Sentiment analysis and time series prediction on Twitter data

Alex Santini

MSc in Data Analytics

Integrated assignment big data processing and Advanced data analytics

CCT

28.10.2023

Sentiment analysis and time series prediction on Twitter data

Introduction

*SQL and NoSQL databases comparison*

*When it comes to storing SQL and NoSQL databases are the two main categories a developer needs to know for choosing the right tool for the right job; especially in the case of managing text data the two architectures have their strengths and weaknesses, In order to test those a testing strategies have been developed through the use of a familiar benchmarking tool called YCSB on a Linux virtual machine.*

*This comparison aims to explore the performances of a SQL database (Mysql) and a Nosql database (MongoDB) when taking a read-heavy and a insert-heavy load.*

*The databases in questions will have to contain the same tabular format to equally compare them, the format chosen is the same present in the excel sheet provided for the rest of this analysis in the csv file (projet\_tweets.csv) with the schema as follow:*

|  |  |  |
| --- | --- | --- |
| **Field** | **type** | **Example** |
| id | INT | 4587 |
| date | DATE | Sat May 16 23:58:44 UTC 2009 |
| flag | STRING | NO\_QUERY |
| user | STRING | bobthebuilder |
| text | STRING | Lyx is cool |

*Fig.1 YCSB comparison schema*

It’s important to notice how the schema must be explicitly mentioned prior to loading the data in the MySQL database whilethe same is not a requirement for MongoDB showing the first noticeable difference between the two.

The testing strategy aim to compare a classic insert heavy load and a more generalist *read heavy load with the following parameters:*

|  |  |
| --- | --- |
| Read Heavy |  |
| Count | 10,000 |
| Read | 80% |
| Update | 5% |
| Insert | 1% |
| Read modify write % | 2% |
| Scan | 3% |
| Distribution | Zipfan |

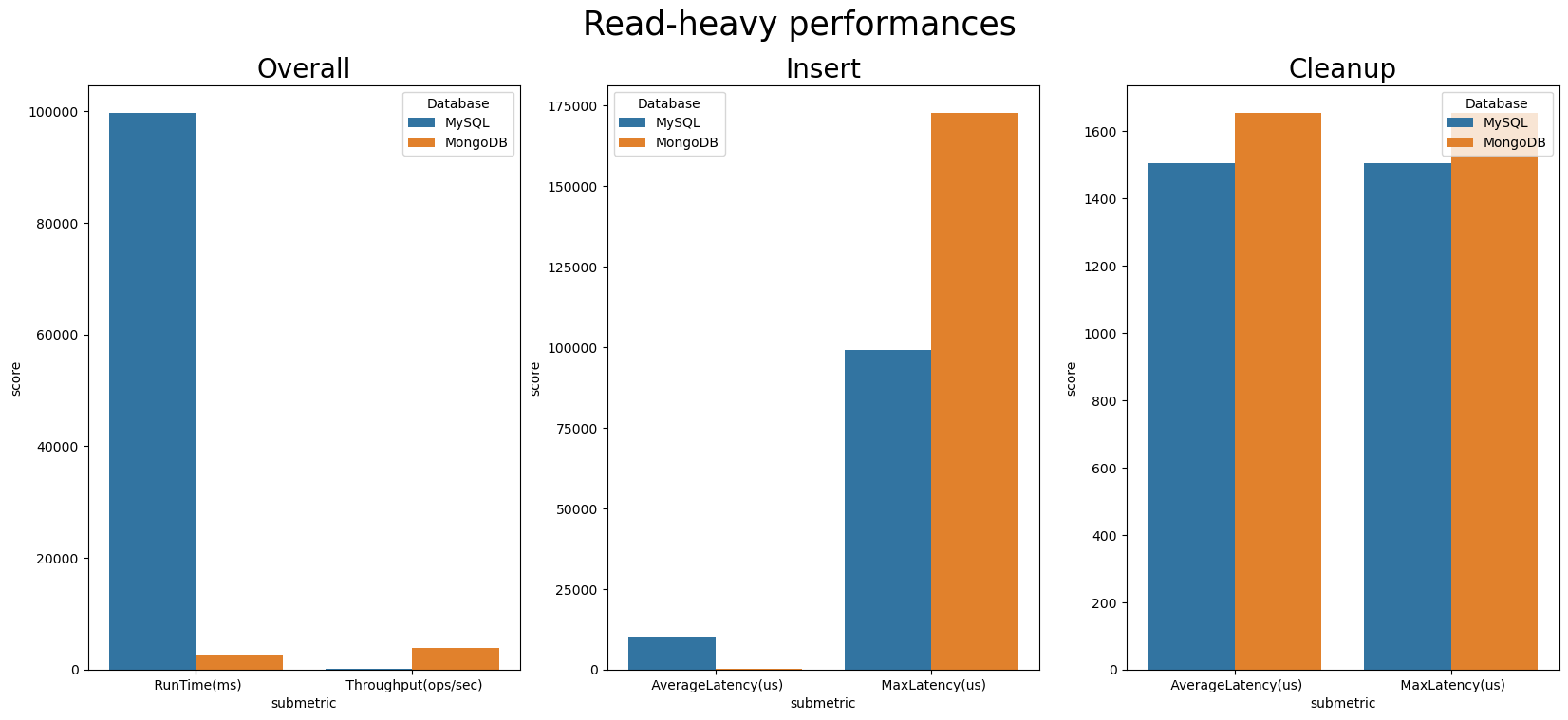
*Fig.2 Read heavy custom load parameters.*

In order to test the scalability of the two architectures a 80-20 insert-heavy load has been tested for different sizes as shown in Fig.3.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Count | 5,000 | 10,000 | 50,000 |
| Insert | 80% | 80% | 80% |
| Read | 20% | 20% | 20% |

*Fig.3 Insert heavy loads.*

By providing both read and insert comparison the test strategy aims to achieve a good comparison of SQL and NoSQL functions in a text heavy context.

As expected the results shown a clear difference between the performances of the two databases; the main metrics took in consideration for this approach were the overall runtime measured in milliseconds and throughput of the database measured in operations per second: this two metrics were chosen because considered good overall indicators of overall performances when it comes to choosing a database for a production pipeline, other metrics like maximum latency and average latency where taken also in consideration but as secondary point od reference*Fig.4 Read heavy load results*

The result for the read heavy workload are clear as MongoDB shows better performances running the load 20 times faster than the SQL counterpart, this is highly expected as being a document-based database MongoDB can handle large volume of read operations outputting a higher throughput than MySQL as shown in figure; MySQL however seems to perform slightly better in cleanup operations and in minimizing the maximum latency in insert jobs which in this case were only counting as 1% of the total workload.

Second step of the testing strategy was to compare three different workloads by increasing the number of total operations (5k,10k,50k) in order to test the scalability of an insert heavy architectures in SQL and NoSQL models.

The result once again proven how MongoDB is more robust in scalability especially in terms of runtime: MySQL seems to increate its runtime almost exponentially comparing a 5k workload with a 50k workload while MongoDB increasing just slightly its runtime, the same is true for the total throughput of the databases where MongoDB managed to increase it as needed for larger workloads while MySQL showing a performance plateau in fig.5; it’s also important to notice how even is increasing MongoDB performances seems to be affected by diminishing returns in throughput signaling how scalability might become an issue for NoSQL databases after a higher threshold.

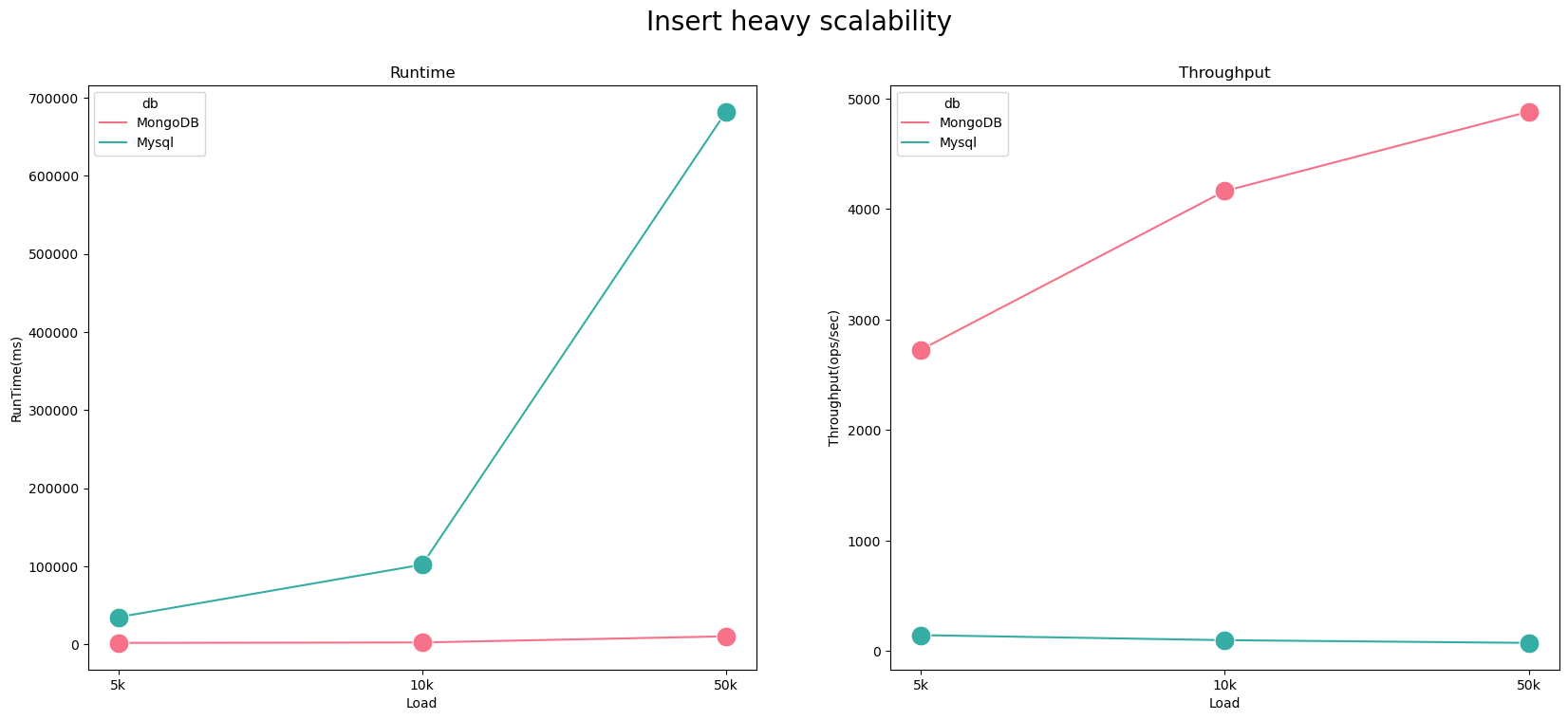


Fig.5 Insert heavy scalability load.

*Processing architecture*

After determining the performances of both SQL and NoSQL databases a conscious decision have been made to process the 1.5 million of data points required for the analysis; multiple databases and processing environments have been researched to come up with the best solution.

As per requirement the analysis needs two databases: one for storing raw data and another which can be used for extracting the final data for the time series analysis.

As result the raw data has been stored in a MySQL database: as shown in the first section MySQL is not the ideal choice for read operations, however raw data needs to be read only once, likewise the slow inserting speed can be surely limiting but again he data needs to be store only once not posing a big issue in terms of scalability and performances.

For storing post map-reduce data a Cassandra database has been deemed the most fitting as there might be a need for reading the data multiple times for the various steps in time series analysis, Cassandra is highly scalable and can produce high performances in heavy read tasks such as this one.

For processing environment Pyspark has been deemed the most idoneous especially because processing such large volume of data as Hadoop reduce job would have meant to not have access to the results of the data processing process in terminal, which for such large and variegated data can be problematic and difficult to handle.

Much focus in this research has been invested in finding the right tools in order to connect MySQL and Cassandra with the Pyspark environment: for connecting pyspark to MySQL it has been necessary to download a driver connector as jar file, while Cassandra connectors can be found online in the MVN repository for correct versions of Scala and Spark.

Main limitation for this approach can be found in the reliance of external connectors for the PySpark environment to read and to MySQL and Cassandra databases as different versions of spark and Scala might require different connectors.

As shown in figure raw data can be loaded into MySQL either directly through CLI commands or by using PySpark; second step of the architecture is to apply reduce processing and data processing for extracting the sentiment of each tweet and storing a daily average sentiment dataset into Cassandra where it can be extracted to a small csv file for proceeding with the time series analysis.

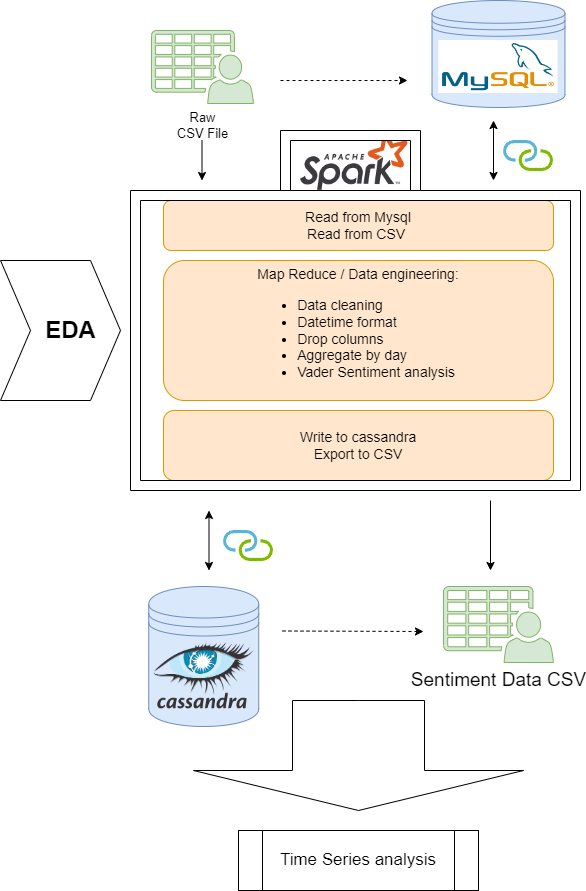


Fig.6 Processing architectures

Main advantage of this architecture are as follows:

1. Ability to read and write to csv file directly: if needed it’s possible to load the csv file as PySpark dataframe directly and to extract a final dataset for sentiment analysis as csc file bypassing both MySQL and Cassandra.
2. Once uploaded through Pyspark, CSV files (raw and final) can be extracted from the MySQL and Cassandra into csv files..
3. More secure data processing: to get from a raw dataset to a daily sentiment average dataset knowledge from Exploratory data analysis has been applied to data engineering carried out in Pyspark reducing the work needed in terms of Pyspark jobs.
4. More explainable output with examples in the code can be produced as Jupiter notebook compared to a reducer python file.

*Sentiment analysis*

*Time series prediction*

Reference List

*Seven steps for effective leadership development* (2012) Available at: http://www.oracle.com/us/media1/steps-effective-leadership-dev-1657106.pdf (Accessed: 24 November 2018).

Guenther, R. and Vittori, G. (2012) *Sustainable healthcare architecture*. New York: John Wiley & Sons.

Smith, V. and Jones, R. (2012) ‘Individual assignments and academic dishonesty: exploring the conundrum’, *The Educational Researcher*, 35(1), pp. 37–56.