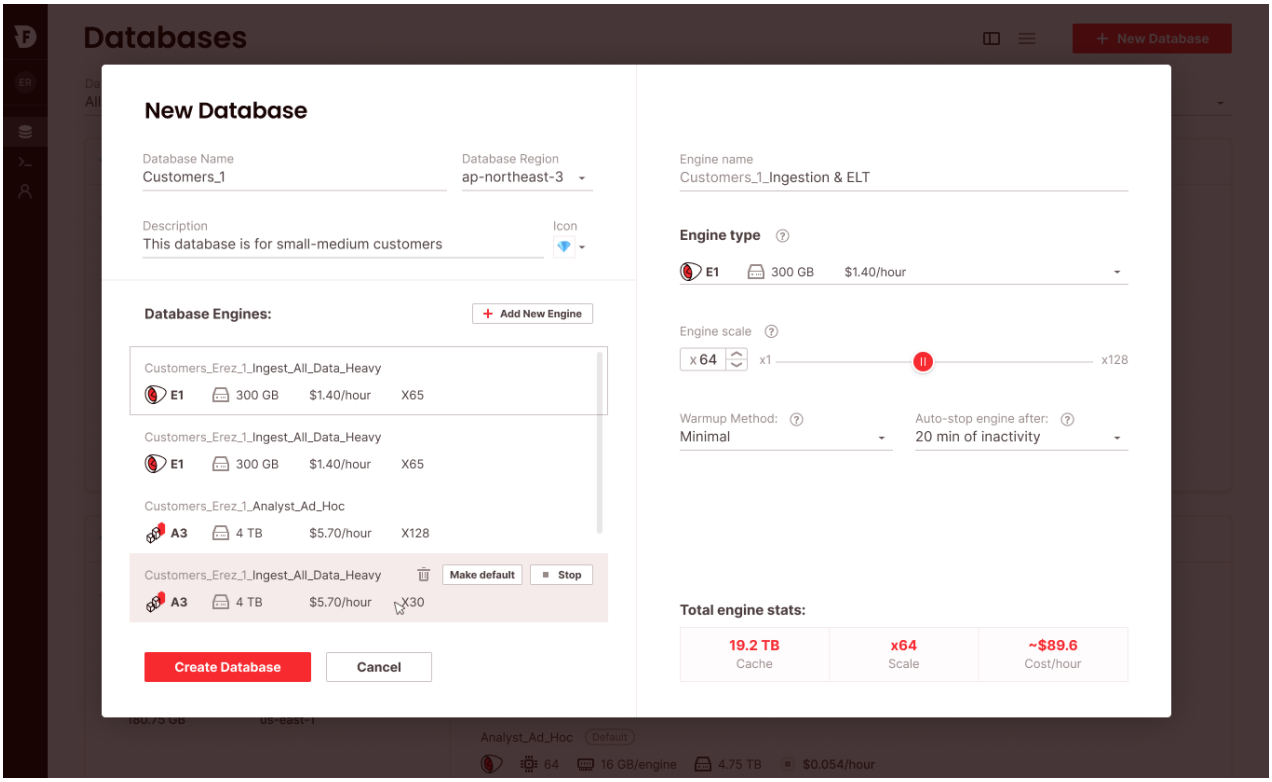

Figure 8. Firebolt Manager - How to choose different node types

Efficiency through indexing

Indexing helps improve efficiency by reducing node size requirements and the amount of required compute for each query without doing any additional optimization. The sparse index for each table helps reduce cache and compute requirements for each engine by only accessing and storing those ranges needed by the users and their specific queries. This can dramatically reduce the cache requirements even when compared to other decoupled storage and compute architectures because ranges are much smaller than segments/partitions. Less data also requires less compute since all scans are smaller. Even without any indexing, the native JSON storage and Lambda expressions within SQL can also reduce cluster requirements by 10x.

Adding more indexes not only improves performance. It can also reduce compute requirements even more. For any repetitive “ad hoc” or dashboards, aggregating indexes can dramatically reduce costs by precomputing the required aggregations and other operations once during ingestion. Firebolt will only access and use the index required for the metrics, not the raw data. Join indexes reduce the compute costs for fact and dimension joins in a similar way by reducing the number of scans.

In early deployments, ingestion resulted in 10-15% of the total compute costs. This included loading the data from external tables into Firebolt. Adding more indexes added less than 3% total additional cost. This is why indexing in some cases has saved up to 50% in compute costs for analytics through reduced compute requirements.

Storage optimization

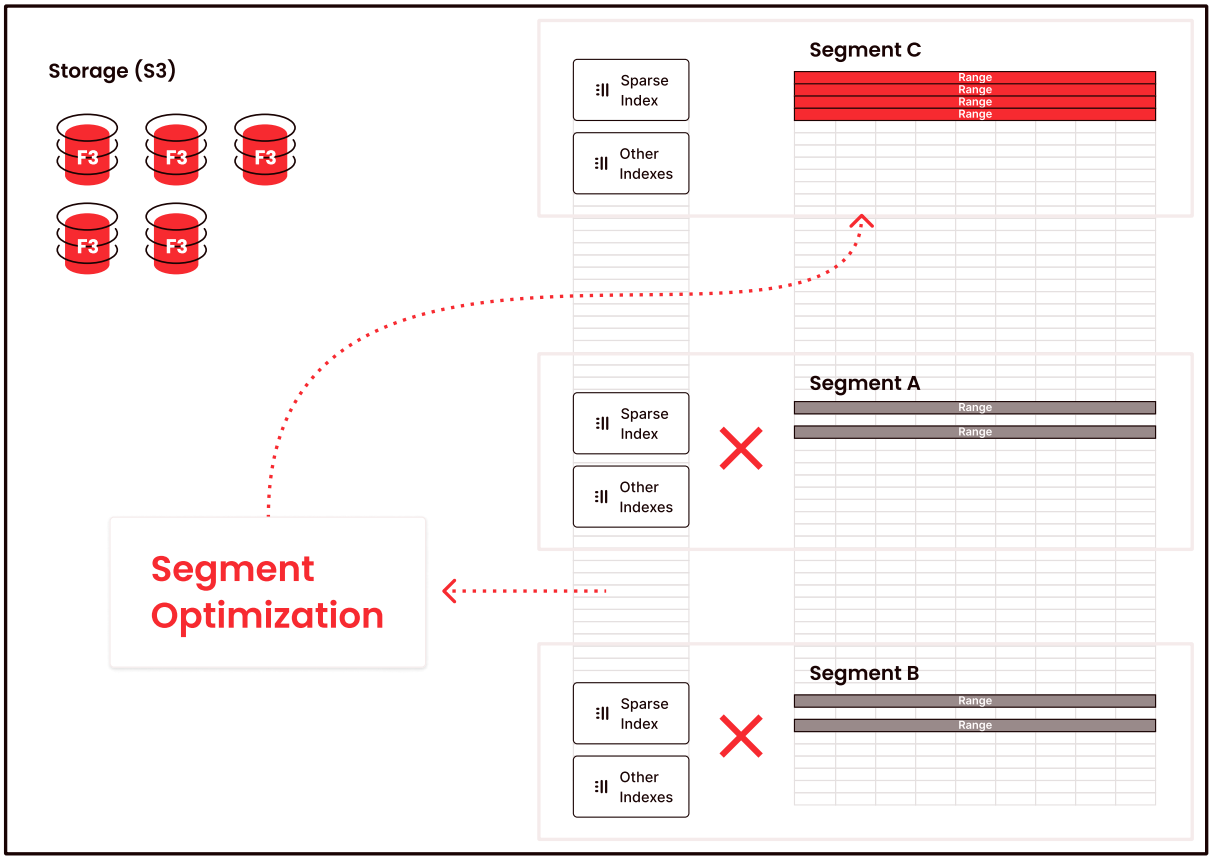

Figure 9. F3 segment optimization process

Firebolt storage does not require any manual optimization such as re-sorting and rebalancing partitions, or vacuuming. It automatically combines segments in the background both during ingestion and after to improve performance. For example, continuous ingestion will add a lot of new segments with very low latency, and data ranges will overlap. Over time, Firebolt folds newer segments into older ones and also combines ranges whenever it believes it will help with performance. Indexes are updated to point to new ranges whenever the changes occur.

Summary

The need to support not just traditional analytics by analysts, but operational and customer-facing interactive analytics by employees and customers, has completely changed what is required from a cloud data warehouse. Unlimited elastic scalability is not enough. They now must deliver sub-seconds performance at any scale to 10-1000x more users at 10x or lower costs.

Firebolt had to be completely redesigned from the ground up as a cloud data warehouse with a new decoupled storage and compute architecture. Tiered data management, extensive indexing, advanced query optimization and execution were all needed to deliver the order-of-magnitude improvements in speed, scale, and efficiency. Any alternative will require similar levels of reengineering to be successful. But it is clearly possible. It proves you do not have to sacrifice the features and flexibility of a SQL-based cloud data warehouse to get the performance and cost at scale needed to deliver analytics to everyone. You may just need to adopt a new data warehouse.

About Firebolt

Firebolt is the world's fastest cloud data warehouse, purpose-built for high performance analytics. It provides orders of magnitude faster query performance at a fraction of the cost of the alternatives by combining the simplicity, elasticity and low cost of the cloud with the latest innovations in analytics. Companies that adopted Firebolt have been able to deploy data warehouses in weeks and deliver sub-second performance at terabyte to petabyte scale for a wide range of interactive, high performance analytics across internal BI as well as customer facing analytics use cases.

Firebolt is founded by members of the original team at Sisense, A global BI leader, and backed by Zeev Ventures, TLV Partners, Bessemer Venture Partners and Angular Ventures.

Contact Firebolt

For more information about Firebolt, you can contact us at anytime at:

https://www.firebolt.io/contact

Or e-mail us at: hello@firebolt.io.