Elasticsearch vs. Rockset

ROCKSET

Table of Contents

Abstract

Converged Indexing™ enables faster time to market and up to 50% lower TCO as compared to Elasticsearch's search indexing, for real-time analytics use cases. This is achieved by optimizing for hardware and developer efficiency in the cloud.

Introduction

Both Elasticsearch and Rockset are queryable data stores that can store data and serve queries. Both of them index data and use the index to serve queries. Both systems are document-sharded, which means that documents are routed to a shard based on the document id. But that is where their similarities end. This paper compares some of the technical differences between Elasticsearch's search indexing and Rockset's Converged Indexing

- Elasticsearch uses search indexing, and is optimized for text search and log analytics use cases
- Rockset uses Converged Indexing, and is optimized for real-time analytics and real-time applications

For real-time analytics use cases, data indexed includes event streams, user behavior data, sensor data, device metrics, database change data capture from OLTP database and third-party data from data lakes. Once indexed, it is primarily used for serving real-time analytics and real-time applications.

Real-time Analytics

- Ad optimization
- A/B testing
- Fraud detection
- Real-time 360

Real-time Applications

- Logistics or fleet tracking
- Gaming leaderboards
- Personalization
- Real-time recommendations

Design goals for converged Indexing:

Better scaling: Achieve low latency queries even with high velocity ingest and massive data volume, by enabling efficient compute & storage scaling in the cloud

More flexibility: Enable fast search, aggregations and JOINs across semi-structured data formats. Avoid denormalization, since real-time data comes in different shapes from different sources, and the types of queries are not always known ahead of time.

Low ops: Optimize for ops efficiency in cloud environments by avoiding manual management of indexes, shards, clusters or node types.

Indexing Technique

History

Elastic uses open source Apache Lucene as a storage engine. Lucene started in 1999 as a text-search library on spinning disks. On the other hand, Rockset uses open source RocksDB as its storage engine. RocksDB started in 2012 as an SSD based embedded key-value store used by Facebook and LinkedIn to index massive datasets.

Search vs Converged Indexing

Elasticsearch builds an inverted index and a range index on your data. This means that filter queries and range queries are served optimally. If you enable doc_values, then you have configured Elastic to behave like a document index where you can find all the fields of a record efficiently. The doc_value configuration is similar to a record store (like Postgres or MongoDB) and allows you to do aggregations on a specific column. These aggregations are not as efficient as it could be in a true columnar storage system (like a data warehouse).

On the other hand, Rockset builds a Converged Index on your data, the Converged Index includes an inverted index and range index just like Elasticsearch, but also includes a columnar store (like a data warehouse) and a record store (like Postgres). Rockset stores all values of a single column co-located in the columnar store and Rockset queries serve low selectivity queries (e.g average, standard deviation, min, max) using this columnar store. Since the values of a column are packed together, Rockset can employ vectorization techniques to process large vector of column values very quickly.

Architecture

Decoupling Compute & Storage

Elasticsearch was created in 2010 when the cloud was nascent. It was optimized for the datacenter and does not decouple compute from storage. It has DataNodes that store the index as well as serves queries on the index. The data size and the query performance are closely tied together, and the ratio of compute to storage size is fixed for your hardware. If you need to change this ratio for other use-cases or for managing price-performance for hot-warm-cold data, you have to manually migrate your cluster to run on a different hardware instances.

Rockset, on the other hand, is cloud-native and decouples compute and storage:

- Data is served from hot storage, and automatically backed up on durable cloud storage.
- Compute resources are scaled independently as required.
- The compute resources are used for ingesting data or querying data from hot storage. Scaling compute resources results in instant performance gains.

For every independent workload, you can spin up fully isolated compute resources in the form of Rockset Virtual Instance.This gives your developers flexibility and agility to serve a variety of workloads. Scaling storage independently from compute allows developers to seamlessly control their price-performance over the lifecycle of the dataset, simply by adjusting the compute allocation over time. This type of hardware efficiency, combined with serverless operations results in efficiency up to 50% lower Total Cost of Ownership (TCO) when using Rockset versus Elasticsearch for real-time analytics.

Separation of durability and performance

Elasticsearch adopts a shared-nothing storage architecture where data durability is guaranteed via replication among data nodes. This architecture was common in the pre-cloud days of 2010 when data stacks were mostly deployed on-premises. Depending on the failure rate of your hardware, you would typically configure two or three replicas of your index. Even if an index is not being queried aggressively, you still need to keep these replicas for durability of your data. You have to manually configure the resiliency of new writes.

Rockset is built for the cloud, and it leverages the cloud's shared-storage model. All data is made durable by storing it in cloud storage (S3, GCS, etc). Replicas are made only if the query or update volume of an index increases. The Rockset system can serve data with just keeping a single replica in SSD based systems, because the other durable copy resides in cloud storage.

Ingesting Data

Click and Connect Data Integrations

If you have to continuously sync data from an external data source to Elastic, you have to deploy Elastic Beats agents or use LogStash. The configurations for these connectors need to be deployed, monitored, upgraded and maintained. For example, if you have to sync a dataset from S3 (in parquet format), another dataset from Kafka (in avro-json format) and yet another dataset from MongoDB (via change-streams) into Elastic, you need a plethora of configurations and manual configurations.

On the other hand, Rockset uses a Click-and-Connect architecture to synchronize data from external sources into the Rockset index. The Rockset service has a set of Ingesters that continuously monitors each of the configured sources and as soon as new data becomes available at the source, it pulls the data into its Index. This provides you with hands-free operations, because you do not have to babysit and shepherd hundreds of Beats and LogStash configurations for each of your data formats and data sources.

Indexing at high write rates

Elasticsearch uses a primary-backup model for replication. The primary replica processes an incoming write operation and then forwards the operation to its replicas. Each replica receives this operation and re-indexes the data locally again. This means that every replica independently spends costly compute resources to re-index the same document over and over again. If there are n replicas, Elastic would spend n times the cpu to index the same document.

Rockset is a primary-less system. New writes are written to a distributed durable log and all replicas tail that log and apply the operation locally to make the data visible to queries. Only one replica does the indexing and compaction using RocksDB's remote compaction and the other replicas fetch and swap newly generated files from cloud storage. Thus, Rockset spends only 1x the cpu for indexing even when there are multiple replicas in the system. Rockset employees a cloud-native primitive called

Remote Compaction offered by RocksDB, that allows it to disentangle compaction cpu from storage nodes, thereby able to speed up indexing speeds even in the face of bursty writes.

Data latency at massive write rates

Data latency is the time duration between when you make an update to your data and when it is visible to a query. Reducing this metric is critical for powering realtime applications. Elastic has a configuration called the refresh interval, Elastic buffers incoming writes into a in-memory buffer and every time a refresh interval expires, it creates a new on-disk segment and makes that data available for queries. To support a higher write rate, you have to increase the refresh interval, which is turn makes you suffer from higher data latency.

On the other hand, Rockset uses open source RocksDB's Log Structured Merge tree to make writes visible to queries as soon as it is written; it too uses a in-memory buffer to cache incoming writes into RocksDB's memtable, and a lockless protocol makes this memtable be visible to existing queries as soon as it is written and even before it is written to storage. This is critical to reducing data latency that is much needed for powering real-time applications.

Queries

Routing

Both Rockset and Elastic are document-sharded systems that are designed for low latency queries. A document is routed to a shard based on its id. A query hits all the shards in the index, processes the query in parallel and returns results.

SQL joins and aggregations

Elasticsearch does not support joins; one way to join two datasets in Elastic is to write the join code as part of your application and Elastic advises that these types of queries are prohibitively expensive. Yet another way is to do write-time de-normalization, but this has the limitation that you need to know the types of queries before-hand so that you can de-normalize the appropriate fields at write time. Elastic supports simple bucketization and counts/min/max using pipelined aggregations. But aggregation queries that need to create large transient data sets (e.g. count distinct) are not supported. The reason being that Elastic does not support distributed aggregations.

Rockset supports a full featured SQL language including joins. A multi level aggregator executes Rockset's join operator. Aggregators are distributed and the JOINs are executed in a parallel fashion over multiple aggregators for both scalability and speed. This means that a single query can use the memory on a large set of aggregator machines to hold and process transient datasets. Combined with the columnar store described in the above paragraph, this single feature enables your developers to be agile because they do not have to worry about joining multiple large datasets as part of an application query.

Support for visualization tools

Most enterprise BI platforms (like Tableau) need a JDBC/ODBC api to access data in a datastore. Elastic supports Kibana for visualization but does not provide first-class support for many BI tools, so connecting it to Tableau visualization tools is difficult and time consuming. Rockset supports a full featured JDBC driver and all standard SQL visualization tools can access it seamlessly.

DATA API & Developer Tooling

Both Elasticsearch and Rockset have extensive REST APIs which allow you to integrate, manage and query the indexed data. Rockset has additional developer tooling such as support for Query Lambdas, which are named parameterized SQL queries stored in Rockset that can be executed from a dedicated REST endpoint. With Query Lambdas, you can version-control your queries in the form of data APIs so that developers can collaborate easily with their data teams and iterate faster.

Scaling to massive datasets

An Elastic index is organized in the form of a set of shards. The number of shards determines the maximum number of nodes on which the dataset can be hosted. When an index is growing in size and you have already spread out all your shards on different machines, your queries become slower and slower and your only option is to create a new index with a larger number of shards and then reindex all the data from the existing index into the new index. This reindexing process is very costly in terms of resource consumption. The amount of cpu and IO needed to reindex tens and hundreds of terabytes at a regular periodic basis is very high.

On the other hand, every Rockset index is designed to scale up to hundreds of terabytes without needing to ever reindex a dataset. A Rockset index is organized in the form of thousands of micro-shards, and a set of micro-shards combine together to form appropriate number of shards based on the number of available servers and the total size of the index. If the dataset increases in size, a subset of micro-shards are peeled away from every existing shard and made into additional independent shards and these additional shards are distributed to the new machines in your cluster. No reindexing is needed and this feature is super-critical to support cloud-scale datasets.

Scaling to a variety of applications

Elastic is primarily built for log processing where the primary workload is to add new documents that rarely update existing documents. Elastic do support updates of documents but if you want to update one field of a document, Elastic will internally read all the fields of the existing document, apply the update to that field and delete the original document and rewrite the entire new document to the data store. Reindexing an entire document on Elastic consumes cpu resources.

Rockset, on the other hand, is optimized for mutating a single field without having to reindex the entire document. Rockset's Converged Indexing stores every individual field of the document into an independently addressable key in the open source RocksDB database, which means that when a field need to be updated, Rockset can update that single key without having to reindex the entire document. Elastic's update-an-entire-document implementation is suitable for logging events when updates are rare. On the other hand, Rockset is built for powering real-time applications where your application need to tag and update individual pieces of your dataset at different points in time, and its field level mutability becomes super handy for scaling to a variety of real-time applications.

Serverless operations

Elasticsearch requires deep expertise for controlling costs at scale. It requires configuring clusters with different node types, pre-configuring the number of shards in an index, tuning the amount of CPU per node, configuring thread-pools, and moving indexes between hot-warm-cold nodes to manage the index lifecycle as data ages. On the other hand, Rockset's cloud-native serverless architecture is optimized for hands-free operations, while providing visibility and control. It independently scales compute & storage resources, and automates management of clusters, shards, indexes and data retention based on policies set by the user.

Summary

Rockset's Converged Indexing enables faster time to market and up to 50% lower TCO compared to Elasticsearch's search indexing, for real-time analytics use cases. This is achieved by optimizing for hardware and developer efficiency in the cloud.

- Better scaling: Low latency queries even with high velocity ingest and massive data volume. Decoupled compute and storage for better price-performance as ingest, queries and data volume scale

- More flexibility: Standard SQL including JOINs on semi-structured data. No need to denormalize data, allowing for any types of queries down the road so teams can deliver on any future roadmap requirements. Data APIs for developer-friendly real-time analytics
- Low ops: Serverless auto-scaling in the cloud. No indexes, shards, clusters or node types to manage.

References:

https://tech.ebayinc.com/engineering/elasticsearch-performance-tuning-practice-at-ebay/

https://logz.io/blog/the-top-5-elasticsearch-mistakes-how-to-avoid-them/

About Rockset

Rockset is a real-time indexing database service for serving low latency, high concurrency analytical queries at scale. It builds Converged Indexes™ on structured and semi-structured data from OLTP databases, streams and lakes in real-time and exposes a RESTful SQL interface. It is used for serving real-time analytics and real-time applications.

Learn more at rockset.com

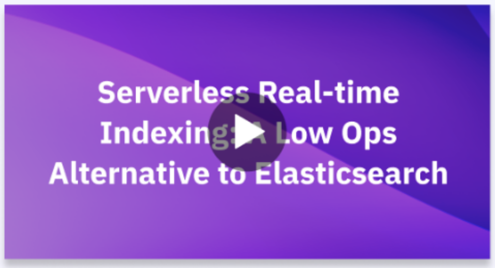
Connect with us at hello@rockset.com

Built for Speed at Scale

Elasticsearch is known for speed but it's a pain to scale. With Rockset, teams achieve sub-second analytics at cloud scale. They are freed from inflexible and hard to manage Elasticsearch. As a result, they can build and iterate on data applications faster.

Get Demo

Try Free



Elasticsearch Scaling Challenges Impacting Your Release Cycles?

Watch a talk with former Elasticsearch solutions architect Ben Hagan as he compares some of the ops differences between Elasticsearch and cloud-native Rockset for real-time analytics.

Watch Now

Getting to a Positive ROI on Real-Time Analytics

Learn how the architectural and design decisions behind Rockset are increasing the ROI for engineering teams.

Read Now