Proceedings of the Fifth International Conference on Trends in Electronics and Informatics (ICOEI).
IEEE Xplore Part Number:CFP21J32-ART; ISBN:978-1-6654-1571-2

Design of Distributed Network Mass Data Processing System based on Cloud Computing Technology

Jianfeng Gong
Department of Computer Engineering, Maoming Polytechnic, Maoming, Guangdong, 525000, China
Gongjianfengmm@mail.com

***Abstract*— Due to the rapid increase in the amount of data that people need to deal with computing tasks, such as distributed computing, cloud computing and other technologies are more and more concerned by most people. In recent years, with people's attention to information security and personal privacy protection, cloud computing security has gradually become the focus of researchers. In the cloud computing environment, massive distributed data storage and management and data centralization autonomy can control the data mechanism, and distribute management of redundant data and transaction processing. These management and application are generally through the distributed database for actual operation, from the user's point of view, the access process to the distributed database is transparent. In fact, from the user's point of view, a single distributed database is essentially a combination of a group of databases stored on multiple computers. This paper studies the distributed data processing system based on cloud computing technology.**

Keywords*— *Cloud Computing, Data Processing, Distributed Systems, Data Compression

I. INTRODUCTION

Big data processing systems provide standardized data parallel programming models for fixed paradigms in data analysis, thus bringing a lot of convenience to application developers [1-4]. Instead of writing similar process code over and over again, distributed processing systems focus on the customization part of the application logic. Data processing systems rely on strong assumptions about the execution process of applications, and the underlying data processing framework can provide developers with a shield-out implementation detail for scalable execution in distributed environments, and simplify the implementation of optimization technologies such as task distribution, load balancing and fault-tolerant scheduling in distributed environments [5-8].

The design idea of data parallelism is widely used in the field of traditional parallel computing, and has been proved to have significant advantages and effects in practice. For example, the multimedia extended instruction sets of universal microprocessors all support the single-instruction stream multi-data data stream (SIMD) instruction mode, which enables a single instruction to simultaneously process several basic types of data continuously placed in memory. . Loop traversing vectors and matrix elements is a common pattern in high performance computing programs. Loop body usually performs the same treatment on each vector or matrix element. Developers can label such looping blocks with no data dependencies as parfor, and compilers that support the OpenMP standard can automatically generate corresponding multithreaded parallel processing versions. The General Graphics Processing Unit (GPGPU) supports the single-instruction stream multithreading (SIMT) programming model, making it easy for non-graphics processing applications to parallel process isomorphic data items by utilizing a large number of GPU stream processing units [9-12].

According to the abstraction level provided by the programming model, the data parallel model can be further divided into three types: MapReduce model, functional data parallel model and domain specific data parallel model [13-16].

(1) MapReduce was originally designed by the Google company (Google) for its massive document analysis requirements when building a search engine. MapReduce model extends the traditional idea of data parallelism in the following two aspects: it borrows the concepts of map and reduce from list operations from Lisp and other traditional functional languages; The basic data elements processed are uniformly modeled as key-value pairs. This abstraction isolates the computational expressions of a user application from the details of massively parallel data processing on a distributed system, which is handled by the MapReduce runtime system. As shown in Figure 1, the execution process is divided into three phases:

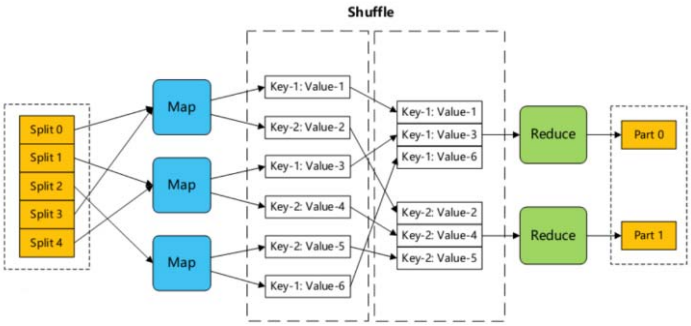

Fig. 1. MapReduce programming model calculation flow

(2) On the basis of the complete set of functional list processing operations, the new generation of data parallel programming model represented by Microsoft's Dryadlinq absorbs the form of key-value pair data representation in MapReduce model, and adds a class relational operation set with "key-grouping based" semantics as the core. Google's FlumeJava, based on the Java language implementation, provides a programming interface that is very similar to DryadLINQ. The user's FlumeJava program is automatically translated into semantically equivalent MapReduce jobs through the compilation layer.

2021 5th International Conference on Trends in Electronics and Informatics (ICOEI) | 978-1-6654-1571-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICOEI51242.2021.9452963

The Spark framework implemented by University of California, Berkeley based on Scala language is currently the most popular open source functional data parallel processing system. This design abstracts all kinds of intermediate results into different data sets which are logically independent and improves the level of abstraction of programming model.

This allows the user to truly program around the data without caring about how the underlying execution process is divided into individual jobs. Furthermore, in traditional MapReduce programming, it is cumbersome to implement the multi-table join operation of similar relational operations for multiple input data sets in a single job.

The functional model explicitly distinguishes logically distinct data sets, making the interface and implementation of join operations easier to design.

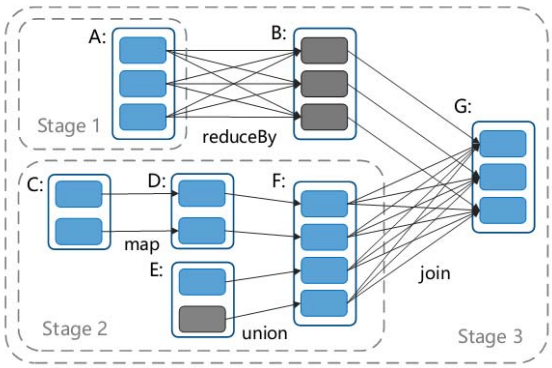

Fig. 2. MapReduce programming model calculation flow

(3) Graph is an important modeling method for various associative structures in the real world. Graph mining, as a special machine learning algorithm, has been successfully applied in scenarios such as road network analysis, protein structure analysis, web page weight ranking and social network analysis. Two parallel graph programming models, namely message sending model and shared memory model, are proposed to simplify the development of graph mining applications. Both models reduce the definition scope of graph mining applications from the original complete graph structure to a single vertex and its neighborhood, that is, users only need to define a vertex update function to deal with a single vertex.

II. The Proposed Methodology

A. MapReduce for Iterative Computing

In scientific computing, data mining, information retrieval, machine learning and other massive data processing fields, many algorithms need to use multiple iterations to achieve, such as PageRank, HITS, recursive relation query, clustering, neural network analysis, social network analysis and network traffic analysis. Since the operations performed in each iteration are the same, MapReduce can call the corresponding Map and Reduce functions through external (external to the task configurator) "drivers" during the implementation of these iterative algorithms [17-20].

PageRank is a kind of search engine based on the mutual hyperlink between the web page to calculate the page ranking technology, has been widely used in web search, network link prediction, etc., its calculation formula is:

$$PR^{(i+1)}(A)=\frac{1-d}{N}+d\sum_{v\in B(A)}\frac{PR^{(i)}(V)}{N_v} \quad (1)$$

In the face of hundreds of millions of web pages on the Internet, Google adopted the method of first giving an initial PR value to each web page, and then using the above formula to iterate until the calculated PR value converges to a relatively fixed number, namely:

$$\sum_A \left|PR_i(A)-PR_{i-1}(A)\right|<\varepsilon \quad (2)$$

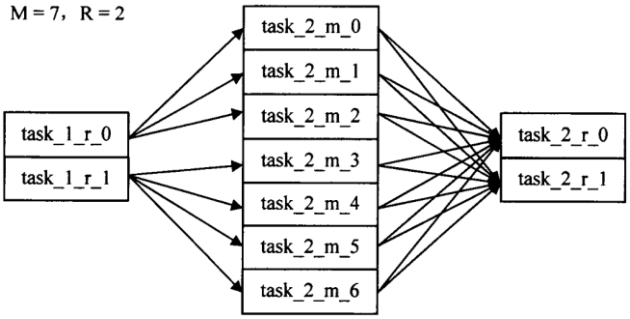

Fig. 3 MapReduce task scheduling

B. Design of Distributed Task Scheduling System

The BitDEW-MR prototype system consists of three main components: an implementation of the MapReduce programming model, a runtime system with master/slave administrative processes, and a WordCount application for benchmarking.

The design of scalable data-parallel task scheduling needs to include four core functions:

(1) Data location awareness task selection;

(2) Dynamic task assignment based on idle computing nodes;

(3) Fault-tolerant task scheduling through failover execution;

(4) Speculative redundancy scheduling for slow tasks.

The Hadoop task scheduler supports the above scheduling mechanism and further achieves two refined improvements from the engineering level:

(1) Each available CPU core of a computing node is abstracts as a Task Slot. The scheduler regards the Task Slot as the smallest resource unit that can be allocated, corresponding to a Task computing thread;

(2) The scheduler collects the task completion progress reported by the calculation node in real time, and marks the tasks that are significantly behind as slow tasks, and triggers speculative execution.

Basic functions such as Map/Reduce task distribution and execution can be easily implemented with data attributes based on BitDEW, but key features such as dynamic load balancing and speculative execution are difficult to implement. For example, the BitDew scheduler dispatches data to nodes in

a node list in a polling manner, and users cannot control the distribution of load through attributes. Meanwhile, the data-driven scheduling mode supported by BitDEW is mainly aimed at traditional CPU-intensive scientific computing applications, requiring data uploading and distribution to be tightly coupled with the processing and computation process, which can significantly prolong the execution time of the job for data-intensive applications. The data nativity scheduling commonly implemented in the MapReduce framework does not work well in this mode, and can lead to unnecessary duplicate data transfers when different computing applications process the same input data.

C. Distributed Collaborative Computing Mechanism

In the design of application system based on collaborative computing, layered management design, flexible scheduling design and easy expansion design are the key objectives of system architecture design. The realization of system architecture design mainly includes three methods: centralized, decentralized and mixed.

(1) Centralized architecture: its essence is a client-type server system architecture. Server centralized management is responsible for the control, management and scheduling of the entire computer application system, as well as other and specific computing applications related program logic design management and data management. Client services mainly refer to the input/output management that interacts with the user. Centralized collaborative architecture is simple to implement and easy to maintain, but collaborative application is highly dependent on the server, which is easy to become an obstacle.

(2) Decentralized architecture: control and management modules related to the system are dispersed in each client of the system, and each node has the same control and management status in the collaborative control system. However, the individual processing of data makes global consistency difficult to maintain.

(3) Hybrid structure combines the former two advantages, greatly reducing the working pressure of the server, with better flexibility and shorter response time.

So in the actual computer application management task scheduling, generally adopts the decentralized architecture design. In the collaborative management of data system, centralized architecture management design is adopted so as to realize the hybrid architecture design of computing application system.

III. Experiment

This section conducts a comparative test on the performance optimization effect of DECA program optimization. Five server nodes are used in the experiment, one of which serves as the master node and the rest as the computing node. Five typical Spark benchmark applications were tested: word counting (WC), logistic regression (LR), k-means clustering (Kmeans), web sorting (PR), and connected subgraph computation (CC).

The time test results are shown in Figure 4.

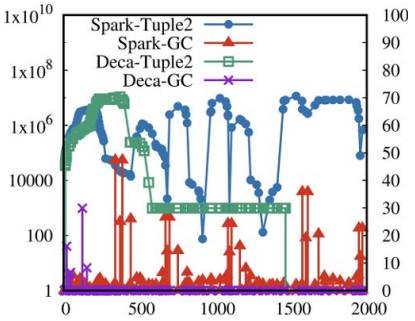

Fig. 4. GC time test results

In order to reduce GC operation overhead, the DECA version of the application reuses the memory space occupied by the aggregated values corresponding to each Key in the shuffle buffer when executing. Further, because DECA stores object data directly in byte form, it avoids the overhead of serialization and deserialization of data during I/O operations in the SHUFFLE phase.

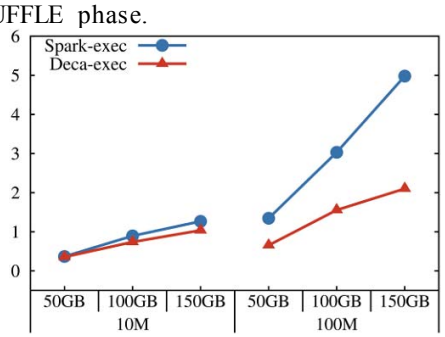

Fig. 5. Execution time comparison

The LR application execution performance comparison is shown in Figure 6.

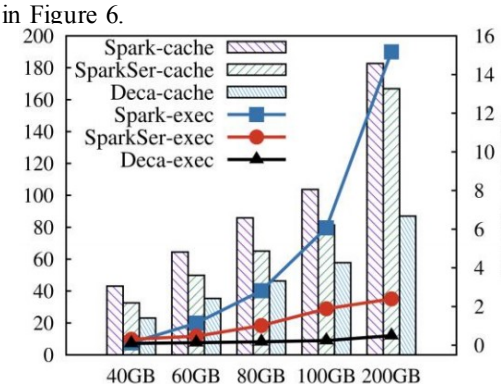

Fig. 6. LR application performance comparison

IV. Conclusion

MapReduce is an excellent programming model for large-scale data processing, which has been widely applied in many scenarios of distributed data processing. In this study, a cloud computing security model is proposed, which utilizes the homomorphic encryption protocol in secure multi-party computing protocol and combines it with the Map Reduce distributed computing framework to achieve privacy protection in cloud computing. And through the test and the above proof, we can find that the model has high efficiency, and to some extent, high security.

V. ACKNOWLEDGEMENT

This research has been financed by Maoming Municipal Science and Technology in 2016 of "Research on Intrusion Behavior Analysis and Defense Method of Cloud Enterprise Virtual Service" (20160012).

REFERENCES

[1] Lin Ziyu. Principle and application of big data technology. Beijing: People's Posts and Telecommunications Publishing House. 2017

[2] X. Zhang, Y. Zhou, Y. Ma, et al. GLMix: Generalized Linear Mixed Models For Large-Scale Response Prediction. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16). San Francisco, CA, USA. August 13-17, 2016. New York, NY, USA: ACM Press, 2016. 363~372

[3] X. Meng, J. Bradley, B. Yavuz, et al. MLlib: Machine Learning in Apache Spark. The Journal of Machine Learning Research, 2016, 17(1):1235~1241

[4] B. Burns, B. Grant, D. Oppenheimer, et al. Borg, Omega, and Kubernetes. ACM Queue, 2016, 14:70~93

[5] R. Xiong, J. Luo, and F. Dong. Optimizing Data Placement in Heterogeneous Hadoop Clusters. Cluster Computing, 2015, 18(4):1465~1480

[6] Fan Yu, Guo Huiming. Research on MapReduce dynamic task scheduling technology in heterogeneous environment. Computer application research, 2018, 35(5): 1001-3695

[7] R. O. Suminto, C. A. Stuardo, A. Clark, et al. PBSE: A Robust Path-Based Speculative Execution for Degraded-Network Tail Tolerance in Data-Parallel Frameworks. Proceedings of the 2017 Symposium on Cloud Computing (SoCC'17). Santa Clara, CA, USA. September 24-27, 2017. New York, NY, USA: ACM Press, 2017. 295~308

[8] C. Wang, B. Urgaonkar, A. Gupta, et al. Exploiting Spot and Burstable Instances for Improving the Cost-efficacy of In-Memory Caches on the Public Cloud. Proceedings of the Twelfth European Conference on Computer Systems (EuroSys'17). Belgrade, Serbia. April 23-26, 2017. New York, NY, USA: ACM Press, 2017. 620~634

[9] S. Shastri and D. Irwin. HotSpot: Automated Server Hopping in Cloud Spot Markets. Proceedings of the 2017 Symposium on Cloud Computing (SoCC'17). Santa Clara, CA, USA. September 24-27, 2017. New York, NY, USA: ACM Press, 2017. 493~505

[10] D. Garbervetsky, Z. Pavlinovic, M. Barnett, et al. Static Analysis for Optimizing Big Data Queries. Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (FSE'17). Paderborn, Germany. September 04-08, 2017. New York, NY, USA: ACM Press, 2017. 932~937

[11] H. Zhang, B. Cho, E. Seyfe, et al. Riffle: Optimized Shuffle Service for Large-Scale Data Analytics. Proceedings of the Thirteenth EuroSys Conference (EuroSys'18). Porto, Portugal. April 23-26, 2018. New York, NY, USA: ACM Press, 2018. 43~43

[12] K. Ousterhout, C. Canel, S. Ratnasamy, et al. Monotasks: Architecting for Performance Clarity in Data Analytics Frameworks. Proceedings of the 26th Symposium on Operating Systems Principles (SOSP'17). Shanghai, China. October 28-31, 2017. New York, NY, USA: ACM Press, 2017. 184~200

[13] Meng Hongtao, Yu Songping, Liu Fang, et al. Research on memory management and cache strategy of spark. Computer science, 2017, 44 (6): 31 ~ 35

[14] Wang Chenxi, Lu Fang, Cui Huimin, et al. Spark based heterogeneous memory programming framework for big data processing. Computer research and development, 2018, 55(2)

[15] V. Rosenfeld, R. Mueller, P. Tözün, et al. Processing Java UDFs in a C++ Environment. Proceedings of the 2017 Symposium on Cloud Computing (SoCC'17). Santa Clara, CA, USA. September 24-27, 2017. New York, NY, USA: ACM Press, 2017. 419~431

[16] J. Shi, Y. Qiu, U. F. Minhas, et al. Clash of the Titans: MapReduce vs. Spark for Large Scale Data Analytics. Proceedings of the VLDB Endowment, 2015, 8(13):2110~2121

[17] K. Ousterhout, R. Rasti, S. Ratnasamy, et al. Making Sense of Performance in Data Analytics Frameworks. Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI'15). Oakland, CA, USA. May 04-06, 2015. Berkeley, CA, USA: USENIX Association, 2015. 293~307

[18] I. Gog, J. Giceva, M. Schwarzkopf, et al. Broom: Sweeping out Garbage Collection from Big Data Systems. Proceedings of the 15th USENIX Conference on Hot Topics in Operating Systems (HotOS'15). Switzerland. May 18-20, 2015. Berkeley, CA, USA: USENIX Association, 2015. 2~2

[19] L. Bindschaedler, J. Malicevic, N. Schiper, et al. Rock You like a Hurricane: Taming Skew in Large Scale Analytics. Proceedings of the Thirteenth EuroSys Conference (EuroSys'18). Porto, Portugal. April 23-26, 2018. New York, NY, USA: ACM Press, 2018. 20~20

[20] J. Jiang, F. Fu, T. Yang, et al. SketchML: Accelerating Distributed Machine Learning with Data Sketches. Proceedings of the 2018 International Conference on Management of Data (SIGMOD'18). Houston, TX, USA. June 10-15, 2018. New York, NY, USA: ACM Press, 2018. 1269~1284

[21] Luo Xiaoxia, Si Fengwei, Luo Xiangyu. The influence of the structural characteristics of large map on the partition effect. Computer applications, 2018, 38(1): 1 ~ 5

[22] P. Carbone, A. Katsifodimos, S. Ewen, et al. Apache Flink: Stream and Batch Processing in a Single Engine. IEEE Data Engineering Bulletin, 2015, 38(4):28~38

[23] A. Verma, L. Pedrosa, M. Korupolu, et al. Large-Scale Cluster Management at Google with Borg. Proceedings of the Tenth European Conference on Computer Systems (EuroSys'15). Bordeaux, France. April 21-24, 2015. New York, NY, USA: ACM Press, 2015. 18~18

[24] Y. Yang, G. Kim, W. Song, et al. Pado: A Data Processing Engine for Harnessing Transient Resources in Datacenters. Proceedings of the Twelfth European Conference on Computer Systems (EuroSys'17). Belgrade, Serbia. April 23-26, 2017. New York, NY, USA: ACM Press, 2017. 575~588