GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

HPS

Seminar Report

Influx DB vs Elasticsearch: A comparative study

Sunny Jain
Matr.Nr: 21435344

Supervisor: Lars Quentin

Georg-August-Universität Göttingen
March 13, 2024

Abstract

The capacity to effectively handle and analyze large volumes of data is essential in today's data-driven environment. InfluxDB and Elasticsearch are two of the many data storage and retrieval options that are accessible; they are both strong competitors with special advantages and functionalities suited to different use cases.

This comparative analysis examines the performance of Elasticsearch and InfluxDB across three key metrics: insertion speed, storage efficiency, and query response time. InfluxDB demonstrates superior response times and storage efficiency for inserting 50k records, while Elasticsearch exhibits higher throughput. However, Elasticsearch shines in query response time, making it ideal for applications demanding low-latency data retrieval and efficient processing. Conversely, InfluxDB excels in high-volume data ingestion and storage, catering well to time-series data applications.

Statement on the usage of ChatGPT and similar tools in the context of examinations.

In this work I have used ChatGPT or a similar AI system as follows: In brainstorming and to create individual passages, altogether to the extent of 0% of the whole text. I assure you that I have stated all uses in full.

Contents

1. Introduction

1.1 Motivation

One of the most important questions to figure out here is which one to choose when selecting InfluxDB or Elasticsearch for time-series data. InfluxData, the company of InfluxDB, had already done some comparisons. They have the benchmark results for InfluxDB v1.8.0 and Elasticsearch v7.8.0 while their analysis is very insightful, they have the financial incentive to do so [1].

At the GWDG, Grafana is utilized alongside InfluxDB, supplanting Elasticsearch within its technology stack. This shift suggests a preference for Grafana and InfluxDB combination over Elasticsearch for their specific requirements [2]. Elasticsearch, initially open source, altered its licensing due to cloud provider competition. AWS Apache open-sourced the Elasticsearch version under the original license and forked to the new OpenSearch project. Grafana is a Kibana fork that focuses on metrics for observability [3].

There are three measure values in which we can compare the performance of both tools. These are as follows: Data ingest performance – measured in values per second, On-disk storage requirements – measured in Bytes, and Mean query response time – measured in milliseconds as done in this study [4].

Elasticsearch performs its full-text search using a library called Apache Lucene, with the indices for the search stored by a utility called mmapfs. This method uses a technique called memory mapping to speed up file access. However, each time mmapfs creates a memory map, the operating system has to keep track of it which by default is limited to 65530 memory maps per process. For the normal system, this limit is sufficient, but Elasticsearch, which relies heavily on mapping for its search operations, needs more, thus creating the need to increase this limit.

This is a kernel setting and changing it requires root privileges [5]. Allowing all users this level of access to the system could lead to major security issues. Therefore, without this authority to increase the available memory maps, one may face difficulties in using Elasticsearch on HPC systems or conducting optimizations through extensive benchmarks.

1.2 InfluxDB

InfluxDB is a powerful open-source time-series database designed to handle high write and query loads for time-stamped data. It excels in storing, querying, and visualizing metrics and event data in real-time. With its schema-less design and SQL-like query language, InfluxDB simplifies the process of collecting, analyzing, and acting upon time-series data. Its efficient indexing and compression techniques make it ideal for IoT sensor data, monitoring systems, and DevOps metrics. InfluxDB's flexibility and scalability, coupled with integrations with various

data visualization and analytics tools, make it a popular choice for organizations seeking to gain insights from their time-series data streams.

1.3 Elasticsearch

Elasticsearch is a distributed, RESTful search and analytics engine built on top of Apache Lucene, capable of handling large-scale, real-time data retrieval and analysis. While not specifically tailored for time-series data, Elasticsearch is often utilized for such data due to its efficient indexing and querying capabilities. With the help of plugins like the Elastic Time Series plugin, Elasticsearch can efficiently manage time-series data by indexing timestamps and values, enabling fast search and aggregation operations. Its distributed architecture allows for horizontal scaling, making it suitable for handling high volumes of time-series data across multiple nodes. Elasticsearch's integration with Kibana further enhances its utility by providing powerful visualization and dashboarding capabilities, making it a popular choice for monitoring, log analysis, and other time-series data use cases.

2. Setup and Conditions

We have employed JMeter, a widely used performance testing tool, to assess the effectiveness and performance of both Elasticsearch and InfluxDB. Both of these databases, Elasticsearch and InfluxDB, have been configured within Docker containers, facilitating an isolated and reproducible testing environment. The environment in which all the performance testing is as follows :

Windows-11; RAM-16GBs; Intel i7-12700H 12th Gen
X-Influxdb-Build: OSS
X-Influxdb-Version: v2.7.5
Elasticsearch-Version: v8.12.0
Docker version 24.0.7, build afdd53b

2.1 Setup of ElasticSearch:

Docker networks provide isolated environments for containers to communicate with each other since we are running 3 containers es01-1, es02-1, es03-1, and kibana.

1. `Create a new docker network.`
   `docker network creates elastic`
2. `Pull the Elasticsearch Docker image.`
   `docker pull docker.elastic.co/elasticsearch/elasticsearch:8.12.2`

3. Start an Elasticsearch container.
   docker run --name es01 --net elastic -p 9200:9200 -it
   docker.elastic.co/elasticsearch/elasticsearch:8.12.2

2.2 Setup of InfluxDB:

It launches InfluxDB v2 in a Docker container on port 8086, ensuring data and configuration persistence with specified volumes.

```
docker run --name influxdb2 --publish 8086:8086 --mount
type=volume,source=influxdb2-data,target=/var/lib/influxdb2 --mount
type=volume,source=influxdb2-config,target=/etc/influxdb2 --env
DOCKER_INFLUXDB_INIT_MODE=setup --env
DOCKER_INFLUXDB_INIT_USERNAME=<USERNAME> --env
DOCKER_INFLUXDB_INIT_PASSWORD=<PASSWORD> --env
DOCKER_INFLUXDB_INIT_ORG=<ORG_NAME> --env
DOCKER_INFLUXDB_INIT_BUCKET=<BUCKET_NAME> influxdb:2
```

2.3 Type of data being inserted:

Time-series data refers to a type of data where each data point is associated with a timestamp, typically representing when the data was recorded or observed. This data is often collected at regular intervals over time, making it well-suited for analyzing trends, patterns, and changes over temporal dimensions.

For time-series data analysis, Elasticsearch can be used to index and store time-stamped documents efficiently. Its flexible querying capabilities allow for complex searches, aggregations, and filtering based on various criteria such as time ranges, metadata, and values.

For Elasticsearch the data model being inserted is as follows with random values for timestamp and value :

```
{"timestamp": "2020-07-12T15:23:45","value":57392}
{"timestamp": "2022-10-18T20:45:15","value":88475}
{"timestamp": "2023-06-30T12:38:00","value":21004}
{"timestamp": "2024-01-19T05:19:10","value":69541}
```

InfluxDB is purpose-built for handling time-series data, offering optimized storage and retrieval mechanisms tailored specifically for this use case. For InfluxDB the data model being inserted is as follows:

```
timeseries_data value=57392 1647262814993578123
timeseries_data value=88475 1643597258471294876
timeseries_data value=21004 1648419387210345689
timeseries_data value=69541 1646800259032013456
timeseries_data value=44762 1645678934567123456
```

The first column (value) represents a randomly generated numerical value between 1 and 100,000, The timestamp provided, 1647262814993578123, represents the number of nanoseconds elapsed since the Unix epoch time.

1647262814993578123 nanoseconds = 1647262814 seconds
January 1, 1970 + 1647262814 seconds = February 13, 2022

Here the timeseries_data is the measurement for the influxDB with the value being the randomly generated number and the last argument is the timestamp.

3. Ingestion Efficiency:

Here we are comparing the insertion of records for the two datastores: Elasticsearch and InfluxDB. We have set up influxDB and Elasticsearch in Jmeter and we are inserting 50k records one by one. So that means the insertion request is performed 50,000 times with single users.

3.1 On comparing the insertion of 50k records -

For Elasticsearch:

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	50000	0	0.00%	5.82	3	157	5.00	8.00	12.00	32.00	159.81	17.95	60.85
HTTP Request	50000	0	0.00%	5.82	3	157	5.00	8.00	12.00	32.00	159.81	17.95	60.85

For InfluxDB :

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**50000**	**0**	**0.00%**	**9.73**	**5**	**229**	**7.00**	**12.00**	**18.00**	**44.00**	**101.10**	**33.47**	**32.30**
HTTP Request	50000	0	0.00%	9.73	5	229	7.00	12.00	18.00	44.00	101.10	33.47	32.30

In terms of Response time: InfluxDB has a lower average response time (5.82 milliseconds) compared to Elasticsearch (9.73 milliseconds). InfluxDB also has a narrower response time range (3 milliseconds to 157 milliseconds) compared to Elasticsearch (5 milliseconds to 229 milliseconds). The median response time for InfluxDB is 5 milliseconds, while for Elasticsearch, it's 7 milliseconds.

In terms of Percentile Response time: InfluxDB generally outperforms Elasticsearch in terms of percentile response times. For example, the 90th, 95th, and 99th percentile response times are lower in InfluxDB compared to Elasticsearch.

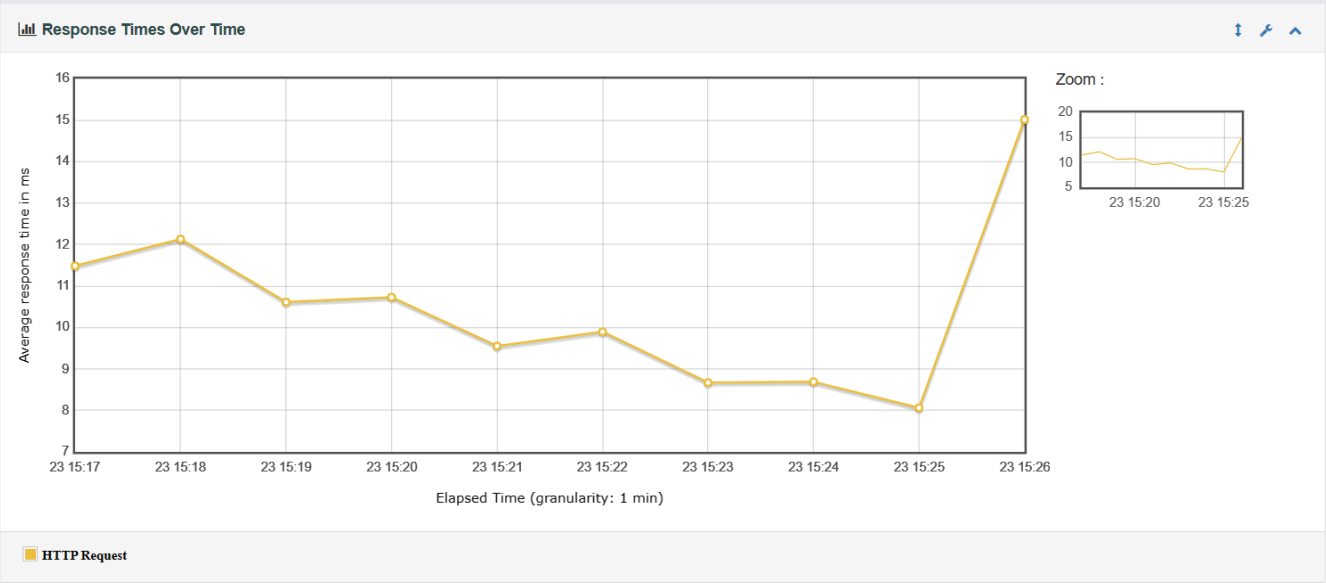
In terms of throughput: In terms of throughput, the sent throughput specifically refers to the rate of successful transmission of data points rather than the acknowledgment responses themselves. Both systems have relatively high throughput rates, but Elasticsearch has a slightly higher throughput in terms of transactions per second (101.10) compared to InfluxDB (159.81). Elasticsearch has a higher received throughput (33.47 KB/sec) compared to InfluxDB (17.95 KB/sec), but InfluxDB has a higher sent throughput (60.85 KB/sec) compared to Elasticsearch (32.30 KB/sec).

Conclusion: InfluxDB shows better response times and lower percentile response times, while Elasticsearch demonstrates slightly higher throughput in terms of transactions per second.

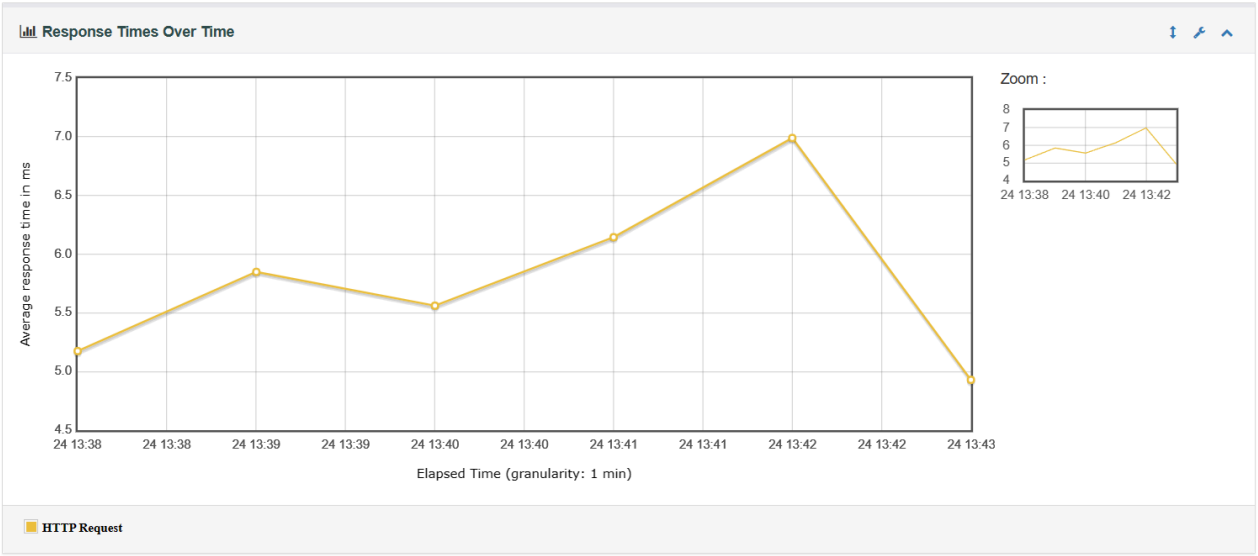
3.2 Ingestion: Response Times over Time

Response time refers to the duration it takes for a system to respond to a request or query. It is assessed starting from the instant a request is initiated up until the point when the requester receives a comprehensive response.

Elasticsearch: In the Elasticsearch, the response time is in the range of 7-16 ms. With the average response time being 9.73 ms. It took a total of 9 minutes from 15:17 to 15:26.



InfluxDB: In the InfluxDB, the response time is in the range of 4.5ms - 7.5ms. With the average response time being 5.82 ms. It took a total of 5 minutes from 13:38 to 13:43.

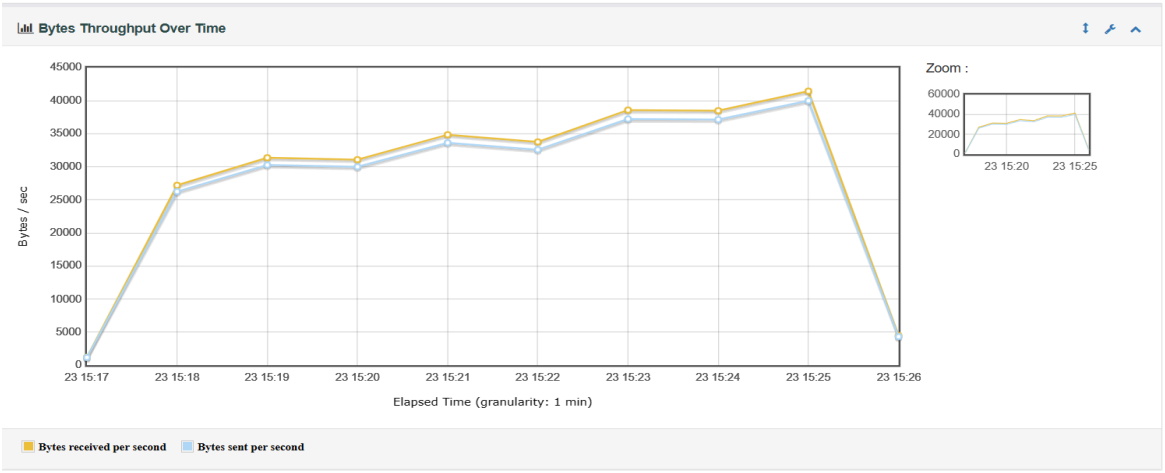


Conclusion: InfluxDB exhibits faster response times compared to Elasticsearch during ingestion, with an average response time of 5.82ms over 9 minutes, while Elasticsearch averages 9.73ms over 5 minutes.

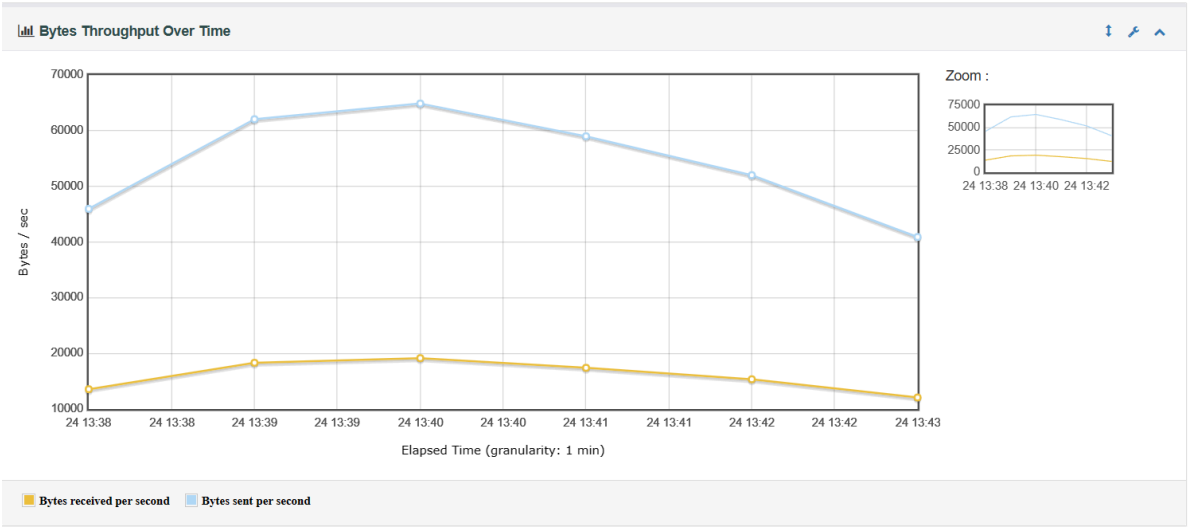
3.3 Ingestion: Bytes Throughput over Time

Byte throughput over time refers to the rate at which bytes of data are transferred or processed within a specific period. It is a measure of the amount of data that is transmitted, received, or processed within a given timeframe.

Elasticsearch: In Elasticsearch, the bytes/sec is between 5000-45000 bytes in 9 minutes i.e. from 15:17 to 15:26. Here the number of bytes received and the amount of bytes sent is almost equal.



InfluxDB: In InfluxDB, the bytes/sec is between 10000-70000 bytes in 5 minutes. Here the amount of bytes received is on the lower side than the amount of bytes sent.



4. Storage Efficiency Comparison:

Elasticsearch:

We have a total of 4 docker containers The **kibana-01** container in Elasticsearch is mainly responsible for the Dashboard and visualization part. The containers: **es01-1, es02-1, and es03-1** are used here for the storage of data and as usual functionality of elastic search.

Before inserting 50k records:

- es01-1

First, we have to navigate to the elastic search-es0-1 container, and from there we will go to the data. The current data to be calculated is present in the “data” folder in the es01 container and is **4.8MB**

- es02-1

A similar process followed for es02-1, the value for “data” is **3.1MB**.

- es03-1

Now, we are going to follow the same process for es03-1. The “data” value is **5.3MB**.

After Inserting 50k records:

- es01-1

The size of data in the container es01-1: **12MB**

- es02-1

The size of data in the container es02-1: **3.3MB**

- es03-1

The size of data in the container es03-1: **5.3MB**

Comparing 50k records with size:

- Size of singular record: 208 bytes.
- Since we are inserting **50,000 records**, it should be 50,000 multiplied by **208 bytes** which is equivalent to 10,400,000 bytes. If we convert it into megabytes then 10,400,000 bytes / 1,048,576 bytes/MB ≈ **9.918 megabytes**.

ELK Container	Initial Size	Size after inserting 50k records	Delta of records
es01-1	4.8MB	12MB	7.20MB
es02-1	3.1MB	3.3MB	0.20MB

es03-1	5.3MB	5.3MB	0.0MB
		Total Size of records	7.40M

As you can see theoretically the data present in the ElasticSearch datastore should be 9.91M but due to compression, it got reduced to 7.40M. The reason for compression and more information are below.

Compression:
The default compression mechanism employed is LZ4, offering a balance between compression efficiency and computational overhead. However, alternative options such as DEFLATE are available, offering higher compression ratios at the expense of increased CPU utilization for compression and marginally slower indexing rates [6].

Influx-DB:

The data is usually stored in /var/lib/influxdb2.

Before inserting 50k records in the influx db the size was 1.8MB
After inserting 50k records in the influx db the size is 4.6 MB

The delta of 50k records inserted before and after is: 4.6M - 1.8M = 2.8M

Size of a single record is 2.8 MB divided by 50,000 records which is equivalent to 60 Bytes.

Conclusion: The size of a single record is 208 bytes in Elasticsearch and the size of a single record in influx db is 60 bytes. The compression and storage efficiency of Influx DB is better than Elasticsearch.

5. Query Response Time:

We have set up JMeter for InfluxDB and Elasticsearch and we are trying to query 50K records.

Search Query for Elasticsearch:

```
{
 "query": {
   "range": {
     "value": {
       "gte": 30,
       "lte": 40
     }
   }
```

```
  }
}
```

Search Query for InfluxDB:

```
from(bucket: "testData")
  |> range(start: -100y)
  |> filter(fn: (r) => r["_measurement"] == "timeseries_data")
  |> filter(fn: (r) => r["_field"] == "value" and r["_value"] >= 40000
and r["_value"] <= 60000)
```

Observation:
We tried inserting more than 400 records for InfluxDB. It started giving Java. lang.OutOfMemoryError but this did not happen in the case of Elasticsearch. The possible solution was to increase the Heap Memory of the Jmeter. Once the size of the heap was increased to 2GBs. We were able to add 50K records.

5.1 On querying 50K records:

Elasticsearch:

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**50000**	**0**	**0.00%**	**2.25**	**1**	**1856**	**2.00**	**2.00**	**3.00**	**9.00**	**422.08**	**766.67**	**159.93**
HTTP Request	50000	0	0.00%	2.25	1	1856	2.00	2.00	3.00	9.00	422.08	766.67	159.93

InfluxDB:

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**50000**	**0**	**0.00%**	**33.40**	**24**	**316**	**30.00**	**42.00**	**54.00**	**108.00**	**29.47**	**38302.20**	**15.08**
HTTP Request	50000	0	0.00%	33.40	24	316	30.00	42.00	54.00	108.00	29.47	38302.20	15.08

In terms of Average Response Time: Elasticsearch demonstrates significantly lower average response times (2.25 milliseconds) compared to InfluxDB (33.40 milliseconds). This indicates

that **Elasticsearch processes requests more swiftly, resulting in a better user experience and faster data retrieval**.

In terms of Percentile Response Times: Elasticsearch consistently outperforms InfluxDB across all percentile response times, showcasing superior responsiveness. For example, at the 95th percentile, **Elasticsearch responds in 3.00 milliseconds compared to InfluxDB's 54.00 milliseconds.**

In terms of Throughput: **Elasticsearch exhibits substantially higher throughput (422.08 transactions per second) compared to InfluxDB (29.47 transactions per second).** This indicates Elasticsearch's ability to handle a larger volume of requests within the same time frame, making it more suitable for high-throughput applications.

In terms of Data Throughput via Network: **InfluxDB demonstrates higher received throughput (38,302.20 KB/sec) compared to Elasticsearch (766.67 KB/sec).** However, Elasticsearch exhibits higher sent throughput (159.93 KB/sec) compared to InfluxDB (15.08 KB/sec). This suggests that while InfluxDB may handle incoming data at a higher rate, Elasticsearch efficiently processes and sends data back to clients.

Conclusion:

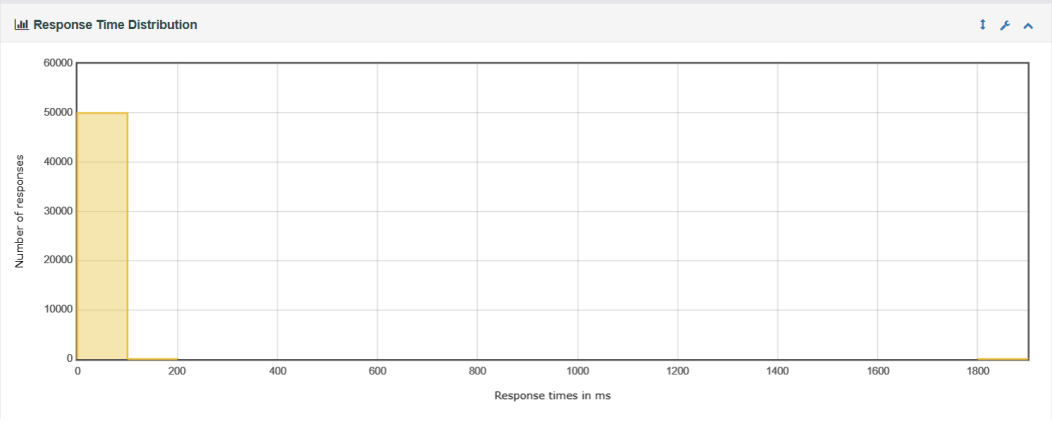
Elasticsearch emerges as the preferred choice for applications requiring low-latency data retrieval, high throughput, and efficient processing of requests. Its superior performance across various metrics makes it well-suited for use cases demanding real-time analytics, logging, and search functionalities.

In contrast, InfluxDB may be more suitable for scenarios prioritizing high-volume data ingestion and storage, especially in time-series data applications. However, its comparatively higher response times and lower throughput may impact applications requiring rapid query processing and responsiveness.

5.2 Query: Response Time Distribution:

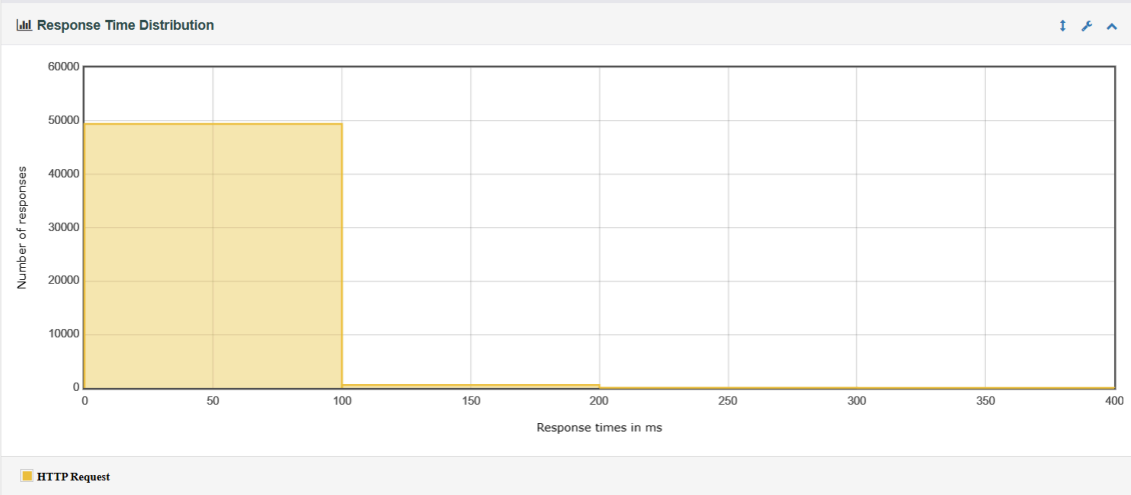
- **Elasticsearch**

Response Time Distribution for Elasticsearch with 49,993 responses for HTTP requests being between 0-100ms, 5 responses being in 100-200 ms, and 2 responses being in 1800-2000ms.



- **InfluxDB**

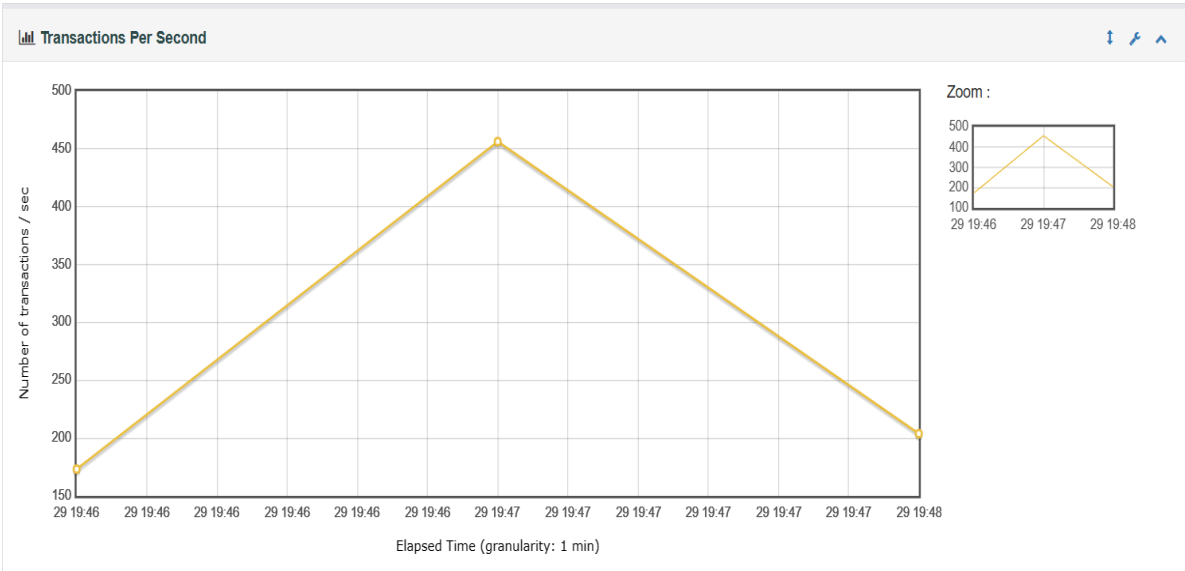
Response Time Distribution for InfluxDB with 49,447 responses for HTTP requests being in between 0-100ms, 542 responses in between 100-200ms, 6 responses in between 200-300 ms, and 5 responses in 300-400 ms.



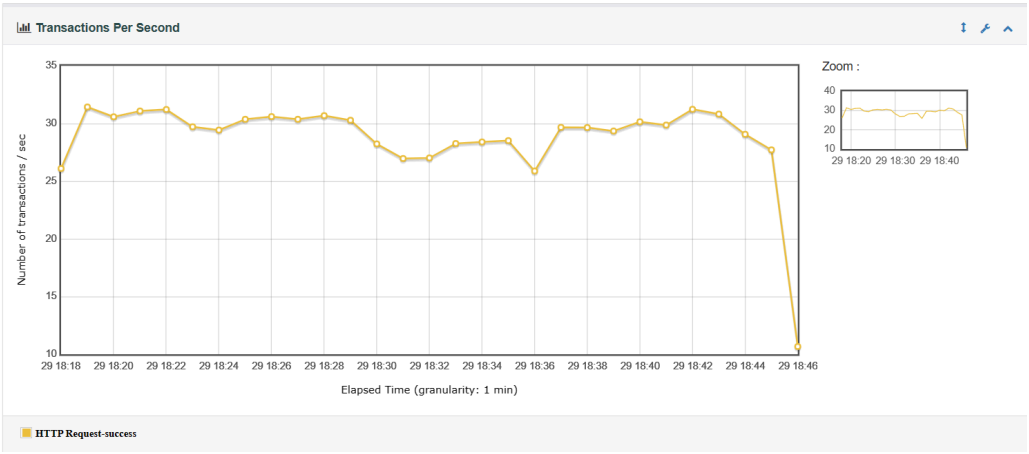
Conclusion: Elasticsearch appears to offer more reliable and faster query responses for HTTP requests compared to InfluxDB, making it potentially more suitable for applications requiring low-latency data retrieval.

5.3 Query: Transaction Per Seconds:

- **Elasticsearch**: Initially, the rate of successful transactions begins at 173 transactions per second eventually peaks at 456 transactions per second, and finally ends at 204 transactions per second. It takes close to 2 minutes from 19:46 to 19:48 to insert the records.



- **InfluxDB:** Initially, the rate of successful transactions is at 26 transactions per second. It keeps oscillating between 26 transactions per second and 31 transactions per second. In the end, it drops to 11 transactions per second. It takes a total of 28 minutes from 18:18 to 18:46.

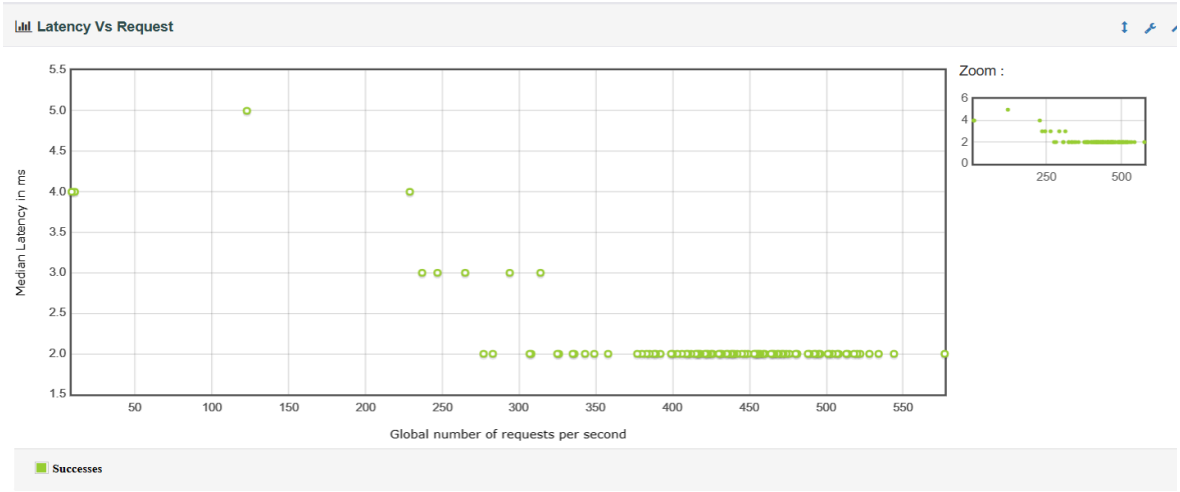


Conclusion:

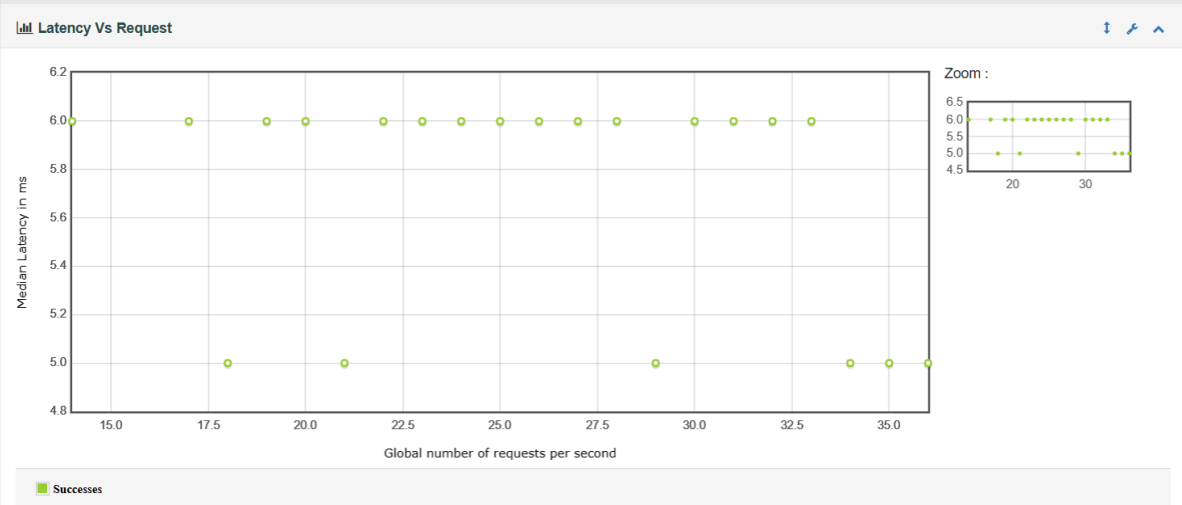
Elasticsearch excels in rapid, stable transaction processing for dynamic workloads and high scalability needs. InfluxDB suits applications with lower transaction volumes and less strict performance demands.

5.4 Query: Latency Vs Request

- Elasticsearch: The most amount of time it took was 5.0 ms for 123 requests per second. The minimum amount of time it took was 577 requests per second at 2.0 ms. The median request per second was 422 requests per second.



- InfluxDB: The most amount of time with the minimum number of requests was 6.0 ms for 14 requests per second. The maximum number of requests was 5.0 ms for 36 requests per second. The median request per second was 29.47 requests per second.



Conclusion: In Elasticsearch, the latency varied between 2.0 ms and 5.0 ms, with request rates ranging from 577 to 123 per second, while InfluxDB showed latency between 6.0 ms and 5.0 ms, with request rates spanning from 14 to 36 per second.

6. Challenges

InfluxDB is time-series data and stores data in a columnar format optimized for time-based queries. Elasticsearch is a document-oriented database where each document can have its schema. Mapping time series data effectively to fit each database's model can be challenging. Other processes running on the benchmarking machine can introduce noise, such as system daemons, background tasks, or other applications consuming system resources. CPU-intensive processes running concurrently with benchmarking can impact CPU utilization and introduce variability in query response times. Memory usage by other applications or system processes can affect available memory for caching and buffering, impacting the performance of both databases.

7. Conclusion

In conclusion, the comparative analysis between InfluxDB and Elasticsearch reveals distinct strengths and weaknesses in various performance metrics. InfluxDB demonstrates superior response times and lower percentile response times, making it an optimal choice for applications prioritizing quick data retrieval and data ingestion. Moreover, its storage efficiency,

evidenced by smaller record sizes and better compression, presents an advantage for managing large datasets effectively.

On the other hand, Elasticsearch exhibits faster query response times across all metrics, indicating its proficiency in processing requests swiftly and ensuring a seamless user experience. Additionally, Elasticsearch showcases substantially higher throughput, making it more suitable for high-volume applications requiring rapid data processing and retrieval. Its efficient handling of network data throughput further enhances its suitability for real-time data analytics and streaming applications.

Ultimately, the choice between InfluxDB and Elasticsearch should be made based on the specific requirements and priorities of the application. For applications emphasizing quick data retrieval, efficient storage, and lower response times, InfluxDB presents a compelling option. Conversely, for high-throughput applications demanding fast query processing and superior network data throughput, Elasticsearch emerges as the preferred solution.

8. References:

[1] InfluxDB: Open Source Time Series Database | InfluxData. (2021, December 10). InfluxData.https://www.influxdata.com/blog/influxdb-markedly-elasticsearch-in-time-series-data-metrics-benchmark/

[2] Labs, G. (2013). grafana. GitHub repository. Retrieved from https://github.com/grafana/grafana

[3] (n.d.). What is ELK stack? - Elasticsearch, Logstash, Kibana Stack Explained - AWS. Retrieved from https://aws.amazon.com/what-is/elk-stack/

[4] Palmer, B. (2014). dinghy. GitHub repository. Retrieved from https://github.com/codekitchen/dinghy

[5] (n.d.). Store | Elasticsearch Guide [8.13] | Elastic. Retrieved from https://www.elastic.co/guide/en/elasticsearch/reference/current/index-modules-store.html

*[6] (n.d.). Does Elasticsearch automatically compress data? - Elasticsearch - Discuss the Elastic Stack. Retrieved from https://discuss.elastic.co/t/does-elasticsearch-automatically-compress-data/9316*1

Code: (n.d.). Retrieved from https://github.com/sunnyjain15699/influxDB-vs-elasticSearch: Comparison analysis of elastic search and influxDB (github.com)