A Fine-Grained Task Monitoring Mechanism in Spark Platform

Cheng Chen[1, a], Fei Liu[1, b], Guangrui Li[2, c], Xiang Chen[1, d] and Ying Hou[1, e]

[1]Faculty of Information Technology, Beijing University of Technology, Beijing100124, China;

[2]NO.10 Building, Suzhou Software Park, No.78, Keling Road, China

[a]ccheng@emails.bjut.edu.cn, [b]liufei@emails.bjut.edu.cn, [c]liguangrui@cmss.chinamobile.com, [d]chenxiang@emails.bjut.edu.cn, [e]HouYing@emails.bjut.edu.cn

Keywords: Spark, In-memory Computing, Fine-grained, Task monitoring.

Abstract. The existing coarse-grained monitoring mechanism in Spark has much difficulty in analyzing and locating the bottleneck during the execution of tasks. Aiming to solve this problem, we divide the execution process of Spark tasks into several subphases, and propose a fine-grained subphases-based monitoring mechanism. The fine-grained monitoring mechanism can help users to understand the detail execution status of Spark tasks and be beneficial to analyze and locate the performance bottleneck in Spark.

1. Introduction

We are at the beginning of a Big Data era, in which both industry and academia are undergoing a profound transformation with the use of large-scale datasets. Currently, many systems for large data processing are presented. In-memory computing platform is a cutting-edge in the data processing system. Spark, the representative of the distributed memory computing platform, has been widely recognized by industry and academia [1, 2].

Apache Spark is a large data parallel computing framework based on memory computing. Spark application has a complicated computing logic, leading a job-stage-task execution model. A stage is a set of parallel tasks all computing the same function that need to run as part of a Spark job, task is a fundamental unit of resource consumption and execution. A Spark task is usually composed of multiple subphases and each phase has the different computing logic. At present, the existing coarse-grained monitoring mechanism in Spark takes each job as the monitor unit, and hence, lack of the detailed information of each stage and corresponding task, which leads to much difficulty in analyzing and locating the performance bottleneck during the execution of task. For Spark users, task monitoring information is the fundamental information of locating exception and system bottlenecks [3]. Therefore, a fine-grained monitoring mechanism can contribute much to the performance analysis and optimization of Spark.

2. Principle of Spark

2.1 Spark Overview.

Spark was developed by the University of California at Berkeley AMPLab, which is based on memory computing large data parallel computing framework. It introduces Resilient Distributed Datasets (RDD) as an abstract expression of distributed data sets [4]. Besides, Spark provides a wealth of advanced data manipulation primitives for building complex data processing logic and utilizes DAG (Directed Acyclic Graph) to describe the dependencies between operations. Spark divides DAG into multiple stages that can be executed in parallel depending on dependencies and hides the details of Spark's distributed parallel execution.

2.2 Task Phases Division.

Spark stages are created by breaking DAG at shuffle boundaries, which introduce a barrier where the behind stage must wait for the previous stage to finish to fetch outputs. There are two types of stages: Shuffle Map Stage, which writes map output files for a shuffle and Result Stage, which read those files after a barrier. Each of them contains a set of parallel tasks, computing the same function,

called Shufle MapTasks and Result Tasks respectively. The two types of tasks are similar to the Map tasks and Reduce tasks in Hadoop platform respectively [5]. Depending on the behavior characteristics of the task execution, task can be divided into several small phases. In the following, we take a two-stage job which similar to Map/Reduce job in Hadoop as an example to illustrate the detailed task phase's division.

As shown in Fig.1, the tasks of the two-stage job can be refined into six small phases. When the Map task are executed, each data record is processed in a pipelined manner, the process can be described as follows: First, reading the data to be processed by the task, and then processing the data, finally storing the processing results. Therefore, the Map task can be split into three subphases: Map Read, Mapper and Shuffle Write. When the Map tasks are fully executed, the Reduce task begins to be scheduled. Reduce task pull the data from all nodes that have executed the Map tasks, and then perform operations such as aggregation and sorting, finally performing the Reducer operations according to the application logic. In summary, the Reduce task can be refined into as following subphases: Shuffle Fetch, Reduce Aggregate and Reducer.

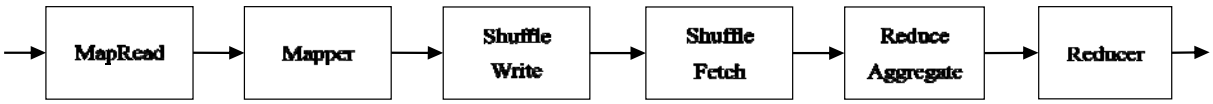

Fig.1 Phases of data processing in Spark

3. Design and Implementation

In this paper, the design and implementation of the fine-grained monitoring mechanism is based on the Spark's Standalone mode. In this mode, the responsibility of the Driver is to manage the status of the Executor and monitor the execution of the task. Executor is the basic processing unit that handles the task. On the basis of the original Spark (version 1.4) components, we modify and extend corresponding components. We use the method of inserting the pile to obtain the detailed index of tasks, to achieve the purpose of fine-grained monitoring.

3.1 Hierarchical Design.

Our fine-grained monitoring mechanism is designed with a hierarchical structure, as Fig.2 shows. The layers in the architecture are monitoring layer, communication layer and user layer respectively. Monitoring layer is mainly responsible for collecting task metrics in real time and summarizing the metrics. Communication layer is designed for the transmission of metrics between nodes and the communication between user layer and monitoring layer. User layer is responsible for receiving the user request, providing the user with a visual interface.

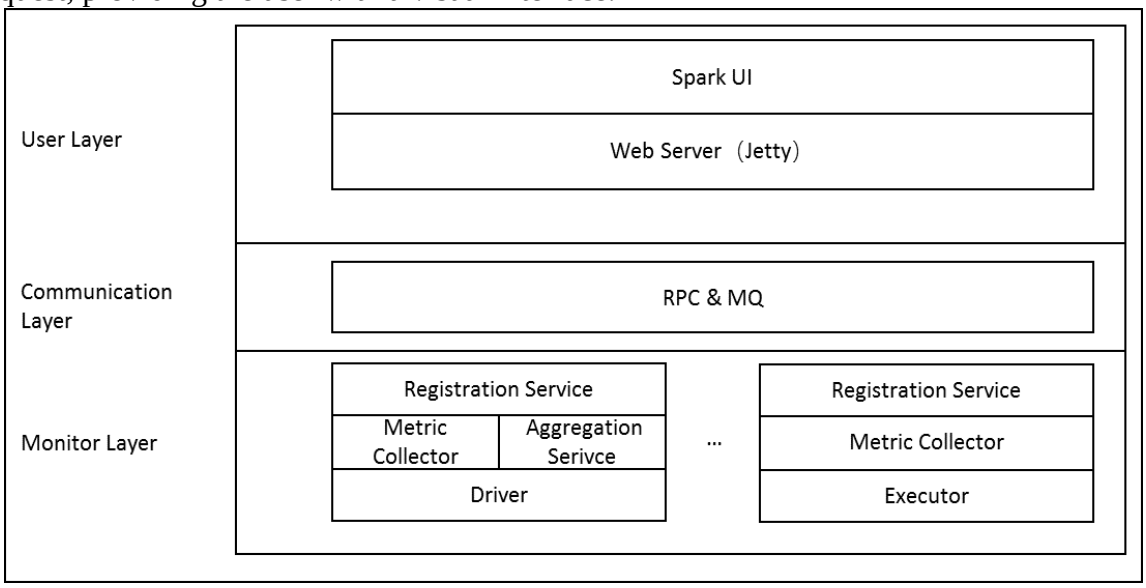

Fig.2 Hierarchy of fine-grained task monitoring system

3.2 System Implementation.

1) Monitoring Layer

Monitoring Layer is at the heart of the fine-grained monitoring mechanism, which consists of three components: registration service component, metric collector, and aggregation service component. The registration service component runs on the Spark node as a daemon thread, responsible for registering monitoring event, monitoring source and monitoring sink. The Metric Collector runs on each Executor node, and the primary responsibility is to collect the metrics information for the registration event during the execution of the Spark application. The aggregation service runs on the Driver node, and its primary role is to aggregate the task information collected by the metric collector component.

The monitoring information collected by Monitoring Layer mainly includes static information and dynamic information. Static information, such as task identification (task id), etc. can be obtained from the original Task Metrics in Spark directly, so this paper focuses on the obtaining of dynamic information. The dynamic metrics include the type of task, and different types of tasks have different metrics. The following will introduce the task metrics that need to be collected according to the task phases that is subdivided in Section 1.2. Metrics can be divided into two categories: the original metrics and calculation metrics, the original metrics can be obtained by a single registration indicator, the calculating metrics need to be integrated with multiple original metrics and obtained by calculation.

Table 1 Task Metrics

Sub-Stage	Metric	Type	Description
MapRead	_bytesRead	original	Number of bytes read for the map
	_recordsRead	original	Number of records read for the map
	_readTime	original	Total time of fetching data
	_progress	calculating	The percentage of data that has read
Mapper	_computingTime	calculating	Calculation time of mapper
ShuffleWrite	_shuffleBytesWriten	original	Number of bytes written for the shuffle
	_shuffleWriteTime	original	Total time of writing data
ShuffleFetch	_remoteBlocksFetched	original	Number of blocks fetched in this shuffle from remote
	_localBlocksFetched	original	Number of blocks fetched in this shuffle from local
	_remoteBytesFetched	original	Number of bytes fetched in this shuffle from remote
	_localBytesFetched	original	Number of bytes fetched in this shuffle from local
	_totalBytesFetched	original	Fetched time in this shuffle from local
	_remoteDuration	original	Fetched time in this shuffle from remote
	_progress	calculating	The percentage of data that has been readed
ReduceAggregate	_aggregateTime	original	Time of aggregation
	_spillBytes	original	Number of on-disk bytes spilled by this task
Reducer	_duration		Calculation time of reducer

2) Communication Layer

Communication layer is responsible for passing metrics between nodes, and the communication between user layer and monitoring layer. Communication layer is implemented based on the existing heartbeat mechanism and message queue. In order to avoid affects to the original Spark, we did not change the original way of communication, only changed the communication tuple information. As shown in Fig 3, during the execution of tasks in Executor, the metrics need to be collected will be recorded in a data structure, and be packaged into a heartbeat, reported to the Driver. After receiving the report information, the driver extracts the monitoring information and sends the information to the message queue, and then waits for the summary service to be processed. After the summary service

receives the reported information, aggregating the metrics according to the task type. Finally, it is provided to the user layer.

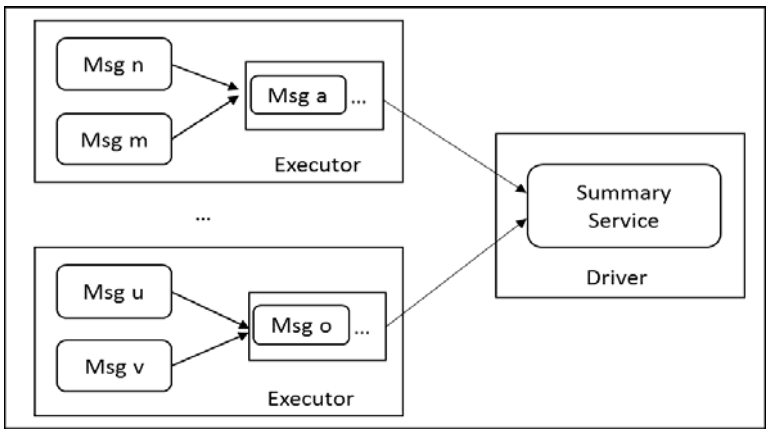

Fig.3 Schematic of Communication

3) User Layer

User layer to provide users with visual monitoring interface. In this System, User layer is displayed in Web mode, which use the built-in Jetty as the Web Server and take the servlet as the request controller. On the basis of the display in original Spark, we add the data metrics that displayed in WebUI with the concept of task and provide users with a richer system metrics. This article uses the Simple Skewed Group By Test provided in Spark as a test load to verify system functionality (The experiment was carried out in a cluster with 6 nodes, 2.2GH Xeon CPUs, 16GB RAM), observe the display of the metric information in the subphases on the Web monitoring page and intercept the Reduce task execution information as a result of the show. As shown in Fig.4, the task progress information, the local and remote shuffle data and other metrics are displayed normally. Through the analysis of the collected metrics, users can clearly understand the cost and task execution bottlenecks of each stage.

Index ▲	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	Progress	Compute Time	GC Time	Shuffle Local Read Time	Shuffle Local Read Size/Records	Shuffle Remote Read Time	Shuffle Remote Read Size / Records
0	26	0	SUCCESS	NODE_LOCAL	1 / centos28	2017/03/09 21:55:02	28 s	100%	5.1 s	2 s	768 ms	58.9 MB/ 3619472	20.2 s	117.8 MB / 7238946
1	25	0	SUCCESS	NODE_LOCAL	1 / centos27	2017/03/09 21:55:02	14 s	100%	2.8 s	1 s	428 ms	24.4 MB/ 1807104	9.3 s	58.9 MB / 3620427
2	24	0	SUCCESS	NODE_LOCAL	1 / centos27	2017/03/09 21:55:02	14 s	100%	2.7 s	1 s	437 ms	24.5 MB/ 1812536	9.4 s	58.9 MB / 3618972

Fig.4 Tasks Info

4. Conclusion and Future Work

This paper proposed a fine-grained monitoring mechanism that divides the execution of task in Spark into several subphases and monitor the sub-phases respectively. We implement the fine-grained monitoring mechanism based on the original Spark monitoring system, and demonstrate its features. This system makes it easier for users to understand the execution procedure of Spark job and analyze the bottleneck factors that exist during the execution of the Spark job.

In the future work, we would like to add another feature in our system: analyzing system bottleneck automatically, extending our system to latest version of Spark and extending more system resource metrics.

References

[1]. Information on: spark.apache.org.
[2]. M. Zaharia, M. Chowdhury, S. S. Michael J. Franklin, and I. Stoica, Spark: Cluster Computing with Working Sets. Hot Cloud, 2010, 6.

[3]. Ganesh Ananthanarayanan, Michael Chien-Chun Hung, XiaoqiRen. GRASS: Trimming Stragglers in Approximation Analytics[C], NSDI, 2014, 289-302
[4]. Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing[C]// Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012:141-146.
[5]. Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters. [C]// Conference on Symposium on Opearting Systems Design & Implementation. 2004:107-1.