Study on Spark performance tuning strategy based on Skewed Partitioning and Locality-aware Partitioning

Guikun Cao, Haiyuan Yu,Liujia Chang,Heng Zhao
Liaoning University of Finance and Trade, Huludao 125100, China

Abstract: Apache Spark is a large-scale data processing engine, widely used in a variety of big data analysis tasks. However, data skew and data locality issues can cause performance degradation in Spark applications. This paper investigates Spark performance tuning strategies based on Skewed Partitioning and Locality-aware Partitioning. Firstly, the influence of data skew and data localizability problems in Spark is analyzed, and then a performance tuning method combining Skewed Partitioning and Locality-aware Partitioning is proposed. Experimental results show that this method can significantly improve the efficiency of Spark jobs when dealing with large data sets, compared with traditional HashPartitioner.

Key words: Skewed Partitioning; Locality-aware Partitioning; Performance tuning; Spark; Data skew

I. Introduction

Apache Spark is a widely used large-scale data processing engine that provides a powerful and efficient platform for processing large amounts of data. However, Spark's performance can suffer from data skew and data locality issues. Data skew is the uneven distribution of data, causing some tasks to take much longer to run than others. Data locality is how data is distributed across a cluster and has a significant impact on the running time of tasks. To solve these problems, a Spark performance tuning strategy based on Skewed Partitioning and Locality-aware Partitioning is proposed in this paper.

II. The overall architecture of Spark

In Spark, the overall architecture generally includes Cluster Manager, Worker Node, node that runs tasks, Driver, control node of each application, and Executor, process that executes specific tasks on each working node. Spark comes with a Standalone resource management framework. It also supports YARN and MESOS resource management systems. FI integrates the Spark On Yarn mode. Its overall architecture is shown in Figure 1.

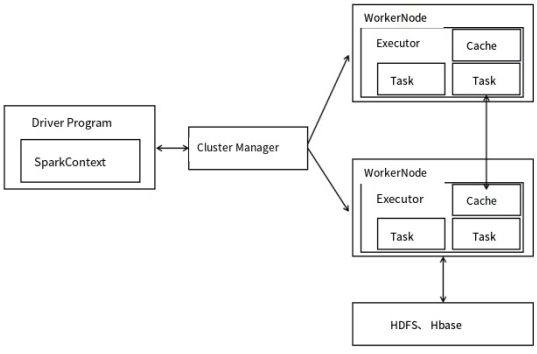

Figure 1 Spark operation architecture

The task scheduling process of Spark includes DAG generation, task partitioning, resource allocation, task execution, Shuffle operation, and result return. The task scheduling process is shown in Figure 2.

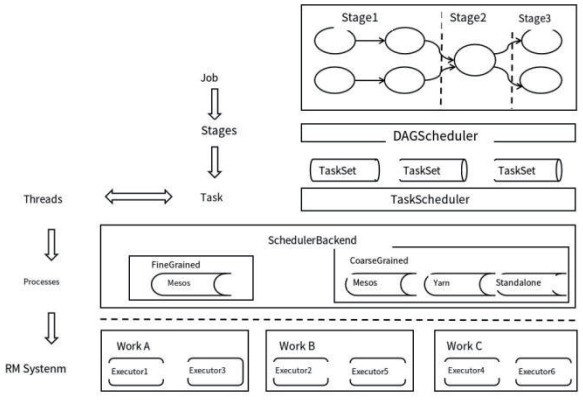

Figure 2 Spark task scheduling flow

III. Spark's data skew and local issues

1. Data skew problem of Spark

In Spark, data skew is a problem in which the amount of data in some data partitions exceeds that in other data partitions when aggregated computing or connection operations are performed. As a result, task loads on some compute nodes are unbalanced. As a result, tasks on these nodes take a long time to execute and the overall computing performance is reduced.

Data skew is usually caused by uneven data distribution, key-value conflicts, or algorithm problems. Data skew can cause problems such as long task execution time, slant data transfer, and memory overflow.

2. Data localization problems of Spark

Data in Spark is usually divided and stored according to partitions, and calculated on different nodes in the cluster. In reality, the relationship between where the data partitions are stored and the compute nodes changes dynamically. In a distributed storage system, data is usually distributed across different nodes for high availability and load balancing considerations. As a result, compute nodes may need to access data through a network, which leads to data localism issues. In addition, data is shuffled and redistributed during Spark's task scheduling and execution, so that data no longer resides in the compute node's local storage. In this case, compute nodes need to obtain the data after reshuffling and reassignment through the network, which affects the computing performance.

IV. Spark performance tuning strategy based on Skewed Partitioning and Locality-aware Partitioning

1. Skewed Partitioning performance tuning strategy

Skewed Partitioning mainly solves the problem of unbalanced computing node load caused by data skew, so as to improve the efficiency and performance of task execution. The following tuning strategies are based on Skewed Partitioning:

(1) Identifying skew keys in the pre-processing stage: sampling part of data randomly in the data set and searching for skew keys through the analysis of the sampled data. Carry out statistics on the data set, such as calculating the frequency of each key, the number of key-value pairs, etc., to find the tilt key.

(2) Partition processing on the tilt key: When initializing, set the initial number of partitions according to the distribution of the tilt key. In the process of processing, monitor the change in the amount of data of the tilt key, and dynamically adjust the number of partitions, so that the data of the tilt key can be evenly distributed to more partitions. For the tilt key, its data is distributed into multiple small partitions to reduce the amount of data in each partition and reduce the tilt phenomenon.

(3) Local aggregation based on the tilt key: the data of the tilt key is distributed to multiple partitions, and local aggregation operations are carried out on each partition to achieve load balancing and reduce the load on a single node.

(4) Dynamically adjust the degree of task parallelism: According to the data distribution of the tilt key, the degree of task parallelism can be dynamically adjusted according to the actual situation, making the load more balanced and improving the computing performance.

Through the above performance tuning strategy based on Skewed Partitioning, the data skew problem can be effectively solved, and the performance of Spark in aggregate computation and join operations can be improved. These strategies reduce the unevenness caused by data skew through repartitioning and load balancing, and improve the computational efficiency and overall performance.

2. Locality-aware Partitioning performance tuning strategy

The Spark performance tuning strategy of Locality-aware Partitioning mainly solves the problem of data Locality, that is, the data that needs to be accessed by a task on a computing node is inconsistent with the physical data location on the node where it is located. The following tuning strategies are based on Locality-aware Partitioning:

(1) Data localization management policy: Based on data localization, tasks should be assigned to the node where the data resides to avoid data transmission over the network. Based on the priority, assign tasks to the local node first, and then consider the remote node. Task inference, by analyzing the data dependencies between tasks, predicts the data localization information required by tasks in order to better schedule tasks.

(2) Data partitioning strategy: data locality, considering the distribution of data in the cluster, in the data partitioning, the adjacent or related data is assigned to the same partition to improve data localization. Partitioning is based on the key, according to the value range or hash value of the key and other information, the data is rationally partitioned to keep the data local as much as possible.

(3) Data caching and persistence: Based on the local disk cache, the hot data or frequently accessed data is cached in the local disk of the compute node, so as to quickly access and reduce the cost of network transmission. For frequently accessed data sets in the calculation process, persistence operations are carried out to persist the data to the distributed storage system, so that subsequent computing tasks can directly read the data from the storage system, reducing the cost of data transmission.

Through the above performance tuning strategy based on Locality-aware Partitioning, data localization can be improved, the overhead of data transmission over the network can be reduced, and the efficiency of task execution can be improved.

V. Experiment and result analysis

In order to evaluate the Spark performance tuning strategy based on Skewed Partitioning and Locality-aware Partitioning, the performance of the strategy is tested using typical tasks that include aggregate computation and join operations. During the experiment, the

performance tuning strategies based on Skewed Partitioning and Locality-aware Partitioning were respectively implemented and compared with the conventional partitioning schemes. The task execution time, data transmission overhead and computing node load of each strategy were recorded. Through the comprehensive analysis and comparison of the experimental results, we draw the following conclusions:

1. The performance tuning strategy based on Skewed Partitioning can significantly reduce task delay and compute node load imbalance caused by data skew problems. We observe that the Skewed data is more evenly distributed and the load of compute nodes is significantly improved under the Skewed Partitioning strategy.

2. The performance tuning strategy based on Locality-aware Partitioning can effectively improve data localization and reduce the overhead of data transmission through the network. Our experimental results show that the Locality-aware Partitioning strategy can allocate tasks to local nodes as much as possible, reducing the number of data transmission over network.

3. In large-scale data processing, the strategies based on Skewed Partitioning and Locality-aware Partitioning both show good performance results. They aim at data skew and data localization respectively, and solve the performance problems caused by these two common problems.

VI. Summary

This paper studies Spark performance tuning strategies based on Skewed Partitioning and Locality-aware Partitioning. Through experiment and result analysis, it is found that these two strategies are effective in solving the problems of data skew and data locality, thus improving the performance of Spark. However, these strategies may behave differently on different application scenarios and data sets, so they need to be selected and adjusted according to the specific situation.

References:

[1] Wei Jin,Peng Liu,Ru Li. A review of data skew problems based on Spark [J]. Computer Science, 2021, 48(2): 89-97.

[2] Wanhang Xie,Hongwei Yuan,Fanyi Liu, etal. Overview of Spark SQL Optimization Algorithm [J]. Computer Engineering and Design, 2020, 41(1): 7-13.

[3] Weichao Guo,Yong Yang,Bo Pan, etal. Research review of Spark framework in Big Data analysis [J]. Computer Engineering and Design, 2019, 40(5): 1052-1060.

[4] Fei Hu. Research on Optimization of large-scale Data Processing Architecture Based on Spark [J]. Modern Computer, 2018(22): 72-74.

[5] Yichen Fang,Liping Zhang. Review of data analysis and processing methods based on Spark [J]. Well Logging Technology, 2018, 42(1): 69-75.

[6] Hongjie Chen,Yu Huang. A review of data analysis technology based on Apache Spark. Computer and Digital Engineering, 2017, 45(7): 1321-1329.