A New Big Data Architecture for Efficient Processing of Spatiotemporal Data using Machine Learning

Sávio S. Teles de Oliveira[1], Vagner J. do Sacramento Rodrigues[1], Wellington S. Martins[1], Anderson da Silva Soares[1]

Federal University of Goiás (UFG)
Alameda Palmeiras, Quadra D, Câmpus Samambaia
131 - CEP 74001-970 - Goiânia - GO - Brazil

{savio.oliveira, vagner, wellington, anderson}@inf.ufg.br

Abstract. *The volume of spatiotemporal data is increasing in the era of Big Data and has brought several challenges in managing and processing these data. Many big data platforms have been created to address these challenges, but none can outperform the others in all scenarios on spatiotemporal queries. This paper presents a scalable architecture for efficient spatiotemporal data processing. This architecture allows a couple of several big data platforms and automatically choose on the fly, using machine learning, the best big data platform to process each spatiotemporal query.*

1. Introduction

Spatiotemporal data comprises spatial and temporal representations, and it has attracted great interest in the world, becoming critical in several areas such as health [Roy et al. 2021] and social networking [Parimala et al. 2021]. Large amounts of spatiotemporal data are being generated and captured through systems that record sequential observations of remote sensing, mobility, wearable devices, and social media. This large volume of data brings unique challenges in storing, managing, and processing these data, leading these challenges to the Big Spatiotemporal Data [Yang et al. 2020].

Several big data platforms have been proposed in the literature [Wang et al. 2020] to handle spatiotemporal data, such as Hadoop [Hadoop 2022], Spark [Spark 2022], Elasticsearch [Elasticsearch 2022] and SciDB [Paradigm4 2022]. Some papers [Zhang et al. 2018, Karim et al. 2018] show that no big data platform performs better than the others in all scenarios with spatiotemporal filters. It is up to the solution developer to choose in advance which platform they want to use for each scenario. This choice is a complicated task to be made dynamically because each big data platform has characteristics that allow it to perform best for specific spatial and temporal filtering scenarios. In addition, the platforms are constantly evolving and releasing new versions, and optimizations may emerge that overlap with competing platforms.

A new processing engine, SmarT, is presented in [de Oliveira et al. 2020, de Oliveira et al. 2021] for filtering and retrieving spatiotemporal data in big data using machine learning algorithms. SmarT seeks to choose, in real time, the best system to process the request. However, SmarT has some limitations: i) it is not extensible for any machine learning algorithm, ii) it doesn't have a complete and scalable architecture for managing and storing spatiotemporal data, iii) it cannot index the spatiotemporal data, and iv) it is not scalable on processing spatiotemporal queries.

Our work presents a new scalable architecture for efficiently processing of spatiotemporal data. Furthermore, it improves the SmarT with a new flexible architecture to be able to include new machine learning algorithms. Therefore, we evaluate many machine learning methods to determine which one we should choose to predict the big data platform to process each query. The architecture proposed in this work is an efficient, scalable, and robust solution for processing large volumes of spatiotemporal data.

The remainder of the paper is organized as follows. Section 2 describes the strategies found in the literature for processing of spatiotemporal data. Section 3 presents the details of our architecture, and Section 4 describes the experiment's methodology and results. Section 5 presents the conclusions of this work and a description of future works.

2. Related Works

There are many attempts to processing spatiotemporal data in database systems. Due to the high computational cost and the large volume of spatiotemporal data available, some works in big data [de Oliveira et al. 2019, Chi et al. 2016] introduced parallel and distributed solutions for processing spatiotemporal data. The MapReduce programming model has caused a revolution in the processing and management of spatiotemporal data because it has facilitated the development of works for processing spatiotemporal data [de Assis et al. 2017, Hamdi et al. 2022].

The Apache Hadoop [Hadoop 2022] and Apache Spark [Spark 2022] rapidly gained popularity among the scientific community due to the efficient spatiotemporal data processing. Some spatiotemporal extensions for Spark, like SpatialSpark [You et al. 2015], and Hadoop, like SpatialHadoop [Eldawy and Mokbel 2015], emerged to provide efficient spatial operations. Popular array databases like Rasdaman [Baumann et al. 1997] and SciDB [Brown 2010] show promising performance gains compared to traditional methods [Xu et al. 2020]. These databases have been used in several works [Lu et al. 2016, Camara et al. 2016] for efficient processing of spatiotemporal data. Beyond the SciDB, Elasticsearch is also recommended for spatiotemporal data processing [Shrivastava 2020].

Some works [Doan et al. 2016, Zhang et al. 2018] show that no big data platform is more efficient than the others in all spatiotemporal filter scenarios. In [de Oliveira et al. 2020, de Oliveira et al. 2021], the authors introduced a new query engine (SmarT) that chooses, in real time, the best big data platform to process the spatiotemporal queries. However, SmarT is not scalable and cannot handle highly concurrent queries. An efficient and scalable architecture for spatiotemporal data ingestion and processing becomes necessary for multiple big data platforms and simultaneous queries.

3. Exploring Big Data Architectures for Spatiotemporal Data Processing

In order to manage a large volume of spatiotemporal data, this work presents a new distributed architecture comprising an indexing and storing system, and a module for processing spatiotemporal queries. This architecture, shown in figure 1, can be coupled with any big data platform for processing spatiotemporal data, such as SciDB, Apache Spark, and Elasticsearch. Our architecture uses machine learning algorithms to choose, in real time, which big data platform will process each query. In other words, the developer does not need to select, in advance, which big data platform will process each type of query.

This architecture supports the insertion and indexing of new data (shown in section 3.1) and can also process queries using spatiotemporal filters (details in section 3.2). The architecture keeps a distributed message queuing to facilitate the communication and coordination between the processing jobs. The distributed queues can significantly improve performance, reliability, and scalability when processing spatiotemporal data.

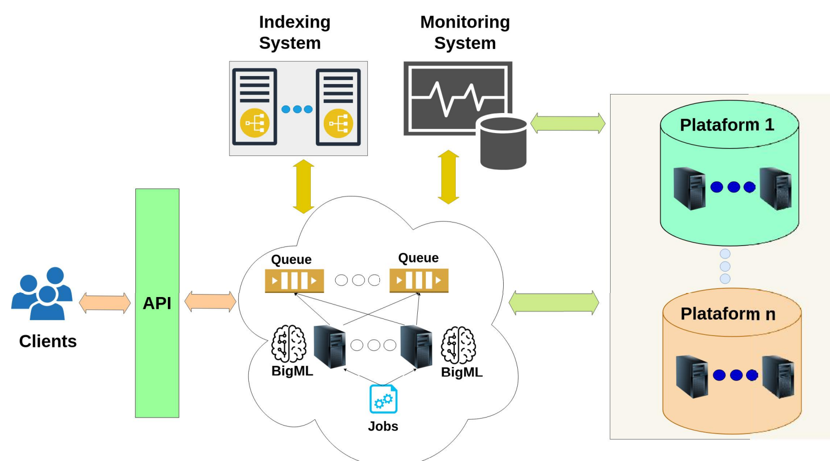


Figure 1. Architecture for efficient spatiotemporal data processing.

3.1. Inserting and Indexing Operations on Spatiotemporal Data

On insertion, the API has as input a set of spatiotemporal objects, where each object contains the following properties: i) geographic location in OGC Well-Known Text (WKT) format with a precision of at most 15 decimal places; ii) *timestamp* in UNIX format in seconds [1]; iii) object metadata in JSON format. With these three properties, it is possible to define each spatiotemporal object and also allow developers to add object metadata according to their business rules.

The architecture inserts the spatiotemporal data in a distributed way using *jobs* for data processing. Each job has a specific role in the data pipeline. The first job, $J1$, gets the data from the API and sends it to the queue. The second job, $J2$, gets the objects in the queue and sends each object's spatial location and timestamp to the Indexing System to index spatially and temporally. As seen in Figure 1, the Indexing System has an exclusive cluster to index these objects in a distributed way. Job $J2$ keeps a list of servers in the Indexing System cluster and sends each object to a server to be indexed. This server is chosen randomly from the list.

Each server has a BKD-Tree index (Block K-D tree) [Procopiuc et al. 2003] to keep the spatiotemporal data indexed efficiently. BKD-Tree is designed to be I/O efficient by keeping most of the data structures in disk blocks, while a small binary tree index structure is stored in RAM. Our work uses Elasticsearch as a distributed Indexing System to coordinate the local indexing of each BKD-Tree. Elasticsearch is a high-performing solution in indexing spatiotemporal [Wang et al. 2019] data.

[1] https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_16

Once indexed, the data pipeline moves on to the next step of storing the object in each big data platform. A message is published on a queue with the object's data. There are n jobs listening to this queue to store the spatiotemporal data across platforms, where n is the number of big data platforms. Each of these n jobs is responsible for storing the new data on a specific platform and knows how to build the object to store on that platform. This architecture facilitates the addition of new platforms, just creating a particular job for each new platform.

This flexible architecture allows us to easily add new steps to the data pipeline. That happens because the distributed queues keep a transparent communication between the producers and consumers in each pipeline step. In this architecture, we can have multiple jobs replicas consuming the data from the queues because the operations are independent, and insertions or indexing operations can be done in parallel. Therefore, the architecture is scalable and elastic due to the ability to add new data processing nodes simply by increasing the number of machines and replicas of the processing jobs.

3.2. Using Machine Learning for Efficient Spatiotemporal Query Processing

Our work proposes the BigML that aims to minimize the response time of spatiotemporal queries by choosing the big data platform with the lower response time predicted by the machine learning algorithm. Since query response time is a continuous variable, the problem of minimizing it is a task of approximating a mapping function f from input variables X to a continuous output variable y (query response time).

The client sends the spatiotemporal query parameters and gets the results through the API. One job S reads the query parameters from the API and becomes the query coordinator. So, this job will decide how to process the query, format the responses, and send the result to the client. Using the architecture proposed in figure 1, it is possible to scale the jobs quickly, which allows the processing of many queries in parallel. The job uses the BigML to choose which big data platform will process the query. BigML has the following input: (i) the spatial and temporal query constraints; ii) the cluster metrics acquired from the monitoring system, and iii) the approximate number of matches for the spatiotemporal query obtained in the indexing system. So, the job is responsible for collecting these inputs to send to BigML.

First, the job S has to get cluster metrics on the monitoring system. The job sends a request to the monitoring system that returns the current cluster metrics. Next, the job S obtains the approximate count of matches for the query in the indexing system. The job then queries the BigML that returns which big data platform must process the query. Afterward, S format the query for the big data platform chosen by BigML and request it to process the query. Then, the big data platform runs the query and returns the results for the job. Finally, the job format the output and sends it to the client.

3.2.1. Training the BigML models

The machine learning model training is triggered manually using data collected over time from client requests. The first training dataset is generated using simulated user requests. The training happens periodically, and at the end of it, the output model is updated on BigML, which uses it to predict the response time of each big data platform to process

users' queries. BigML has to train and keeps a model for each *big data* platform, where the dependent variable represents the response time of the *big data* platform, and the query parameters are the (independent) explanatory variables. We can denote the response variable by y and the set of explanatory variables by $x_1, x_2, ..., x_p$, where p indicates the number of explanatory variables. The relationship between y and $x_1, x_2, ..., x_p$ is approximated by the regression model of the equation 1:

$$y = f(x_1, x_2, ..., x_p) + \varepsilon \tag{1}$$

with ε being a random error representing the discrepancy in the approximation. Thus, the function $f(x_1, x_2, ..., x_p)$ describes the relationship between y and $x_1, x_2, ..., x_p$.

There are many regression models [Chatterjee and Hadi 2015] that can be used to estimate the response time of each big data platform. The BigML solution proposed in this paper is flexible to integrate any machine learning model that meets the following requirements: algorithm inputs are pairs of training examples ($\vec{x}_1$, y_1), ($\vec{x}_2$, y_2), ($\vec{x}_p$, y_p), where $\vec{x}$ is a feature vector representing the explanatory variables and y is the query response time. Being aware that the response time is the response variable y, then for each training input pair ($\vec{x}_i$, y_i), the response variable y_i is defined by the equation 2 and the feature vector $\vec{x}_i$ by the equation 3:

$$y_i = response_time_i \tag{2}$$

$$\begin{aligned} \vec{x}_i = (&area_i, time_interval_i, count_i, swap_free_i, mem_free_i, \\ &load_one_i, load_five_i, load_fifteen_i, \\ &cpu_user_i, cpu_system_i, bytes_in_i, bytes_out_i) \end{aligned} \tag{3}$$

The explanatory variables *area* and *time_interval* come from the spatial and temporal query filters, and *count* is generated by the Indexing System, which returns the estimated number of the query results. By combining the *area*, *time_interval* and *count* variables, it is possible to estimate, more accurately, the cost of processing the query. The other metrics are obtained from the cluster monitoring system and are important because they impact the performance of a big data platform when running a spatiotemporal query. The cluster metrics are collected to represent the state of the cluster at the time of the query. Several other variables can be coupled to the BigML architecture, but this work explores the abovementioned variables.

4. Experiments

This section evaluates the architecture proposed in this paper for processing a large volume of spatiotemporal data. The section 4.2 measures the architecture when ingesting and storing a large volume of data per second. The section 4.3 evaluates the architecture performance when processing spatiotemporal queries using the BigML search engine to choose, in real time, between the big data platforms Elasticsearch, Apache Spark, and SciDB, the best one to run each query.

4.1. Experimental Environment

The architecture integrates three big data platforms in our work: i) Elasticsearch version 7.0.1, ii) SciDB version 18.3, and iii) Apache Spark version 2.3.2. The platforms were

deployed on eight servers in the Amazon EC2[2] Cloud Computing environment. Each server has 3.1 GHz Intel Xeon Platinum 8175 processors with 16 vCPUs, 128 GB of RAM and 600 GB of SSD. The computers were connected by a 10 Gbit/second Ethernet network with dedicated bandwidth of 3500 Mbps.

More than 7.2 billion observations related to 16.7 million pixels of an area of 800.824 km^2 covering a piece of the Midwest and Southeast of Brazil have been stored on every big data platform. Each pixel has a time series with 435 observations between 2000 and 2019 extracted from NASA's MOD13Q1 product [3] (National Aeronautics and Space Administration), with a periodicity of 16 days between observations and a spatial resolution of 250 meters.

4.2. Evaluation of Data Ingestion and Distributed Storage

This section aims to evaluate the architecture's performance in collecting, transforming, and ingesting a large volume of spatiotemporal data. In this evaluation, we use Apache Kafka on version 2.0.0 as the message exchange system and Spark Streaming, version 2.3.2, as the engine for creating the data processing jobs. The data is replicated for indexing and storing on the big data platforms Elasticsearch, SciDB, and Apache Spark.

Three servers were used in the Amazon EC2 Cloud Computing environment to receive, transform, and insert the datasets. Each server has an Intel Xeon Platinum 8175 processor with 16 vCPUs, 128 gigabytes of RAM, and 600 gigabytes of storage. The machines were connected to a network with a dedicated bandwidth of over 3000 Mbps. The input data is sent as messages to Kafka containing the temporal observations with the pixel band values and vegetation indices. One machine was used to send the query requests to the cluster and collect the metrics. This machine has a 2.4 GHz Intel Xeon E5-2676 v3 processor with 8 cores, 32 GB of RAM, and 1TB of SSD.

4.2.1. Results and Discussions

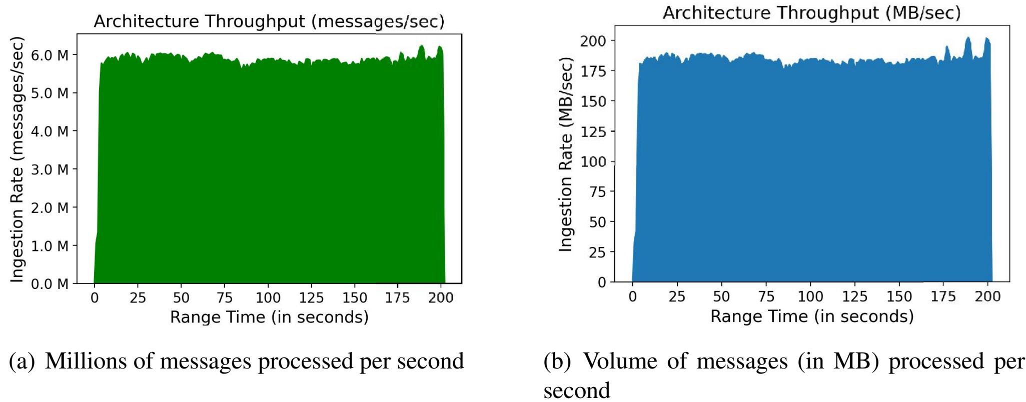
Experiments were carried out to measure the number of messages and the volume of data that the architecture can index and store. Figure 2(a) shows the number of messages processed per second during 200 seconds. It processed up to 6.2 million messages per second, keeping an average of approximately 5.8 million messages processed per second over time. The standard deviation of roughly 630,000 messages processed per second shows that the platform maintained a spatiotemporal data processing rate over time.

Figure 2(b) presents the volume of messages in MB processed per second in 200 seconds by the architecture. Each message has approximately 40 bytes because they were compressed using the lz4 algorithm before being sent to Apache Kafka to increase the throughput. The proposed architecture processed up to 164 MB per second, with an average of 148 MB/second. The consumption in MB of messages per second remained constant, with a standard deviation of 15 MB/second.

Table 1 shows how long takes to store and index the 7.2 billion spatiotemporal data in Elasticsearch, Spark, and SciDB. Spark had the shortest time (780 seconds) to store all

[2]https://aws.amazon.com/pt/ec2/.

[3]This dataset is available by LAPIG (Image Processing and Geoprocessing Laboratory)



(a) Millions of messages processed per second

(b) Volume of messages (in MB) processed per second

Figure 2. Evaluation of the architecture in processing the spatiotemporal data.

the data because it compresses with Snappy and encodes the data in parquet format, and also it does not build an indexing system on top of the data. Apache Spark had insertion throughput almost 39 times higher than Elasticsearch and 43 times higher than SciDB. In addition, Spark consumed only 12.7 GB of disk space to index and store the data in parquet format, which is 21 times lower than Elasticsearch and 20 times less than SciDB.

	Apache Spark	Elasticsearch	SciDB
Total time (seconds)	780	30 360	33 534
Dataset size (in GB)	12,7	273	262

Table 1. Time to store the dataset in each big data platform.

Elasticsearch required 30,360 seconds to store and build spatial and temporal indexes. This indexing has a high computational cost because each pixel has to be indexed spatially (using its latitude and longitude) and temporally, using the date of each observation. The spatial indexing has less weight than the temporal indexing because the 16,777,216 *pixels* are indexed only once. But, the temporal indexing significantly impacted the Elasticsearch throughput, since it had to index all 7.2 billion observations.

SciDB stored and indexed the data in 33534 seconds. It was worse than Elasticsearch because it indexed spatially and temporally each of the 7.2 billion observations. In its dimension-based indexing system, each observation's latitude, longitude, and date are stored as an array with three dimensions. Each of these three dimensions must be indexed to allow spatiotemporal filters to be applied to queries. SciDB consumed 9 GB more disk space than Elasticsearch to store the data. Elasticsearch and SciDB have a larger storage footprint than Spark because they lack efficient formats for storing spatiotemporal data.

4.3. Evaluation of Query Processing with BigML

This section evaluates the BigML when deciding which big data platform is better for each query, analyzing the following machine learning models. First, the Generalized Linear Model (GLM) [Nelder and Wedderburn 1972] setting $family = gaussian$ and $alpha = 0$. Second, the Distributed Random Forest (DRF) [Cutler et al. 2012] using $score_tree_interval = 5$ and $distribution = gaussian$. Third, the Gradient

Boosting Machine (GBM) [Friedman 2001] with $ntrees = 68$, $max_depth = 11$, and $min_rows = 1$. Fourth the XGBoost [Chen and Guestrin 2016] with the parameters $ntrees = 93$, $max_depth = 12$, and $min_rows = 5$. Finally, the Deep Learning [LeCun et al. 2015] model using $epochs = 400$ and $activation = RectifierWithDropout$. We have used the open source library H2O [4] in version 3.36. The experiments use the K-fold cross-validation method, with k=5, using the default values for the remaining H2O algorithms' parameters.

In all, 1983 spatial and temporal constraints were generated from: (i) 247 spatial filters extracted from the region of the state of Goiás in Brazil, ii) 7 temporal filters in the interval from 2016 to 2019 and iii) 1729 (247 * 7) spatiotemporal filters combining the 247 spatial and 7 temporal filters. The queries return from 0 to 2 billion results using these spatial and temporal filters. Each query ran ten times on each of the three big data platforms. We collected cluster metrics ($\vec{x}_i$) and the query response time (y_i) as input for training each machine learning algorithm. We extracted 75% for the training set and 25% for the test set.

SciDB and Apache Spark were installed with the default settings, without any optimization or customization. Regarding Elasticsearch, the JVM memory default was increased from 1 GB to 48 GB, as queries were causing memory overflow during testing. The tests on Spark ran using the Hadoop ecosystem version 3.1.1, with Apache Yarn scheduling the queries and managing the cluster resources consumed by Spark. The data were stored in HDFS on parquet format with snappy [5] compression.

4.3.1. Results and Discussion

This section presents the results of evaluating BigML in predicting the big data platform to process each query. BigML predicts the query response time for each big data platform tied to the architecture (e.g., Elasticsearch, SciDB, and Spark). BigML chooses the platform with the lowest response time predicted. The architecture is flexible to use different machine learning algorithms for this prediction, and the table 2 presents a comparison between the five algorithms evaluated in this work.

Table 2 compares the accuracy and RMSE of the machine learning models. The highest accuracy was identified in the GBM and XGBoost models, 92.1% and 90.9%, respectively. Some studies show that these boosting methods perform well with tabular input data, which is our input format, for two main reasons [Nielsen 2016]: (i) ability to create a rich representation of the data by approximating complex relationship functions between the data, including interactions in high-dimensional spaces; ii) versatility to adapt to different sets of explanatory variables by automatically performing feature selection.

The XGBoost is the model used by SmarT [de Oliveira et al. 2021], and table 2 shows that GBM has a better accuracy in this task. Our proposed solution, BigML, is flexible in using any machine learning method and the remaining evaluation in this Section considers that BigML uses the GBM algorithm. BigML achieved an overall accuracy of 92.1%, in choosing the best platform compared to SmarT with 90.9%. This result is

[4] https://www.h2o.ai/
[5] https://github.com/google/snappy

satisfactory because the BigML using the GBM algorithm achieved a high accuracy that reduces the response time from spatiotemporal queries by choosing the Big Data platform to run each query.

	Overall Accuracy	**RMSE**$_{Elastic}$	**RMSE**$_{Spark}$	**RMSE**$_{SciDB}$
GLM	31.8%	9378	1752	434
DRF	75.2%	1373	242	123
GBM	**92.1%**	**334**	**165**	**101**
XGBoost (SmarT)	90.9%	369	164	99
Deep Learning	77.7%	1218	295	138

Table 2. Comparison of the accuracy and RMSE of machine learning algorithms.

Table 2 shows that the RMSE$_{Elastic}$ (concerning to Elasticsearch) is the highest among the three big data platforms. This result can be explained because Elasticsearch has a huge variation in response time as the size of query results increases. This causes the task of predicting the query response time more difficult. Elasticsearch has a very efficient internal indexing mechanism, but its performance depends on the size of the results. When it gets larger than a certain limit [6], the response time increases substantially [Thacker et al. 2016].

When the number of results increases, Spark and SciDB perform better. Spark stores the data in Parquet format and can retrieve a larger dataset faster than Elasticsearch. SciDB has an efficient internal system for retrieving spatiotemporal data from disk. It can perform better than Spark for many results because it exploits its efficient indexing and retrieval system for multidimensional data [Zhang et al. 2016]. Spark, by contrast, has to scan a lot of Parquet files to filter the results. When the dataset is larger than the memory size, the intermediate results need to be serialized on Spark and stored on secondary storage devices (disk or SSD) or recalculated

Table 3 compares the mean response time for the spatiotemporal queries with BigML, using GBM algorithm, when all big data platforms are available or just two of them. As seen in Table 3, using BigML, the query response time was 490ms versus 482ms if it had always chosen the best platform ("Best" column). Even when BigML does not select the best platform, there is a slight difference in response time if it has always chosen the best one for each query. It is recommended to have all three platforms to have the best possible performance. But if it is not possible, table 3 shows that keeping Spark and SciDB is the best choice. SciDB proved to perform better than the others when choosing only one big data platform. Table 3 shows that BigML had a high accuracy when Elasticsearch was deployed with Spark or SciDB because it could identify the features that modeled the high response time variance of Elasticsearch.

5. Conclusion

The large volume of spatiotemporal data, with high dimensionality being generated in large quantities at all times poses the problem of processing it in the context of Big Data. The efficient filtering and retrieval of a large volume of data are one of the biggest open

[6]This limit depends heavily on the Elasticsearch and the cluster configuration.

	Best	BigML	Spark	Elasticsearch	SciDB
All platforms	482	490	822	6291	605
Elasticsearch + SciDB	533	533	∅	6291	605
Elasticsearch + Spark	796	796	822	6291	∅
Spark + SciDB	513	522	822	∅	605

Table 3. Comparison of the average response time (ms).

challenges in the area. This work proposes a new architecture to process spatiotemporal data efficiently, ingesting and processing a large volume of data per second.

The architecture has a search engine, BigML, that can efficiently process spatiotemporal data, and it is flexible to integrate any machine learning algorithm. In this work, we evaluated five machine learning algorithms, where GBM had the best accuracy with 92.1% against 90.9% from SmarT [de Oliveira et al. 2021] in choosing the best platform among Apache Spark, Elasticsearch, and SciDB to process a spatiotemporal query. The response time decreases 23% using BigML than the best big data platform evaluated (e.g., SciDB). Furthermore, SmarT's architecture is not scalable and cannot process many parallel spatiotemporal parallel queries, which is mandatory in Spatial Big Data. The architecture proposed in this paper can handle many concurrent queries efficiently using distributed jobs to process each spatiotemporal query.

One disadvantage of the architecture proposed in this work is that data must be replicated among the big data platforms since there is no efficient bus for data storage and retrieval. In future work, we plan to create an efficient data bus that integrates the various big data platforms. Lastly, we also intend to evaluate other machine learning models and the impact of other variables, such as internal metrics for each big data platform.

References

Baumann, P., Furtado, P., Ritsch, R., and Widmann, N. (1997). The rasdaman approach to multidimensional database management. In *Symposium on Applied Computing: Proceedings of the 1997 ACM symposium on Applied computing*, volume 1997, pages 166–173.

Brown, P. G. (2010). Overview of scidb: large scale array storage, processing and analysis. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 963–968. ACM.

Camara, G., Assis, L. F., Ribeiro, G., Ferreira, K. R., Llapa, E., and Vinhas, L. (2016). Big earth observation data analytics: matching requirements to system architectures. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*, pages 1–6. ACM.

Chatterjee, S. and Hadi, A. S. (2015). *Regression analysis by example*. John Wiley & Sons.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM.

Chi, M., Plaza, A., Benediktsson, J. A., Sun, Z., Shen, J., and Zhu, Y. (2016). Big data for remote sensing: Challenges and opportunities. *Proceedings of the IEEE*, 104(11):2207–2219.

Cutler, A., Cutler, D. R., and Stevens, J. R. (2012). Random forests. In *Ensemble machine learning*, pages 157–175. Springer.

de Assis, L. F. F. G., de Queiroz, G. R., Ferreira, K. R., Vinhas, L., Llapa, E., Sanchez, A. I., Maus, V., and Câmara, G. (2017). Big data streaming for remote sensing time series analytics using mapreduce. *Revista Brasileira de Cartografia*, 69(5).

de Oliveira, S. S. T., Martins, W. S., Sacramento, V., Bueno, E., Cardoso, M., and Pascoal, L. (2019). A parallel and distributed approach to the analysis of time series on remote sensing big data. *Journal of Information and Data Management*, 10(1):16–34.

de Oliveira, S. S. T., Rodrigues, V. J. S., and Martins, W. S. (2021). Smart: Machine learning approach for efficient filtering and retrieval of spatial and temporal data in big data. *Journal of Information and Data Management*, 12(3).

de Oliveira, S. S. T., Vagner, J., Martins, W. S., Palmeiras, A., Quadra, D., and Samambaia, C. (2020). Smart: Uso de aprendizado de máquina para filtragem e recuperação eficiente de dados espaciais e temporais em big data. In *SBBD*, pages 85–96.

Doan, K., Oloso, A. O., Kuo, K.-S., Clune, T. L., Yu, H., Nelson, B., and Zhang, J. (2016). Evaluating the impact of data placement to spark and scidb with an earth science use case. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 341–346. IEEE.

Elasticsearch (2022). Elasticsearch. `https://www.elastic.co`.

Eldawy, A. and Mokbel, M. F. (2015). Spatialhadoop: A mapreduce framework for spatial data. In *2015 IEEE 31st international conference on Data Engineering*, pages 1352–1363. IEEE.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232.

Hadoop (2022). Apache hadoop. `https://hadoop.apache.org`.

Hamdi, A., Shaban, K., Erradi, A., Mohamed, A., Rumi, S. K., and Salim, F. D. (2022). Spatiotemporal data mining: a survey on challenges and open problems. *Artificial Intelligence Review*, 55(2):1441–1488.

Karim, S., Soomro, T. R., and Burney, S. A. (2018). Spatiotemporal aspects of big data. *Applied Computer Systems*, 23(2):90–100.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

Lu, M., Pebesma, E., Sanchez, A., and Verbesselt, J. (2016). Spatio-temporal change detection from multidimensional arrays: Detecting deforestation from modis time series. *ISPRS Journal of Photogrammetry and Remote Sensing*, 117:227–236.

Nelder, J. A. and Wedderburn, R. W. (1972). Generalized linear models. *Journal of the Royal Statistical Society: Series A (General)*, 135(3):370–384.

Nielsen, D. (2016). Tree boosting with xgboost-why does xgboost win" every" machine learning competition? Master's thesis, NTNU.

Paradigm4 (2022). Scidb. https://www.paradigm4.com.

Parimala, M., Swarna Priya, R., Praveen Kumar Reddy, M., Lal Chowdhary, C., Kumar Poluru, R., and Khan, S. (2021). Spatiotemporal-based sentiment analysis on tweets for risk assessment of event using deep learning approach. *Software: Practice and Experience*, 51(3):550–570.

Procopiuc, O., Agarwal, P. K., Arge, L., and Vitter, J. S. (2003). Bkd-tree: A dynamic scalable kd-tree. In *International Symposium on Spatial and Temporal Databases*, pages 46–65. Springer.

Roy, S., Bhunia, G. S., and Shit, P. K. (2021). Spatial prediction of covid-19 epidemic using arima techniques in india. *Modeling earth systems and environment*, 7(2):1385–1391.

Shrivastava, S. (2020). A review of spatial big data platforms, opportunities, and challenges. *IETE Journal of Education*, 61(2):80–89.

Spark (2022). Apache spark. https://spark.apache.org.

Thacker, U., Pandey, M., and Rautaray, S. S. (2016). Performance of elasticsearch in cloud environment with ngram and non-ngram indexing. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pages 3624–3628. IEEE.

Wang, F., Li, M., Mei, Y., and Li, W. (2020). Time series data mining: A case study with big data analytics approach. *IEEE Access*, 8:14322–14328.

Wang, S., Zhong, Y., and Wang, E. (2019). An integrated gis platform architecture for spatiotemporal big data. *Future Generation Computer Systems*, 94:160–172.

Xu, M., Zhao, L., Yang, R., Yang, J., Sha, D., and Yang, C. (2020). Integrating memory-mapping and n-dimensional hash function for fast and efficient grid-based climate data query. *Annals of GIS*, pages 1–13.

Yang, C., Clarke, K., Shekhar, S., and Tao, C. V. (2020). Big spatiotemporal data analytics: A research and innovation frontier.

You, S., Zhang, J., and Gruenwald, L. (2015). Large-scale spatial join query processing in cloud. In *2015 31st IEEE international conference on data engineering workshops*, pages 34–41. IEEE.

Zhang, X., Khanal, U., Zhao, X., and Ficklin, S. (2016). Understanding software platforms for in-memory scientific data analysis: A case study of the spark system. In *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, pages 1135–1144. IEEE.

Zhang, X., Khanal, U., Zhao, X., and Ficklin, S. (2018). Making sense of performance in in-memory computing frameworks for scientific data analysis: A case study of the spark system. *Journal of Parallel and Distributed Computing*, 120:369–382.