ByteHouse

ClickHouse vs. Elasticsearch

A Publication of ByteHouse

About

ByteHouse is part of ByteDance, a global technology platform dedicated to creating innovative solutions to entertainment, collaboration, education, and business operations.

ByteHouse enables real-time analytics with lightning-fast data warehousing products and solutions.

You can sign up for a free trial here with US$400 credit.

ByteHouse

Part of ByteDance

Data warehousing products and solutions delivering significant lift on data compression, speed, and scalability.

Plus, less hardware, low maintenance, dedicated support, lower TCO, multiple deployment options.

Cloud-Native SaaS

Private Cloud (VPC)

On-Premise

Start Free »

Popular uses cases of ByteHouse: real-time data analytics and OLAP events & log processing, A/B experimentation, reporting, deep data analysis, etc.

Apache Airflow tableau Apache Superset Grafana kafka

Table Of Contents

01

CHAPTER ONE

Introduction

Introduction

In the early years, many companies adopted Elasticsearch to serve as their distributed log aggregator. Elasticsearch is able to provide near-real-time indexing and full-text search functionalities on large volumes of logs.

With its popularity, a number of companies not only use Elasticsearch in the domain of search but also extend their usages to analytic queries. But, in recent years, I have observed more companies adapting ClickHouse* over Elasticsearch for their OLAP workloads.

In particular, ContentSquare reported queries in ClickHouse are 4 times faster, and 11 times cheaper; Uber moved to ClickHouse to manage service logs at massive scale and observed 10x performance increase.

So, you may wonder why ClickHouse is so powerful and whether you should I include it in your analytic toolkit?

In this eBook, I am going to briefly introduce both systems from an architectural perspective, and provide a point-to-point functional comparison on considerations of OLAP systems, including data model, data ingestion, query, enterprise support, and ecosystem.

**ClickHouse is a trademark of ClickHouse, Inc.*

02

CHAPTER TWO

Architecture

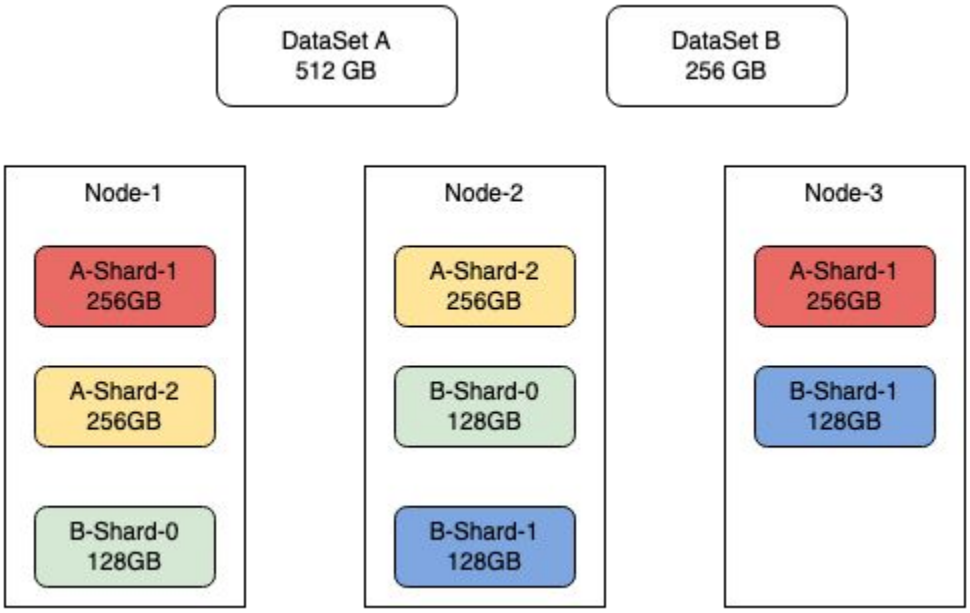
Architecture

Both ClickHouse and Elasticsearch adopt a distributed architecture aiming to support ultra-large data volume. Usually, the choice of architecture has a profound impact on the usage and extensibility of the system.

Interestingly, both ClickHouse and Elasticsearch adopt "sharding" and "replication" techniques to counter the challenges of large data volume while providing resilience from node failures.

To put it simply, Sharding is a process to split the large logical dataset into disjoint pieces (aka shard) such that every shard can be efficiently handled by a single node, which improves the performance via parallelism. Replication is to replicate shards among the nodes so that a single node failure would result in a loss of shard.

The diagram below demonstrates how two large datasets could be allocated to the sharding and replication architecture:



The difference between ClickHouse and Elasticsearch lies in their choice of how to manage the synchronisation of replicas, how to coordinate shards during query processing, and how to organize the shard to support their workloads.

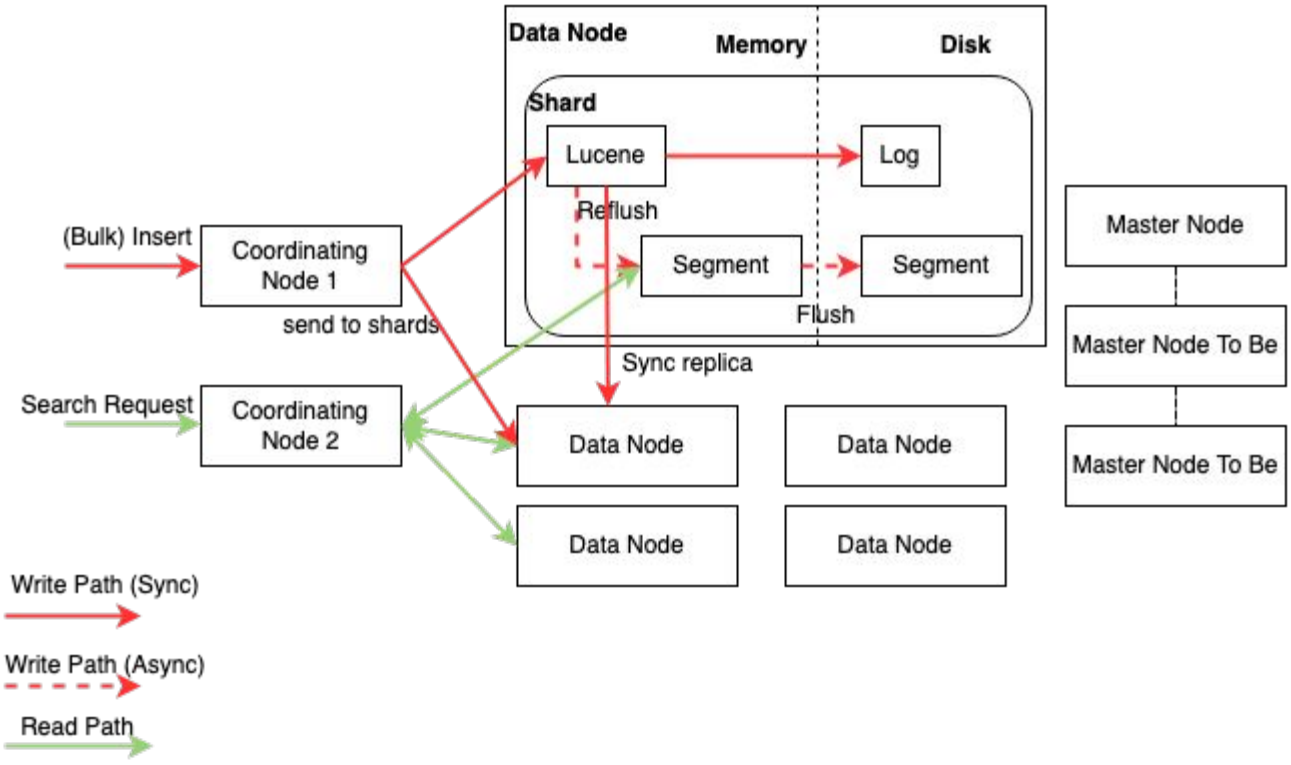
Elasticsearch

Elasticsearch is a distributed extension of Lucene, so its shards/replicas are organized as Lucene documents where the searching techniques like inverted-index and bloom-filter could be used to speed up the searching.

To counter the issue of shards synchronisation, Elasticsearch introduced three different nodes in its architecture:

- **Master node:** is responsible for coordinating cluster tasks like distributing shards across nodes, and creating and deleting indices. Each cluster will only have 1 dedicated master node, but with multiple master-eligible nodes for backup
- **Data node:** data node stores data in the form of shards and performs actions related to indexing, searching, and aggregating data.
- **Coordinating node:** coordinating node is neither master nor data, it acts as a load balancer that helps route indexing and search requests.

With this, the overall architecture of Elasticsearch is as follows:



In the chart above, the red lines indicate the write flow. During the write, the data would be sent to different data nodes according to its shard. Within a

shard, the Lucene engine will persist a redo log, and then update its replica for the write. Once the redo log is persisted, the write request is returned.

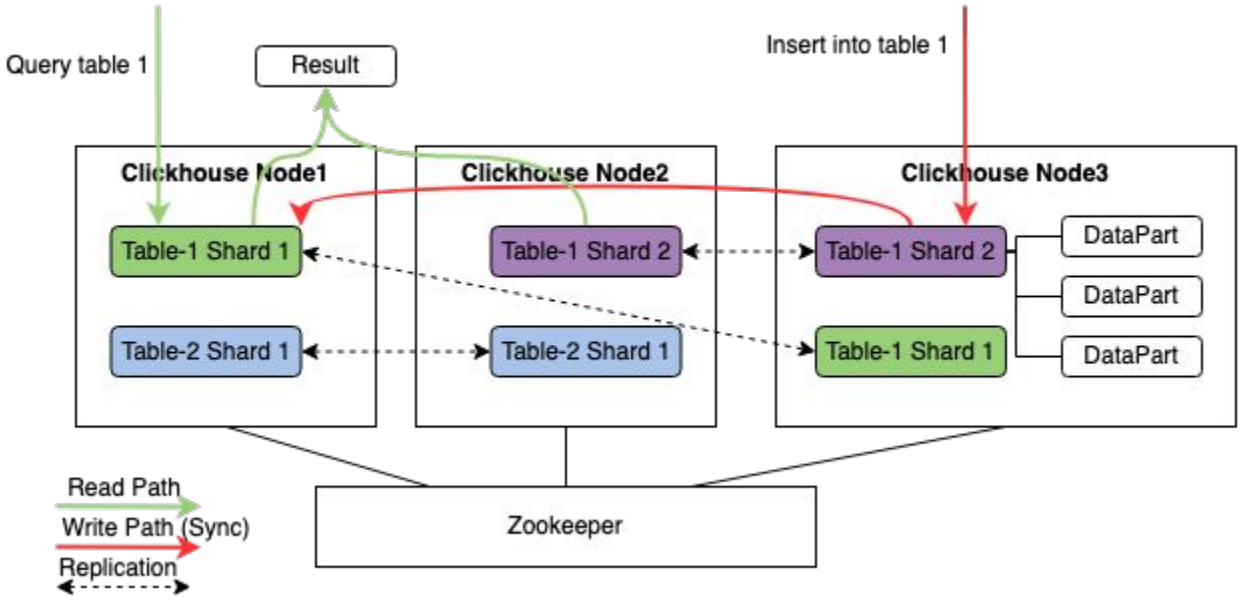
However, the newly persisted data would not be searchable until a reflush process converts them into a segment. The reflush frequency can be tuned by the client, but it is usually a trade-off between real-time and performance.

On the read path, Elasticsearch performs query-and-fetch pattern. There are a few variants of this pattern, but the core idea is the same. In this pattern, the read request would first be sent to a different shard where local processing is performed, and then the results from different shards are returned to the coordinating nodes and final results are returned.

There are a few caveats here: first the query and fetch stages are synchronous, and there is no pipeline, which is not efficient when returning results are large. Second, this query model fits best for top-N queries but is very limited to SQL-alike analytics which involves join and aggregations.

ClickHouse

ClickHouse is designed in a simpler way than Elasticsearch. In ClickHouse, the replication and synchronisation functionalities are offloaded into Zookeeper. As a result, there are no separate roles in ClickHouse. All nodes in ClickHouse are equal, and they handle different shards.



As shown in the above chart, when bulk data is inserted into ClickHouse, they are automatically sharded to different nodes based on their sharding key. The shards are then replicated across nodes via Zookeeper.

Since it is an active-active architecture, writing queries for a shard can be processed by any replica nodes. Within a shard, all new writes are flushed to the disk immediately and form a DataPart file.

ClickHouse uses its MergeTree engine to manage those data parts efficiently, including compression and compaction. Since the DataPart is append-only, the writing speed is very fast. In practice, it is easy to observe 200 MB/s write throughput per shard.

During the query phase, ClickHouse nodes fetch data from different shards to form the result. Although the flow is similar to Elasticsearch, there are a few

core differences.

First, ClickHouse supports pipelining, meaning the process of fetching data and computation is not entirely stage-by-stage. This gives much better performance even when the result set is large. Second, ClickHouse stores data in highly compressed columnar format, which is efficient for both disk storage and network transmission. Third, ClickHouse builds a vectorized engine to process columnar data efficiently.

03

CHAPTER THREE

Indexing and caching

Despite the closeness in architecture, the most prominent difference between ClickHouse and Elasticsearch is on Index.

Elasticsearch is famous for its full-text search functionality. It allows users to build primary and secondary indexes, where those indexes are globally ordered. With Lucene, Elasticsearch is able to provide full-text search and fuzzy search, which makes it efficient to answer search-by-keyword queries.

ClickHouse only provides a globally ordered primary key, and data in the same DataPart is sorted by the primary key. This can help with the range queries without employing additional storage space. However, due to the lack of secondary indexes, ClickHouse is not good at full-text search.

Another difference is caching.

In Elasticsearch, there are many places using cache, like Result Cache, Request Cache, Data Cache, and Index Cache. This is helpful in the full-text search scenario where there is likely hotspot data.

However, ClickHouse only employs disk page cache. This is because in OLAP cases, most queries are ad hoc, where optimizing the disk performance is more beneficial.

Now you get a rough idea of the best use cases of ClickHouse versus Elasticsearch. ClickHouse is suitable for range query and aggregation, with limited join ability while Elasticsearch is suitable for search (e.g., low selectivity in "where").

04

CHAPTER FOUR

Functionality

With different architectural choices, Elasticsearch and ClickHouse also have different features for their supported scenarios. Hereby, I would like to make a comparison on the features related to OLAP.

Data Model

Elasticsearch has the feature of Schemaless modeling. In this model, Elasticsearch can auto-infer the schema from the first piece of data. However, Elasticsearch is dependent on the schema in its Lucene storage. This is because it needs to know the data type to build primary and secondary indexes. As a consequence, when users wish to update the type of a field, the only way is to reindex.

On the contrary, ClickHouse does not couple data types with storage. Since its processing is based on scanning, it is able to cast data types during processing or update data types during DataPart compaction. This makes data type change much easier. Another advantage of ClickHouse is that it provides many useful data types for analytic purposes, such as LowCardinality, Geo, Interval, etc.

In most OLAP scenarios, auto-schema inference is not really useful. Users still need the correct type to gain the best performance from Elasticsearch. However, when there are data model changes, it would be a big headache for Elasticsearch to reindex its existing data.

Data Ingestion

There are typically two ways to load data to OLAP systems: batch loading and streaming loading. Due to fast disk append, ClickHouse can achieve a much faster bulk ingestion speed than Elasticsearch.

The MergeTree structure natively supports concurrent write, therefore when the volume is large, the user is able to increase write concurrency in ClickHouse to gain higher write throughput. On top of that, ClickHouse also natively supports real-time ingestion of Kafka via its built-in KafkaTableEngine.

In Elasticsearch, the write performance is bottlenecked by two parts: Lucence indexing and redo Log. There are also segment merging processes but it is usually configured as asynchronous. The redo log is not well designed for concurrent write. When the write volume is large, Elasticsearch takes a long time to ingest the data

Query Engine

As explained in the architectural comparison, Elasticsearch is based on Lucene and uses query-and-fetch model to process queries. In OLAP workloads, most queries require filters and aggregations. The intermediate result usually is large, and Elasticsearch's fetch-and-query model is very inefficient and prunes to the out-of-memory error.

On the other hand, ClickHouse is column-oriented, is based on vectorized processing engine, and supports pipelining between stages. These are tailored for range scans, which delivers good performance for OLAP workloads.

Query Language

Elasticsearch originally provides DSL to query its API and recently provides an SQL-like interface. However, in order to reach good performance, users still need to interact with the native API. This is not friendly to analytic users where most of them are SQL-trained.

ClickHouse, since it is a DMBS, supports SQL out of the box. Users can use most ANSI-style SQL to query ClickHouse without any trouble, and there are also highly optimized built-in functions to simplify analytics tasks. The learning curve for business analysts is also much flatter.

Concurrency

Elasticsearch is seemingly better at concurrency than ClickHouse. With its multi-level caching, Elasticsearch is usually able to serve queries from its indexes, which is very efficient. However, when the query involves aggregation, Elasticsearch performance will drop significantly.

ClickHouse looks at the concurrency queries separately. It aims to shorten

the single-query-latency by maximal using the available hardware. Therefore, when multiple queries arise, there are often not enough resources to process those queries. However, there are ways to control the max resources that can be used by every query, and with tuning, ClickHouse can still reach a good concurrency performance. However, the resource segregation on ClickHouse is a very challenging job, but at ByteHouse, we have made this feature out-of-box.

05

CHAPTER FIVE

Enterprise Support

Access control

Both Elasticsearch and ClickHouse support TLS and RBAC. The difference lies in granularity. Elasticsearch provides index level role-based access control, but not on the row or column level, while ClickHouse provides DBSM-like role-based access control, where data owners can define row or column level access policies.

ClickHouse also provides SQL commands to manage those policies, which makes it easy for traditional DBAs. On the contrary, Elasticsearch requires managing the policies of the web console. A plus is that Elasticsearch's access control policies are shared with Kibnana and Logstash, where users only need to access one place to manage the policies for all three tools.

Data Security

Both Elasticsearch and ClickHouse support GDPR features including authentication, RBAC, Data Backup, encryption, and retention. Elasticsearch only supports fixed interval TTL like 10s or 1 month while ClickHouse is more flexible by allowing defining arbitrary expressions, like "biz_date + 30d". Other than GDPR compliant features, ClickHouse additionally provides row filtering and data masking features which allow users to shade row/column values according to different audiences.

ETL

Being analytic tools, neither Elasticsearch nor ClickHouse provides full-fledged ETL functionalities. In comparison, ClickHouse has materialized views that allow users to define some transformation logic. Most ETL tools such as Nifi, TalenD have ES and ClickHouse sink. A notable difference is that ClickHouse natively supports external tables where users can query Hive/MySQL data directly in ClickHouse without loading them. This is handy for some quick data experiments.

Visualisation

Elasticsearch is best packaged with Kibana to form the ELK solution. Kibana is a one-stop data visualisation and exploration tool, which allows users to craft various dashboards.

Other than Kibana, Elasticsearch also supports third-party BI tools such as Tableau and Superset. ClickHouse, on the other hand, does not provide any built-in solution for visualization. It provides JDBC-drivers that are compatible with most BI tools. Now most major BI tools have ClickHouse support.

06

CHAPTER SIX

Conclusion

ClickHouse is more suitable for OLAP workload than Elasticsearch due to its ultra-performance design.

However, Elasticsearch is still recommended for full-text search, especially to bundle with Logstash and Kibana to provide log analysis features.

ClickHouse, on the other hand, is capable of handling complex OLAP queries with a flat learning curve and providing good connectivity upstream and downstream.

Reimagine ClickHouse

On top of the new disaggregated compute and storage architecture, cloud-native ClickHouse can utilise resources more efficient handling large volume of data and significantly improve the performance and reduce the infrastructure cost.

This eBook shares some of the architecture design behind ByteHouse Cloud Platform.

Get A Free Copy

ByteHouse.cloud

ByteHouse