Mainstream Big Data Parallel Computing System Performance Optimization

Yarong Lv[1, *]

[1]*Information Engineering College, Xi'an Mingde Institute of Technology, Xi'an, 710000, Shaanxi, China*
lvyr@mdit.edu.cn

Abstract:
In recent years, with the widespread application of the Internet and information technology in people's production and life, the amount of data generated by all walks of life every day has shown a geometric and explosive growth trend, and the real-time nature of data analysis has become increasingly high. Therefore, big data parallel computing is widely used. The main purpose of this paper is to analyze and research the performance optimization of parallel computing systems based on mainstream big data. This paper mainly analyzes the design requirements and elastic resource scheduling strategy of the big data parallel computing system, introduces the framework and module design and the main functional modules, and preprocesses the data. The experimental results show that as the parallelism of Shuffle increases, the running time of Task decreases. This is because as the degree of parallelism increases, the amount of data processed by a single task decreases and the processing speed becomes faster. However, the proportion of shuffle data read I/O waiting time is the smallest only when the shuffle parallelism is 800, which is optimal.

Keywords: *Big Data, Parallel Computing, Parallel Computing System, System Performance Optimization*

1 INTRODUCTION

The Shuffle framework of the existing big data processing system still has some problems in terms of stability and computing performance. On the one hand, with the expansion of the application data scale, the scale of the cluster is also increasing, and the node failure is frequent, which leads to the failure of Shuffle and reduces the computing performance. On the other hand, the parallelism of Shuffle also has a great impact on performance. If the parallelism of Shuffle is too small, it will be difficult to make full use of computing resources to speed up. If the parallelism of Shuffle is too large, it is easy to cause more network communication overhead. It is difficult for developers to set a reasonable Shuffle parallelism to achieve a balance between computational overhead and network communication overhead [9] [5].

In the research on big data parallel computing systems, Badri et al. believed that it is crucial to consider the dynamics of the system when designing the application placement mechanism, and modeled the energy-aware application placement problem in the edge computing system as a multi-stage stochastic program [1]. The goal is to maximize the QoS of the system while considering the limited energy budget of edge servers. A novel parallel sample averaging approximation (SAA) algorithm is designed. Sangaiah proposes that the integration of big data analytics and cognitive computing yields a new model that can take advantage of the most complex advancements in the industry and their associated decision-making processes, as well as address the glitches faced during big data analytics; EPPS problem, develop a hybrid fuzzy multi-objective optimization algorithm named NSGA-III-MOIWO [8], including non-dominated sorting genetic algorithm III (NSGA-III) and multi-objective invasive weed optimization (MOIWO) algorithm. The goal is to simultaneously minimize variance, skewness, and kurtosis as measures of risk, and maximize total expected return.

With the explosive and geometric growth of data scale, the era of massive data has arrived. How to process massive data quickly, efficiently and accurately will be an important research topic faced by a large number of researchers. Therefore, in recent years, new computing technologies, computing platforms and solutions have been proposed one after another, including high-performance single-computer computing solutions, distributed computing solutions composed of clusters, and finally distributed computing solutions composed of large-scale clusters. Because of its low cost and high efficiency, it has been applied and promoted on a larger scale in the industry.

2 DISCUSSION

2.1 Demand Analysis

Support large data query under multi-user: Since the national center stores about 2TB of data every day, it takes 11 hours to scan the data serially at a speed of 50MB/s, so parallel processing is the only way. In addition, since the national center needs to support the data query of hundreds of users at the same time, and the number of connections to the Oracle database is limited, the server often crashes or works abnormally in the case of multiple users, so a new architecture needs to be designed to solve the problem. this problem [3] [6].

Provide high-performance query efficiency: A high-performance query engine should include at least three aspects:

(1) Index, that is, a specially designed arrangement form for queries. By analyzing the hoofs to select the most commonly used queries of earthquake users, and then establishing corresponding indexes for these queries, the query performance can be optimized;

(2) Result buffering, that is, the definition of the query and the returned result are stored in advance. When the same query is submitted again, the result can be directly returned to the user to speed up the execution of the query;

(3) Parallel processing, according to the optimization method of the data layer of the p-DOT model, in the earthquake precursor network management system with multiple data copies, the query is pressed into the corresponding database Gan points, and by accessing these nodes in parallel, A speedup of at least 3 times can be achieved.

Maintain the independent integrity of each database: Since the databases of each station, regional center and national center have independent job responsibilities, such as operational duty, system management, data management, equipment management, operation status management and data collection, etc. Personnel are already familiar with these daily operations, so if the new architecture is built from scratch, the cost of personnel training is very high. In addition, since each database has a complex relational data model and huge historical legacy data, the new architecture is preferably compatible with these databases, maintaining the independent integrity of their original functions [2].

2.2 Resource Scheduling Strategy

A good elastic resource scheduling strategy for a parallel computing system needs to consider the following issues:

First, accurately locate the cluster performance bottleneck. That is, the elastic resource scheduling strategy should be able to accurately detect when the cluster has a performance bottleneck, specifically which operator has insufficient computing resources to cause the performance bottleneck, and even accurately locate the performance bottleneck caused by the cluster. Being able to find problems in a timely and accurate manner is a necessary condition for formulating a reasonable and flexible resource scheduling plan.

Second, formulate the optimal flexible resource scheduling plan. On the basis of accurately locating the cluster performance problem, the strategy should be able to formulate the optimal elastic resource scheduling plan, that is, calculating how many computing resources should be added, how to increase and other specific scheduling measures, which is the core content of the elastic resource scheduling plan .

Third, minimize the extra overhead caused by elastic resource scheduling. In fact, any operation that intervenes externally on a computational task is bound to generate additional computational and transmission overhead. Therefore, the elastic resource scheduling strategy should reduce this unnecessary overhead as much as possible, and have as little negative impact on the normal execution of computing tasks as possible. Research an online, real-time elastic resource scheduling strategy to avoid triggering during the scheduling process as much as possible. Operations such as job stagnation, restart, etc., will be the key issues and hot directions of future research in the field of elastic resource scheduling strategies. Because whether the extra overhead generated in the scheduling process can be avoided as much as possible is one of the important factors for whether a scheduling strategy can be applied and promoted in the actual industrial environment.

A good task scheduling strategy should have the following three excellent characteristics:

First, make a reasonable task scheduling plan. That is, it can accurately judge and analyze the execution of the job topology, and accurately discover the high-load nodes and low-load nodes in the cluster.

Second, try to reduce the extra overhead generated in the task scheduling process. Because in the process of task scheduling, information such as computing tasks, computing loads, and status data will inevitably be migrated, and additional computing and transmission overhead will be introduced during the migration process. A good task scheduling strategy should minimize this additional overhead, thereby reducing the performance impact of task scheduling on the entire job topology.

Third, online task scheduling is superior to offline task scheduling strategies. Because the offline task scheduling strategy requires the entire job to suspend execution during the execution process, and restart the entire job after the scheduling and migration are completed. However, in many practical application scenarios, this kind of job stagnation is unacceptable, and this kind of job stagnation itself is an additional and unnecessary time overhead [4].

2.3 Stateful Streaming Computing

In big data streaming computing, there is a special kind of application scenario, that is, stateful streaming computing. In the traditional streaming computing platform, the data is calculated and processed in the memory, and the intermediate results of the calculation are not saved. However, in a stateful streaming computing environment, state data is stored on disk or other external storage media. Once the computing task is restarted after sending an error, the state data of the previous calculation can be automatically obtained, and then the subsequent calculation can be continued from a certain moment, and the correctness and consistency of the final calculation result can be ensured during the calculation process [7]. At the same time, in the state data management of stream computing, there are the following four typical fault tolerance levels of state data management:

(1) At most once: that is, each piece of data is processed at most once, which may not be processed due to job failure, exception or data omission.

(2) At least once: that is, each piece of data is processed at least once, and may be repeatedly processed due to job failure, restart and data blocking.

(3) End-to-end exactly once: that is, it is guaranteed that in the entire link from the source to the sink, each piece of data is processed and processed only once, neither missing nor repeated processing.

2.4 Implementation of Distributed SVM Trainer

For a dataset $\{(x_i, y_i)\}_{i=1}^{n}$ containing n sample pairs, denote the division of sample numbers on m nodes as $\{B_1, \cdots, B_m\}$, then the pairwise classification learning problem can be written in the following form:

$$\min_{w_1,\cdots,w_m,z} \frac{1}{2}\|z\|_2^2 + C\sum_{j=1}^{m}\sum_{i\in B_j}\max(1-y_i w_j^T x_i, 0)^2 + \sum_{j=1}^{m}\frac{\rho}{2}\|w_j - z\|^2$$
$$\textit{subject to} \quad w_j - z = 0, j = 1, \cdots, m \tag{1}$$

where ρ is the fixed iteration step size, wj is the weight vector of the sub-dataset xBj, z is the regularization vector, and $\sum_{j=1}^{m}\frac{\rho}{2}\|w_j - z\|^2$ is used to strengthen the convergence of the algorithm.

The Lagrange transform can be written in the following form:

$$\mathcal{L}(w, z, \lambda) = \frac{1}{2}\|z\|_2^2 + C\sum_{j=1}^{m}\sum_{i\in B_j}\max(1 - y_i w^T x_i, 0)^2 + \sum_{j=1}^{m}\left(\frac{\rho}{2}\|w_j - z\|^2 + \lambda_j^T(w_j - z)\right) \tag{2}$$

where λ is a binary variable. Let $\lambda_j = \rho u_j$ and after a series of formula derivation, the (k+1)th iteration in the problem can be written in the following form:

$$w_j^{k+1} = \arg\min_{w} C\sum \max(1 - y_i w^T x_i, 0)^2 + \frac{\rho}{2}\|w - z^k + u_j^k\|_2^2 \tag{3}$$

$$z^{k+1} = \frac{\sum_{j=1}^{m}(w_j^{k+1} + u_j^k)}{m + 1/\rho} \tag{4}$$

$$u_j^{k+1} = u_j^k + w_j^{k+1} - z^{k+1} \tag{5}$$

3 EXPERIMENT

3.1 Framework and Module Design

The overall architecture of the distributed Shuffle framework based on the multi-copy data model relies on the computing engine, and is a master-slave distributed structure, including a Driver process and multiple Executor processes. The Driver process is responsible for generating the computing job DAG, computing task scheduling and job information management, and the Executor process starts multiple Task threads responsible for the specific computing tasks of each partition data. Data multi-copy shuffle is divided into three modules: metadata management, fault tolerance management, and data backup. The first two modules run in the Driver process, the data backup process and multiple Executor processes run together on the computing node, and there is only one data backup process on a computing node. The three modules are described in detail below.

3.1.1 Metadata Management

The metadata management module is responsible for the management and storage of Shuffle metadata information, as well as the life cycle of backup data.

Since this paper proposes the backup file metadata information based on the computing job structure, as described in Section 2.2.3, the backup file storage path and the mapping relationship between the backup file and the original file are generated with specific backup file naming rules and backup directory construction rules to avoid storage of metadata information for backup files.

3.1.2 *Fault-Tolerant Management*

The fault-tolerance management module provides fault-tolerance by combining the fault-tolerant method of recalculation of the computing framework with the backup data fault-tolerance mechanism of the data multi-copy model Shuffle. Specifically, fault tolerance is carried out in a prioritized manner. First, consider the fault tolerance mechanism of Shuffle with multiple copies of data, that is, backup data fault tolerance. If this method does not work, then recalculate fault tolerance.

3.2 Introduction of Main Functional Modules

Database Connector: Connects Hadoop with the database so that Map Reduce can directly access the Oracle database. It first uses the DBInput Format class provided by Hadoop to access the Oracle database, and then uses the Text Output Format class to write the obtained results into HDFS.

Data Loader: Loads the data dictionary table and the local index of the data table from the database into the data node of HDFS. For the data dictionary table, due to its high query frequency, but the amount of data is small and the table is basically fixed, all the data dictionary tables are loaded into the data cache layer; for the data table, only its local indexes are loaded into HDFS the global index layer. It can be seen that the data loader of H-DB has a small workload and will not become the bottleneck of the system.

Index generator: Using the Map Reduce programming model, the local indexes loaded into HDFS are merged into global indexes and stored in the data nodes of HDFS. When the global index is generated, the local index in HDFS will be released to reduce space. If the global index is too large, it will be automatically divided into appropriate sizes, and the index directory will be used to record the location of each block. The index directory is in XML format and is stored in the name node of HDFS. The index buffer uses the LRU (Least Recently Used) algorithm to move the most frequently used global index blocks from the HDFS data node to the name node.

Query engine: Provides different execution strategies for different query requests. When querying the data dictionary table, it will directly access HDFS; when querying the fields that have not been indexed in the data table, it will directly access the database of the National Center; when querying the fields that have been indexed in the data table, it will first access the data in HDFS. The global index layer obtains which databases should be accessed, and then accesses these databases in parallel, each database completes part of the query, and finally merges and returns the obtained results.

4 EXPERIMENTAL ANALYSIS

4.1 Data Preprocessing of H-DB System

Data preprocessing of the H-DB system includes loading the data dictionary table into the database and creating global indexes on the fields of the data table. Table 1 Shows the performance of H-DB data preprocessing.

Table 1. Data preprocessing performance of H-DB system

Preprocessing of data dictionary tables		
Load time	Table stationInfo	28s
	Table pointInfo	26s
	Total Time	54s
Index generation time	Local index loading	493.5s
	Global index creation	364.3s
	Global index chunking	109.4s
	Total time	967.2s
Global index size	Global index per block space	97.6MB
	Total space	877.4MB

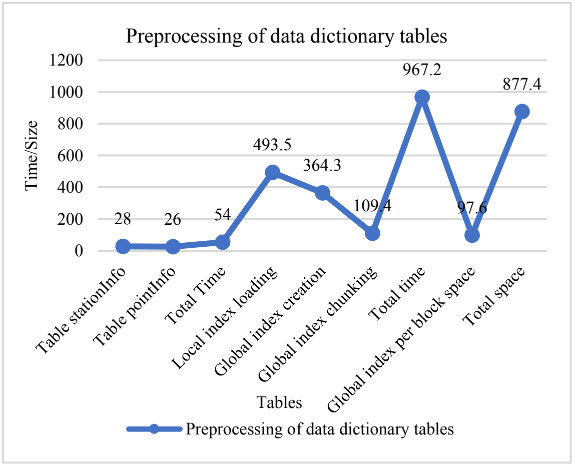

Figure 1. Data preprocessing performance analysis of H-DB system

As can be seen from Figure 1, the total loading time of the data dictionary table is less than 1 minute; it takes a total of 16 minutes to generate the global index on the data table, including loading the local index from the Oracle database into HDFS (about 8 minutes), The index is merged into a global index (about 6 minutes) and the global index is divided into an appropriate size (about 2 minutes); the space occupied by the global index is 877.4MB, which is 0.9% of the table minuteDate space (95GB).

In summary, H-DB can complete the preprocessing process of a 95GB dataset in only 17 minutes, and the loaded data dictionary table and the generated global index are only 0.9% of the original table space, so data preprocessing will not Become the bottleneck of the H-DB query system.

4.2 Shuffle Parallelism Tuning Performance Evaluation

For the same job, when the parallelism of Shuffle is 400, 800 and 2400, the comparison results of the task-level indicators are shown in Table 2. The table mainly lists the task running time and the I/O waiting time of Shuffle data reading. A job contains multiple tasks, and due to data distribution and execution logic, the difference between the task with the shortest running time and the task with the longest is often large. Therefore, the table gives the indicators within the distribution of each running time of the task, and 50th represents the median of the running time of the task.

Table 2. Comparison of multiple indicators at the Task level for jobs under different Shuffle parallelisms

Shuffle parallelism					
Task-level metrics	25th	50th	75th	90th	95th
400 Task Duration/s	132	156	180	198	210
Shuffle Read Blocked Time/s	96	114	138	150	168
%	73	73	77	76	80
800 Task Duration/s	55	60	72	72	84
Shuffle Read Blocked Time/s	30	40	48	53	60
%	55	67	67	74	71
2400 Task Duration/s	26	31	36	41	43
Shuffle Read Blocked Time/s	17	22	28	32	34
%	65	71	78	78	79

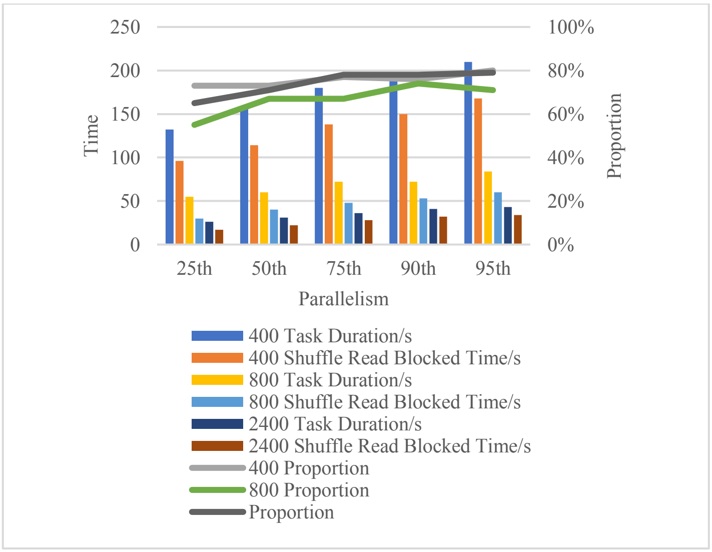

Figure 2. Comparative analysis of multiple indicators under different Shuffle parallelism

It can be seen from Figure 2 that as the parallelism of Shuffle increases, the running time of Task decreases. This is because as the degree of parallelism increases, the amount of data processed by a single task decreases and the processing speed becomes faster. However, the proportion of shuffle data read I/O waiting time is the smallest only when the shuffle parallelism is 800, which is optimal, and the job achieves a good balance in the utilization of CPU, disk, and network bandwidth.

5 CONCLUSIONS

The analysis and processing of large-scale data will be one of the important challenges for the development of information technology in the future. How to extract valuable information from large-scale and complex data and ensure the timeliness of data statistics and analysis is very important. To a large extent, it determines whether the enterprise can accurately locate the development trend of the market and find suitable business opportunities in the future development, so as to take the lead. Secondly, the value density of many data begins to decline significantly after missing the effective time of features. Therefore, the real-time processing and analysis of large-scale data will be another important challenge in the field of information technology.

REFERENCES

[1] Badri H , Bahreini T , Grosu D , et al. Energy-Aware Application Placement in Mobile Edge Computing: A Stochastic Optimization Approach[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(4):909-922.

[2] Czarnul P , Proficz J , Drypczewski K . Survey of Methodologies, Approaches, and Challenges in Parallel Programming Using High-Performance Computing Systems[J]. Scientific Programming, 2020, 2020(5):1-19.

[3] Grant Z P. Crisis and Convergence: How the Combination of a Weak Economy and Mainstream Party Ideological De-Polarization Fuels Anti-System Support:[J]. Comparative Political Studies, 2021, 54(7):1256-1291.

[4] Kaur M, Khan M Z, Gupta S, et al. MBCP: Performance Analysis of Large Scale Mainstream Blockchain Consensus Protocols[J]. IEEE Access, 2021, PP(99):1-1.

[5] Licht J, Besta M, Meierhans S, et al. Transformations of High-Level Synthesis Codes for High-Performance Computing [J]. IEEE Transactions on Parallel and Distributed Systems, 2021, 32(5):1014-1029.

[6] Lowther D , Ghorbanian V , Mohammadi M H , et al. Design tools for electromagnetic- driven multi-physics systems using high performance computing[J]. Compel, 2020, 39(1):198-205.

[7] Narayanan M , Kumar R G , Jayasundaram J , et al. Big Data Analytics and an Intelligent Aviation Information Management System[J]. Turkish Journal of Computer and Mathematics Education (TURCOMAT), 2021, 12(11):4328-4340.

[8] Sangaiah A K , Goli A , Tirkolaee E B , et al. Big Data-Driven Cognitive Computing System for Optimization of Social Media Analytics[J]. IEEE Access, 2020, PP(99):1-1.

[9] Soeung S. A REVIEW OF CAMBODIAN PRIVATE TUTORING: PARASITIC AND SYMBIOTIC FUNCTIONS TOWARDS THE MAINSTREAM SYSTEM[J]. Journal of Nusantara Studies (JONUS), 2020, 6(1):42-58.