UDC 004.67

M. Mamuta, T. Likhouzova

IMPROVING THE EFFICIENCY OF DISTRIBUTED DATA WAREHOUSES

Abstract: The problem of optimal processing and storage of big data is considered. It is proposed to prepare a repository based on a combination of several different types of repositories and adapted to user tasks. An algorithm for transforming the internal structure of the repository and data synchronization has been developed. To measure performance, we used indicators of the amount of memory used for backup and the speed of data processing.

Keywords: big data, data warehouse, optimal allocation of resources, distributed data processing systems, cloud computing.

Introduction

Every year, the problem of storing large amounts of data on various projects becomes more and more important in the IT industry. Interacting with big data and improving the methods of its processing and storage is now a common task for many software developers and technology companies. Every day in the world, a large amount of diverse data is generated and processed, most of which is valuable and should be stored and structured in data warehouses. When data accumulates over a period of time with high intensity, it becomes too much for one database, or when we have different data endpoints, then we move to a solution that uses multiple repositories and databases at once, and build the interaction between them.

Today a large number of different repositories and databases have been developed and are actively used, but they are aimed only at solving a rather narrow range of problems with optimal methods. An alternative to this is a big data warehouse solution that can change the internal structure according to user requests.

The purpose of the study: to increase the speed of big data processing.

Object of research: big data storage architecture.

Subject of research: software and algorithms for processing and storing big data.

Analysis of existing solutions

Analyzing the literature, it can be said that big data software can be both open source and commercial projects, and most of them are cloud services and have a web interface for querying and working with them.

Airflow [1] is a platform made by the community to create, execute and monitor streaming tasks. Airflow has a modular architecture and uses the message queue to organize

any number of executable modules. Pipelines in Airflow are configured with code that allows you to create them dynamically.

Airflow provides a variety of integrations with services that can handle the user's data and can run on the Google Cloud platform, Amazon web services, Microsoft Azure and many other services. This makes Airflow easy to use in the current infrastructure. Anyone who knows the Python programming language can try to create their own Airflow pipeline.

Google Bigtable [2] is a high-performance storage system built into the Google file system. This is the basis of the Cloud Datastore, which is available as part of Google's cloud platform. This is a non-relational database, but it can be described as a distributed, multidimensional and sorted map. Each cell in a large table can have zero or more temporary versions of the data.

Bigtable is designed to scale in the petabyte range to hundreds or thousands of machines to make it easier to add additional machines to the system and automatically use these resources without changing the configuration.

RethinkDB [3] is the first database for real-time web applications for open-source documents that effectively supports complex queries. RethinkDB interacts with traditional storage systems and implements only added nested storage structures, making it more consistent, reliable, and easily replicated. RethinkDB offers easy scalability and fast response to customer requests in real time.

Redis [4] is an alternative open-source cache store and key-value store for big data, which provides an efficient data structure for indexing and speeding up operations. Redis has significant potential for expansion in the master-slave environment. However, support for multiple data structures makes it a better choice for situations with more frequent data access requests.

Table 1.

Comparison of existing solutions for working with data warehouses

Name	Run on your own server	REST API	Data Querying	Selective access to data	Data Lake
Airflow	+	+	+	-	-
Google Bigtable	-	-	+	+	+
RethinkDB	+	-	+	+	-
Redis	+	-	+	+	-

The various types and structures of repositories and databases are aimed at optimally solving only a narrow range of tasks, so a repository that can embody several different types and approaches to data storage and processing will have significant advantages over others.

 ISSN 1560-8956

Materials and methods

Tasks that need to be solved when working with big data include: data collection from various sources, storage, data exchange with external sources and repositories, mining, finding the necessary information in the processed data, visualization of processing results [5, 6].

Repositories for storing raw data can be used as a starting point before processing and analyzing data to retrieve source and unmodified data from this repository, which may have the necessary information in different variations for different types of processing.

SQL technologies can still be used in big data processing, which simplifies the transition from the traditional type of data storage and processing without significant changes in the code base.

NoSQL technologies implement flexible data models that may not have a specific schema and structure, but have horizontal scalability. These databases are aimed at facilitating the management of large-scale projects [5, 7, 8].

You can also select storage types that do not use a specific data structure, but use a distributed file system of one or more servers and contain files with different types and formats of storage.

The trend of transition from structured data to unstructured data [7] makes traditional relational databases undesirable for storage. This inadequacy of relational databases encourages the creation of efficient distributed storage mechanisms. Providing scalable, reliable, and efficient storage for fast-growing data is a key goal of deploying a big data storage tool. Therefore, innovative development of storage systems with improved access performance and fault tolerance is needed.

In different scientific fields, large data sets are becoming an important part of shared resources. Such a huge amount of data is usually stored in cloud data centers. Thus, data replication, which is commonly used for distributed large data management, speeds up data access, reduces access delays, and increases data availability. A detailed review of the state of modern technologies and mechanisms in this area is given in [9, 10]. Mechanisms of data replication in cloud systems can be classified into two main groups: static and dynamic mechanisms. Static data replication mechanisms determine the location of replication nodes at the design stage (Fig. 1), and dynamic replication nodes are selected at runtime. In addition, [10] presents the systematics and comparisons of the considered mechanisms and highlights their main features; open problems and some tips for solving them are given. The review shows that some dynamic approaches allow you to adjust the associated replication strategies at runtime according to changes in user behavior and network topology. In addition, they are applicable to a service-oriented environment, where the number and location of users who intend to access data often have to be determined very dynamically.

To date, the most important thing is to transform the internal structure of the big data warehouse for more optimal work with them. Algorithms used to manage data in repositories.

MapReduce algorithm

MapReduce performs transformations to divide the data to be processed into small pieces and assign them to multiple workers. Technically, the MapReduce algorithm helps send Map and Reduce tasks to the appropriate servers in the cluster.

MapReduce is used to process parallel tasks in large datasets using a large number of computers (nodes). All nodes can be located in one local network and use similar equipment - a cluster, or nodes are separated geographically and administratively, are distributed systems and use more diverse equipment - the network. MapReduce can take advantage of the locality of data by processing it near where it is stored to minimize overhead links. Processing can take place on data stored either in a file system (unstructured) or in a database (structured).

Parallel PFP algorithm

Parallel Frequent Pattern is a useful tool for identifying elements that are common in a sample. A number of important FIM algorithms have been developed to speed up the search for such elements in the big data sample. Unfortunately, when we have too much data and we have to use a lot of memory, the estimated cost can still be too expensive.

To overcome this problem, they began using the FP-Growth algorithm, the so-called parallel PFP algorithm on a cluster of distributed machines. PFP distributes tasks for processing in such a way that each machine performs a separate group of tasks for searching and processing data. This distribution eliminates computational dependencies between machines and, thus, the relationship between them, speeds up the search and processing of data.

Given the huge list of transactions, the algorithm finds all unique functions (sets of field values) and excludes those functions whose frequency is lower in the whole data set. The PFP algorithm is a common implementation, we can use any type of object to denote a function.

HPPC algorithm

HPPC implements standard collections (list, queue, queue, map) with specialized versions that store primitive types without packing them as objects. This leads to better memory usage and increased productivity. High-performance primitive collections have several purposes: first, to use typical collection classes that store memory for primitive types and avoid automatic packaging; secondly, the speed of each operation is a priority.

Results

The approach described in [6] was used to select the mechanisms of static replication. It allows you to cluster data by the types of tasks for which they are used, and to recommend the optimal storage format (replica type) for each cluster.

A modern system must have data backups. The left part of the figure shows a traditional system with a main data warehouse and several backups that are updated from the main. An architectural solution is proposed on the right side of the figure. As a dynamic replication mechanism, the algorithm shown in Figure 2 is proposed. It is proposed to use the resources

 ISSN 1560-8956

available in the system, in this case disk space and computing power, with greater efficiency: not just save backups in standard format, but store them in several different big data stores, each of which performs a specific task better. At the same time, perform constant synchronization between all repositories of the repository and the main repository, using data conversion algorithms for repositories with different structures. The API will also provide access to repositories so that users can use them and benefit from speed and efficiency. At the same time the standard main storage can be used for its intended purpose without any changes.

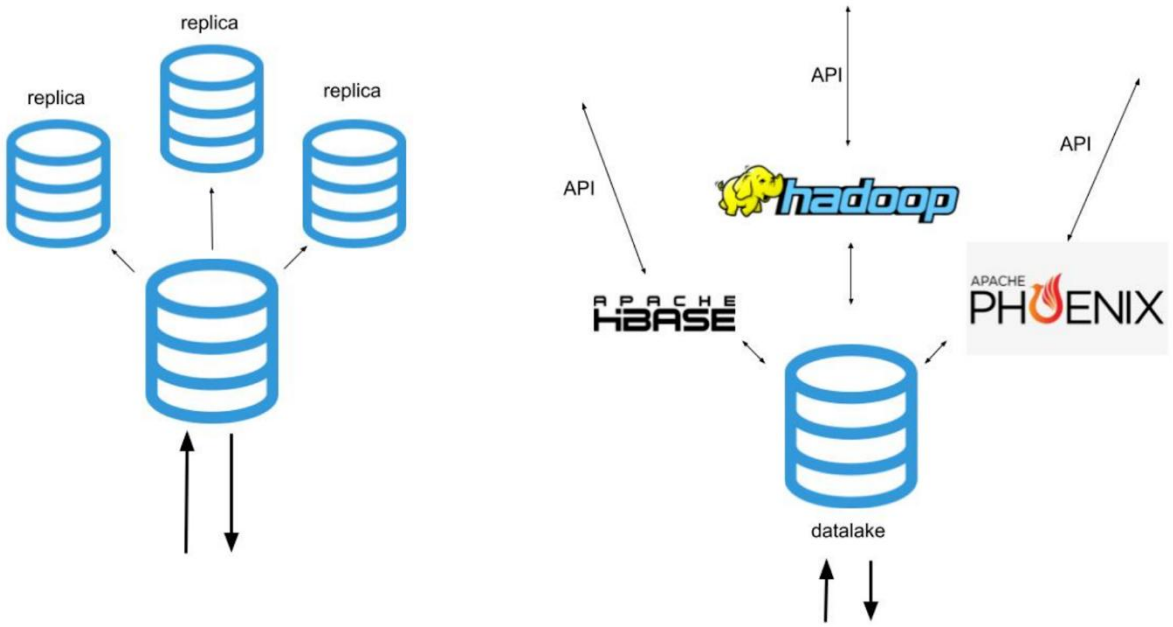

Figure 1. Architectural solution

To evaluate the effectiveness of the big data warehouse, we will use two metrics:

- the amount of memory used to ensure the fault tolerance of the repository;
- speed of execution of tasks that require requests to the repository.

By adding an internal storage conversion unit, the performance of different data warehouses can be optimized. Measurements of this are the processing time of read and write data operations. To assess the effectiveness of the developed system, simulations of several types of data warehouses were performed.

In the same repositories, different tasks give different processing times. Here is an example: HDFS as the primary data warehouse (or datalake) and HBase as the replica repository.

Assume that the data is sales of goods, records can be presented in the form of objects with the fields "Name" and "Price".

1. Take the problem of finding the sum of all fields with "Price". For this algorithm it is necessary to bypass each record and find their sum. At the same time, the calculation algorithm using HDFS through the structure of the repository and the features of the calculations in Apache Spark will spend 5.45% less time than with Hbase.

2. Take the problem of sampling the price of a particular product by its name. Hbase performs data selection tasks at a fixed time, and HDFS must go through all the data from start to finish. For comparison, the search for the price of a particular product Hbase performs for a fixed *K* ms, and HDFS can from *X* to *n* * *X* ms, where *n* is the number of elements from the beginning to the desired element in the sorted sample, *K* is the time of element selection in Hbase, *X* is the time item selection in HDFS.

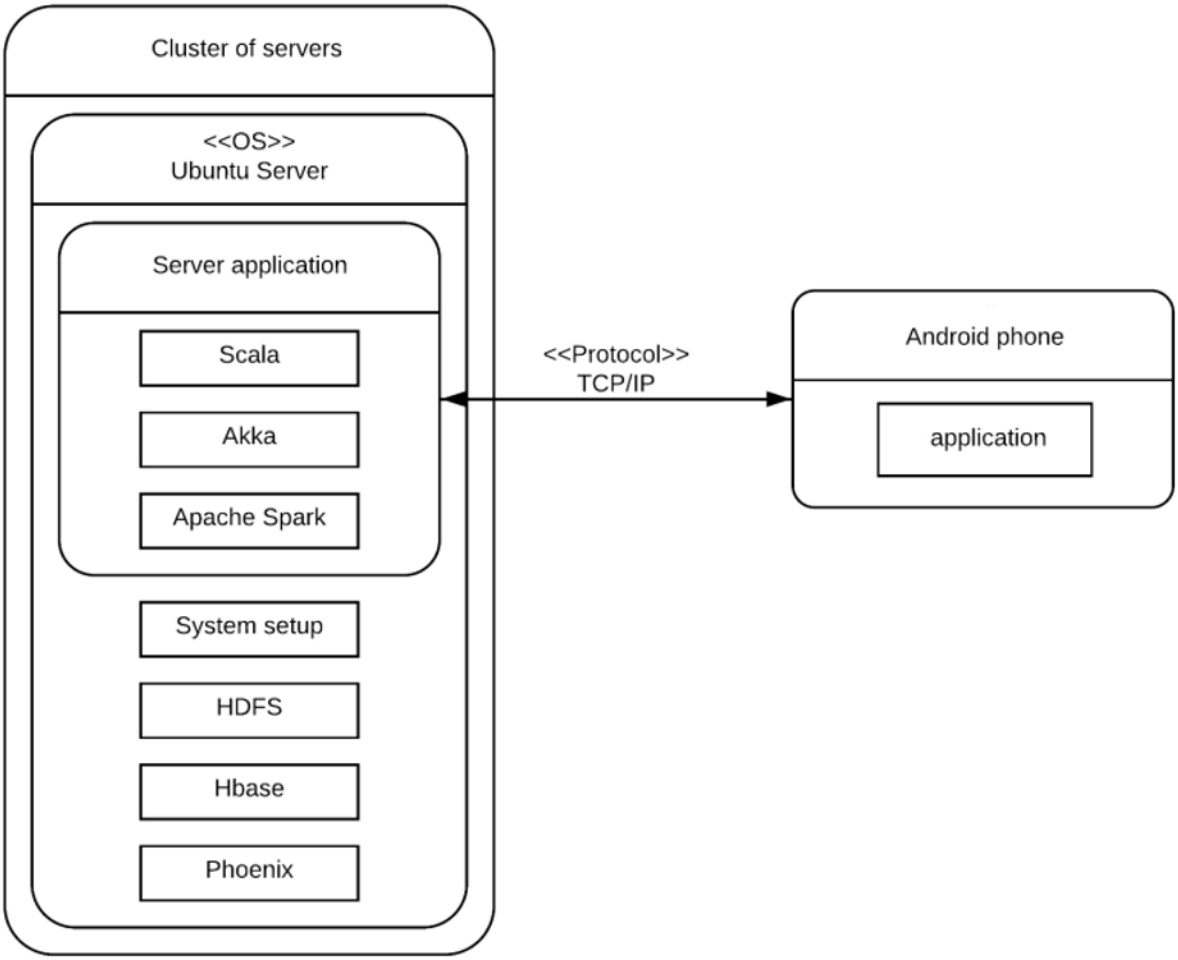

Figure 2. Block diagram of the deployment of the dynamic replication mechanism

The results of experimental studies to assess the effectiveness of the proposed solution showed that the speed of tasks increased by an average of 5%; 3% of task execution time is spent deciding which replica to use to access data.

Conclusion

Further development of methods and software for working with large data warehouses through the use of algorithms for adapting storage architecture is needed. A unit responsible for putting the conversion algorithms into operation has been added to the system for converting the internal structure of the data warehouse.

 ISSN 1560-8956

A set of programs has been developed that allows you to choose how to enter big data, collect and enter big data, choose an algorithm to change the internal structure of the repository, execute the conversion algorithm, get execution results, save the execution result.

Practical meaning:

• development of software that uses existing repository replicas (created for backup) to increase the performance of the repository as a whole;

• the advantage is that there is no need for additional space for data storage, and only the storage control module is added.

REFERENCES

1. Airflow / Apache Software Foundation // URL: https://airflow.apache.org/ [accessed 23.10.2021]

2. Bigtable / Google Inc // URL: https://cloud.google.com/bigtable/ [accessed 23.10.2021]

3. Rethink DB / Rethink DB Community // URL:https://rethinkdb.com/ [accessed 23.10.2021]

4. Redis / Redislabs // URL:https://redis.io/ [accessed 23.10.2021]

5. Mondal, A.S., Neogy, S., Mukherjee, N. et al. A survey of issues and solutions of health data management systems. Innovations Syst Softw Eng 15, 155–166 (2019). https://doi.org/10.1007/s11334-019-00336-4

6. Mohebi, A., Aghabozorgi, S., Wah, T. Y., Herawan, T., Yahyapour, R. Iterative big data clustering algorithms: a review. Software-Practice & Experience 46 (1), 107-129 (2015). https://doi.org/10.1002/spe.2341

7. Siddiqa, A., Karim, A. & Gani, A. Big data storage technologies: a survey. Frontiers Inf Technol Electronic Eng 18, 1040–1070 (2017). https://doi.org/10.1631/FITEE.1500441

8. Plase, D. A Systematic Review of SQL-on-Hadoop by Using Compact Data Formats Baltic J. Modern Computing 5(2), 233-250 (2017). http://dx.doi.org/10.22364/bjmc.2017.5.2.06

9. Nachiappan, R., Javadi, B., Calheiros, R., Matawie,K. Cloud storage reliability for Big Data applications: A state of the art survey. Journal of Network and Computer Applications 97, 35-47 (2017). https://doi.org/10.1016/j.jnca.2017.08.011

10.Milani, B., Navimipour, N. A comprehensive review of the data replication techniques in the cloud environments: Major trends and future directions. Journal of Network and Computer Applications 64, 229-238 (2016). https://doi.org/10.1016/j.jnca.2016.02.005