2021 International Conference on Big Data Analysis and Computer Science (BDACS) | 978-1-6654-2561-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/BDACS53596.2021.00031

Distributed Large-scale Time-series Data Processing and Analysis System Based on Spark Platform

Bangyan Du
School of Information Science and Technology
Peking University
Beijing,China
dubangyan@stu.pku.edu.cn

***Abstract*—With the diversification of applications and the diversity of users, the era of big data is always producing various new types of data, such as graph data, time series data, and spatial data. When predicting the future trend of the system, we always need to learn from the historical working state, and time series data plays a huge value in this aspect. Hardware resources are becoming cheaper, computing power has made major breakthroughs, and multi-machine and multi-core scenarios have become commonplace. At present, the focus of large-scale time series data processing is either focused on data collection and storage, or on single-machine processing, which has been unable to meet the existing needs. In response to this problem, this paper designs and implements a distributed large-scale time series processing and analysis system based on the Spark platform. The system framework is mainly divided into storage layer, operator layer and algorithm layer. At the storage layer, the system completes the organization and indexing of large-scale time series data based on HDFS and Hive. At the operator layer, the system provides users with basic operations commonly used for time series data on the Spark platform, and allows users to directly use these operators to implement custom time series related processing algorithms. At the algorithm level, the system implements some commonly used time series analysis algorithms in the Spark platform, including time series similarity query, clustering and prediction, so that users can directly use these algorithms to analyze time series. Finally, the feasibility and practicability of this system are verified by testing the performance and function of the system.**

Keywords-Time-series data; distributed ; analysis ; Spark

I. Introduction

We are always collecting various time series data through time nodes. In business, we observe the weekly interest rate, daily stock closing price, monthly price index, annual sales volume, etc. In meteorology, we will observe the daily maximum and minimum temperature, annual precipitation and drought refers to the hourly wind speed, etc. In agriculture, we will record annual crop and livestock output, soil erosion, export sales and other aspects. In biology, we will observe the state of the electrical activity of the heart every millisecond. In ecology, we will record the changes in animal populations. In computer science, we will record the memory, CPU usage and other resource usage of the machine at every moment. These data are all time series data, and we cannot list all the fields of time series research.

The analysis and mining of time-series data can often produce more than estimated value. For example, if we can dig out the potential changes in stock prices from stock data, then we can make buying and selling decisions in a timely and correct manner. If we find the changes in certain physiological indicators of our body (such as blood pressure), then we can accurately adjust our physical condition and predict the future physical condition based on the current situation to prevent the occurrence of certain diseases in the future. If we can analyze the change law of the machine load from the machine log data, it will help us carry out the correct load balancing, so that all machines can play the maximum effect.

Driven by big data analysis, excellent platforms like Hadoop and Spark have been born. They are based on the MapReduce[5] framework proposed by Google, which transforms each task into a Map and Reduce task that can be distributed on multiple machines. The resources of a cluster constructed by multiple machines can be used to store large-scale data and run our tasks in a distributed manner, which is of great help to the storage of large-scale time series and the efficient execution of analysis algorithms.

However, Hadoop and Spark are both general distributed platforms and have not been customized for any field. Therefore, if we want to perform some time series related tasks on it, we need our users to program and implement them. For example, time series storage, clustering, prediction algorithms, etc. are not available in the algorithm libraries provided by the latest versions of these platforms. This makes the threshold for us to run time series analysis on distributed platforms higher, and different users may need to rewrite the same algorithm, resulting in inefficiency. Therefore, SparkR came into being.

We know that R is a stand-alone data analysis platform, which contains a wealth of algorithms and models, so that we do not need to rewrite various known algorithms, which facilitates our use. SparkR is a callable package of R. It allows us to call Spark[6] in R through the R language API provided by SparkR. But it only implements the DataFrame structure in R in a distributed environment, which makes the DataFrame we use in R changed from the previous local data to distributed data. However, all the algorithms in the rich algorithm library in R were run on the local data loaded into the memory. So when the DataFrame becomes a distributed data set, the algorithms in the original rich algorithm library are no longer in use, unless these distributed data can also be placed in a single machine's memory. Therefore, in order to perform data analysis on distributed DataFrame, the current version of SparkR provides APIs for a small number of machine learning algorithms such as GLM, AFT, GBT, kMeans, etc.. So that we can also execute machine learning algorithms for distributed DataFrame in R. However, there are very few algorithms provided in the current API, and none of them are specifically designed for time series data. Therefore, if we want to perform time series analysis in the Spark environment, we still need to manually write the algorithm.

Spark-timeseries is a time series analysis package based on the Spark platform developed by the Google team on GitHub. It encapsulates large-scale time series data into distributed data and RDD for storage and processing. However, it is still in the development stage, and only provides traditional statistical time series analysis algorithm packages such as ARMA and GARCH, and has not implemented the new neural network model. The neural network computing platforms such as Tensorflow and Theano are only for neural network computing, and cannot support the storage of large-scale time series data and traditional statistical analysis.

With the rapid development of deep learning, some new non-traditional statistical neural network models, such as RNN[2], RNN-RBM[3], LSTM[4], etc., are gradually used in the analysis of time series. These models themselves are relatively complex and have a large number of parameters.

Therefore, based on the above analysis, we lack a complete system for processing and analyzing large-scale time series in a distributed environment, and this paper aims to solve this problem.

II. Related Work

This chapter needs to introduce some existing work on time series processing and analysis, which will be introduced from the two aspects of time series data storage and time series data analysis.

A. Time Series Storage

OpenTSDB is a distributed and scalable time series database. It supports millisecond data collection for all metrics (some time series data), and supports permanent storage (without downsampling). Its storage does not have a pattern, but uses tags to implement the concept of dimensions. Therefore, the storage of OpenTSDB is more flexible, and it is very suitable for real-time collection and storage of large-scale time series data. Therefore, it is often used in various data analysis-oriented enterprises to collect user data and provide early warning of some data. The biggest advantage of OpenTSDB is that it can carry out distributed storage, which solves the storage problem of large-scale time series.

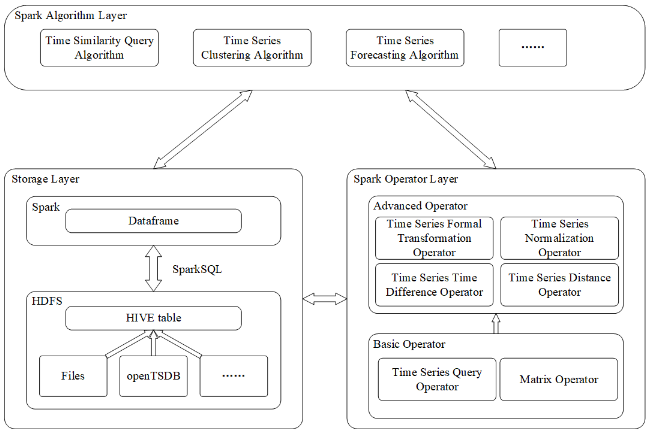
InfluxDB is an open source distributed time series, event and indicator database. Written in Go language. No external dependencies are required. Its design goal is to achieve distributed and horizontal expansion. Moreover, InfluxDB is one of the few time series databases in TSDB that implements users and roles. It provides three roles: Cluster Admin, Database Admin, and Database User. In addition, InfluxDB's data collection system also supports multiple protocols and plug-ins like line text, UDP, Graphite and CollectD. So in general, InfluxDB, is also suitable for the storage of large-scale time series.

B. Time Series Analysis

Spark-Timeseries is a GitHub open source project developed by the Google team. It is based on the Spark platform and is used to analyze large-scale time series data sets. It uses RDD on Spark to store time series data, and mainly provides some operations for large-scale time series data, such as data alignment, missing value estimation, etc.; and provide some traditional statistical time series forecasting models, such as ARIMA, GARCH, etc.. There are also some statistical tests, such as the Durbin-Watson test. However, the project is still under development, the functions provided are not yet complete.Moreover, it only contains a small number of traditional statistical time series forecasting algorithms, and does not involve the latest neural network-related analysis algorithms and other time series analysis algorithms, such as time series clustering. Therefore, this project still needs to be perfected for time series processing and analysis.

III. System Structure

Through the introduction and analysis of the two parts of introduction and related work, we know that it is very necessary to build a large-scale time series processing and analysis system. Therefore, this chapter introduces the thoughts and architecture of the system designed in this paper for the processing and analysis of large-scale time series as a whole. The architecture of the system is shown in Figure 1.



Spark Algorithm Layer
Time Similarity Query Algorithm
Time Series Clustering Algorithm
Time Series Forecasting Algorithm
......
Storage Layer
Spark
Dataframe
SparkSQL
HDFS
HIVE table
Files
openTSDB
......
Spark Operator Layer
Advanced Operator
Time Series Formal Transformation Operator
Time Series Normalization Operator
Time Series Time Difference Operator
Time Series Distance Operator
Basic Operator
Time Series Query Operator
Matrix Operator

Figure 1. System architecture diagram

The overall design idea of the system is to use the distributed file system HDFS to store large-scale time series data, and then use the distributed computing platform of Spark to perform the analysis of large-scale time series data, which is mainly divided into three parts: storage layer , operator layer and algorithm layer. The three layers included in the system will be introduced below.

A. *Storage Layer*

The storage layer is responsible for the storage of time series. In our system, it acts as a data source. However, for our system, not only the time series database, but other files such as HDFS, Hbase, etc. can be used as data sources, and these data sources form a data warehouse. In order to perform offline analysis on this huge data warehouse, we need to build a tool on top of this data warehouse to help us organize the data. Hive is a data warehouse tool based on the distributed file system (HDFS). It can provide us with functions such as offline data query and simple analysis efficiently and conveniently. Moreover, Hive supports the creation of relational database tables, SQL-like statements, etc., so it is very Suitable for this role. In this way, we can use Hive in HDFS to build a data warehouse for storing large-scale time series.

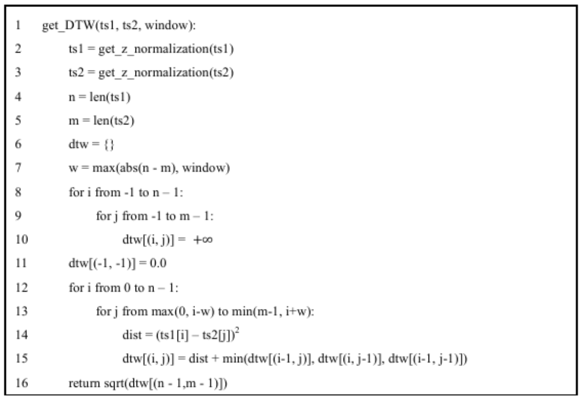
However, we still have to analyze in Spark in the end, so we also need to store time series data in Spark. The distributed data structure on Spark is called RDD, but RDD is a general data structure, not designed for time series, and RDD only provides some general operations like map, reduce for time series data. It is too basic, so it is not suitable for storing time series. But fortunately, the new version of Spark provides the DataFrame data structure, which is very similar to the dataframe in R. But it is a more advanced distributed data structure based on RDD and suitable for data analysis.It can be seen as a table in a relational database. At the same time, Spark also designed many very practical operations for the DataFrame data structure, such as select, join, groupby, etc., and supports the execution of SQL statements on the DataFrame. So it is very suitable for storing large-scale time series data in the Spark environment . What's more convenient is that Spark now supports SparkSQL (DataFrame is also the core data structure in SparkSQL). So we can easily retrieve the data we want from Hive by using SparkSQL. In this way, we can use DataFrame to store time series data and perform data analysis on Spark.

Our system uses structured files on HDFS to store time series related data, uses Hive as the engine, organizes and manages these data by building Hive tables, and uses DataFrame to store time series data and matrix data in Spark.

B. *Operator Layer*

The operator layer is responsible for providing some basic and frequently-called operations for the upper algorithm. These operations are collectively called operators. For example, if we need to obtain time series data for a period of time, then we need time-series query operators. Almost all time series analysis algorithms involve matrix operations, so we need to abstract the operations and representations involving matrices into matrices operator. The classic algorithms of SVD matrix decomposition include: two-sided Jacobi rotation algorithm [7], single-sided Jacobi rotation algorithm [8] and so on. But more commonly, we can get it by calculating the feature vector algorithm .

The above two operators are basic operators. On top of these two operators, we need to customize other commonly used operators for time series. These operators are called advanced operators. (1) The time series form conversion operator is used to transform the general representation form of the time series into its fourier, wavelet, SAX and other representation forms according to algorithm requirements; (2) Time series normalization operator is used to normalize time series data; (3) Time series time difference operator is used to perform time difference processing on time series data, so as to obtain a time series with stationarity; (4) Time series distance operator is used to calculate the distance between time series, providing a basis for time series similarity query algorithm, time series clustering and other algorithms. Takint the code for calculating the DTW distance of the time series as an example, as shown in Figure 2.



```
get_DTW(ts1, ts2, window):
    ts1 = get_z_normalization(ts1)
    ts2 = get_z_normalization(ts2)
    n = len(ts1)
    m = len(ts2)
    dtw = {}
    w = max(abs(n - m), window)
    for i from -1 to n – 1:
        for j from -1 to m – 1:
            dtw[(i, j)] = $+\infty$
    dtw[(-1, -1)] = 0.0
    for i from 0 to n – 1:
        for j from max(0, i-w) to min(m-1, i+w):
            dist = $(ts1[i] – ts2[j])^2$
            dtw[(i, j)] = dist + min(dtw[(i-1, j)], dtw[(i, j-1)], dtw[(i-1, j-1)])
    return sqrt(dtw[(n - 1,m - 1)])
```

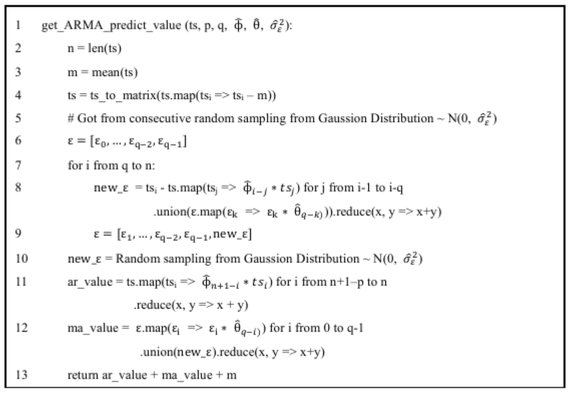
Figure 2. Pseudo code for calculating DTW distance of time series

All in all, the operator layer is to provide some general time series processing operators according to the needs of the upper algorithm. The upper algorithm does not need to consider the operation problems outside the algorithm itself. Therefore, the operator layer can also enable us to continue to implement more advanced algorithms at the algorithm layer and enrich the functions of the system without the need to consider more details of the implementation of low-level operators.

C. *Algorithm Layer*

The top layer of the system is the algorithm layer. This layer is responsible for providing various time series analysis and processing algorithms to complete the tasks that users need to complete. These algorithms are implemented based on the operation of the storage layer and the operator layer. The algorithm layer of the system in this paper mainly provides three types of algorithms: similarity query, clustering and prediction of time series. The clustering algorithm includes kMeans clustering and kShape clustering. The prediction includes traditional ARMA model and RNN model. These algorithms are all processing algorithms for large-scale time series data implemented under the distributed platform Spark. In the implementation of these algorithms, we can see that they make extensive use of the various operators we mentioned in operator layer, especially matrix operators. From here we can also see the scalability of the system proposed in this paper in terms of algorithms. That is to say, for the new algorithms that users want to implement, users can use the various operations provided by our operator layer to implement them, so as to ensure these algorithms can be run in parallel to process large-scale time series data and realize the expansion of the system.

Taking the calculation of ARMA model parameters and the prediction of future values for large-scale time series as an example. The matrix operations that need to be used when calculating the ARMA parameters can be implemented using the inverse matrix and matrix multiplication operators we implemented in the operator layer. The storage of the matrix and time series adopts the DataFrame structure. After we calculate the model parameters in ARMA, we can use the model to make time series forecasting. Assuming that our input is a time series S of length n, we want to predict the time series at time point n+1 value. See Figure 3.



```
get_ARMA_predict_value (ts, p, q, $\hat{\phi}$, $\hat{\theta}$, $\hat{\sigma}_\varepsilon^2$):
    n = len(ts)
    m = mean(ts)
    ts = ts_to_matrix(ts.map(ts$_i$ => ts$_i$ – m))
    # Got from consecutive random sampling from Gaussion Distribution ~ N(0, $\hat{\sigma}_\varepsilon^2$)
    $\varepsilon = [\varepsilon_0, \dots, \varepsilon_{q-2}, \varepsilon_{q-1}]$
    for i from q to n:
        new_ε = ts$_i$ - ts.map(ts$_j$ => $\hat{\phi}_{i-j} * ts_j$) for j from i-1 to i-q
                    .union(ε.map($\varepsilon_k$ => $\varepsilon_k * \hat{\theta}_{q-k)}$)).reduce(x, y => x+y)
        $\varepsilon = [\varepsilon_1, \dots, \varepsilon_{q-2}, \varepsilon_{q-1},$new_ε]
    new_ε = Random sampling from Gaussion Distribution ~ N(0, $\hat{\sigma}_\varepsilon^2$)
    ar_value = ts.map(ts$_i$ => $\hat{\phi}_{n+1-i} * ts_i$) for i from n+1–p to n
                .reduce(x, y => x + y)
    ma_value = ε.map($\varepsilon_i$ => $\varepsilon_i * \hat{\theta}_{q-i)}$) for i from 0 to q-1
                .union(new_ε).reduce(x, y => x+y)
    return ar_value + ma_value + m
```

Figure 3. Pseudo code for calculating the predicted value of the ARMA model

IV. EXPERIMENT

The previous chapter gave a detailed introduction to the system proposed in this article. This chapter will implement the performance and functions of the system to illustrate the feasibility and performance scalability of the system in the face of large-scale time series data. Because the system implements a lot of things, the experiments in this chapter only carry out experimental tests on the relatively important part of the experimental combination, but this does not affect the experimental description of the overall performance of the system.

All the experiments in this section are done in the cluster. There are a total of 7 servers in the cluster. Each machine has 39GB of available memory, 10T of available disk space, and 8 physical cores of the processor. We built a Hadoop 2.6.5 system in the cluster to provide an HDFS file system for distributed storage of files, and used YARN to manage the cluster. At the same time, we built Hive 2.1.1 on HDFS as the time series data organization and query engine. On YARN, we built the Spark 2.1.1 platform, and used Python 2.7 to develop system functions. Finally, all our experiments are done by running the Spark program.

A. Time Series Clustering

This paper chooses the more commonly used and flexible kMeans clustering algorithm for performance experiments, and in the experiment we choose the LB_Keogh distance used to optimize the DTW distance calculation as the distance function of the time series. The number of clusters k is set to 10. For the clustering experiment, we still simulated three sets of data manually:

1) A data set of 1GB containing 5.5×10^5 time series with a length of 100;

2) A 5GB data set containing 2.8×10^6 time series with a length of 100;

3) A 10GB data set containing 5.7×10^6 time series with a length of 100. For these three data sets, the results of our kMeans performance experiment are shown in Figure 4.

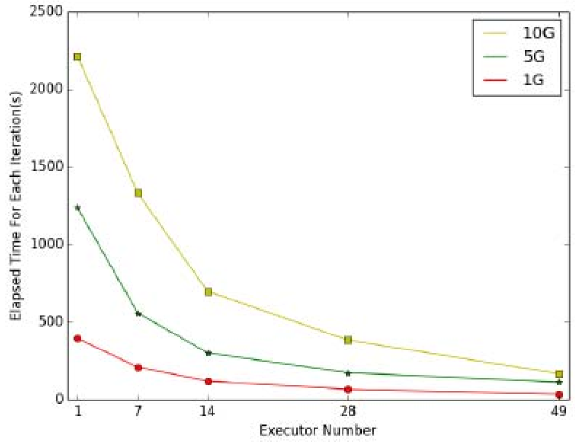


Figure 4. Performance test of kMeans algorithm

In the functional experiment, this paper wants to explore whether there are some potential laws in the training process of the changes in the weights of artificial neural networks through clustering. We collected 4096 weight change data in the weight matrix of a certain layer of a certain convolutional neural network (CNN) in 1000 iterations. We can think of these data as 4096 time series, and the length of each time series is 1000. Therefore, in the experiment, we use the kShape clustering algorithm implemented by the system, set the number of clusters to 10, and perform 100 rounds of iterative clustering operations. Finally, we group these 4096 time series into 10 clusters, and the time series data in one of the clusters is shown in Figure 5.

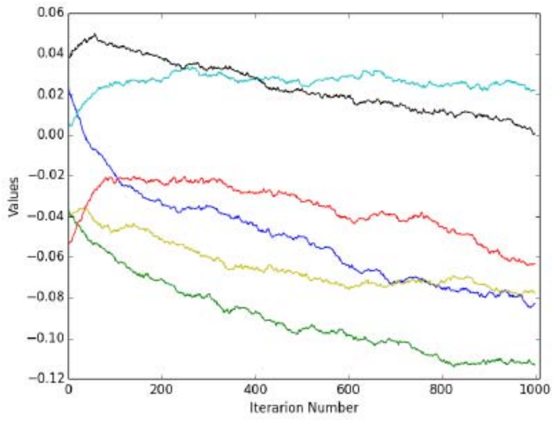


Figure 5. Time series data in cluster 1

Each curve in the figure represents the change process of a weight. The horizontal axis represents the number of iterations in the kShape algorithm, and the vertical axis represents the value of the weight. From the above two figures, we can also see that the time series in each cluster have roughly the same direction of change, while the two clusters have different directions of change. Our clustering results also show that in the training process of the CNN model, some weights have a consistent trend. So these weights may have some inherent relationships, which also achieves our purpose of cluster analysis.

B. Time Series Forecasting

Another example application of time series is that we can model the order data of JD.com from February 1st to March 31st, 2016 to predict the daily order volume. First of all, JD.com only provided a subset of user behavior data from February 1st to March 31st, 2016. We calculated user behaviors in this subset to get the daily order volume of JD.com during this time period. The order volume data is shown in Figure 6.

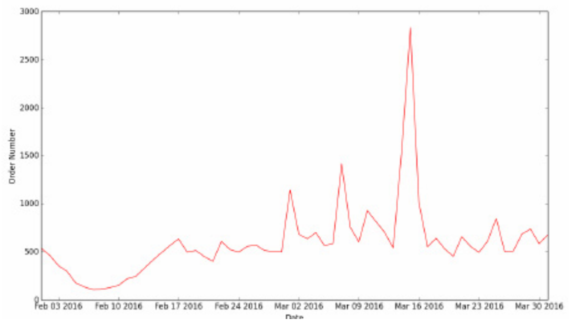


Figure 6. JD daily order volume data

For this time series data, we use ARMA for fitting. We first perform logarithmic transformation processing on the original data; then we perform second-order time difference on the transformed data; finally, we use the ARMA model with p=1 and q=3 to fit the differenced data. After the fitting is completed, we use the data of the first 14 days of February to estimate the error term for these 14 days, and then carry out the ARMA rolling forecast for the following dates, then restore the predicted value of ARMA to the data before the second-order difference and logarithmic transformation, so we get the rolling forecast chart shown in Figure 7.

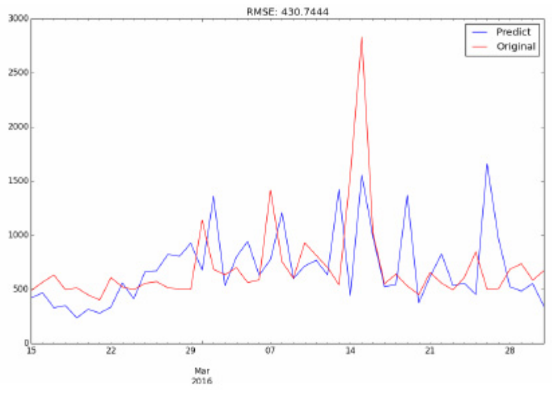


Figure 7. JD daily order volume data

V. CONCLUSION

In today's era of big data, massive amounts of data are generated every day, and many of these data are time series data, such as daily log data of machines, electricity consumption data in life, and daily physiological index data of the human body. These time series data also have very important potential value in our daily lives. If we can analyze these data reasonably, then the results of the analysis will have an important guiding role in the decision-making in our lives.

Existing time series tools are either used for stand-alone analysis, such as R and Matlab; or used to collect time series data, such as OpenTSDB and InfluxDB. There is no tool or platform for analyzing large-scale time series, so how to analyze large-scale time series data has become a top priority.

This paper implements a large-scale time series processing and analysis system based on the distributed environment Spark. The system includes three parts: storage layer, operator layer and algorithm layer. The storage layer is based on HDFS and Hive, used to organize and index the time series data warehouse, so that we can quickly query the time series data in the data warehouse. The operator layer implements the basic operations commonly used in time series analysis on the Spark platform, such as matrix operations, time series form conversion, time series time difference, and time series distance calculation. Above the operator layer, we have implemented an algorithm layer on the Spark platform. The algorithm layer contains some commonly used time series analysis algorithms, such as similarity query, time series clustering, and time series prediction.

Through the construction of these three layers, the system we have implemented can perform common time series analysis tasks on a large-scale time series data warehouse. Users can use the algorithms provided by our algorithm layer to process large-scale time series data. You can use the operations provided by our operator layer to implement new algorithms that meet your own needs at the algorithm layer, that is, our system is scalable in terms of algorithms. In addition, because we use layered design and implementation for the system, the improvement of each layer will also improve the part that depends on this layer, so it will be easier to improve and optimize the system in the future.

Finally, we conducted experiments on some important parts of the system implementation to illustrate the feasibility and performance scalability of our system when dealing with large-scale time series. And we also applied the system to the analysis of actual scenes, explaining the practicability and usability of the time series analysis function provided by our system.

Acknowledgment

The authors are supported by the National Key R&D Program of China 2018YFB1004100.

References

[1] Claire. Time series analysis and application. Pan Hongyu et al., translated. 2 edition. Beijing: Machinery Industry Press, 2011: 1.

[2] Haijin. Neural Networks and Machine Learning. Shen Furao et al., translated. 3rd edition. Beijing: Mechanical Industry Press, 2011: 501-533.

[3] Boulanger-Lewandowski N, Bengio Y, Vincent P. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. ICML, 2012.

[4] Hochreiter S, Schmidhuber J. Long short-term memory. Neural computation, 1997, 9(8): 1735-1780.

[5] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Communications of theACM, 2008, 51(1): 107-113.

[6] Shivaram Venkataraman, Zongheng Yang, Davies Liu. SparkR: Scaling R Programs with Spark.SIGMOD, 2016..

[7] Guo Qiang. Research on parallel JACOBI method for solving matrix singular values. Soochow University, 2011.

[8] De Rijk P.P.M. A one-sided Jacobi algorithm for computing the singular value decomposition on avector computer. SIAM journal on scientific and statistical computing, 1989, 10(2): 359-371.

[9] Liu J, Liang Y, Ansari N. Spark-based large-scale matrix inversion for big data processing. IEEEAccess, 2016, 4: 2166-2176.

[10] Keogh E, Kasetty S. On the need for time series data mining benchmarks: a survey and empiricaldemonstration. Data Mining and knowledge discovery, 2003, 7(4): 349-371.

[11] Koman. Introduction to Algorithms. Pan Jingui et al., translated. 2 ed. Beijing: Mechanical Industry Press, 2011: 511-519.

[12] Keogh E, Chu S, Hart D, et al. An online algorithm for segmenting time series. ICDM 2001,Proceedings IEEE International Conference on. IEEE, 2001: 289-296.

[13] Lin J, Keogh E, Li W, et al. Experiencing SAX: a novel symbolic representation of time series. DataMining and knowledge discovery, 2007, 15(2): 107.

[14] Claire. Time series analysis and application. Pan Hongyu et al., translated. 2 ed. Beijing: Mechanical Industry Press, 2011: 64.

[15] Paparrizos J, Gravano L. k-shape: Efficient and accurate clustering of time series. Proceedings of the2015 ACM SIGMOD International Conference on Management of Data. ACM, 2015: 1855-1870.