Applied Soft Computing 38 (2016) 1109–1118

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Scheduling algorithm based on prefetching in MapReduce clusters

Mingming Sun*, Hang Zhuang, Changlong Li, Kun Lu, Xuehai Zhou

Computer Science University of Science and Technology of China, Hefei, China

ARTICLE INFO

Article history:
Received 1 January 2015
Received in revised form 24 March 2015
Accepted 19 April 2015
Available online 1 May 2015

Keywords:
Data locality
MapReduce
Prefetching
Task scheduler
Memory
Big data

ABSTRACT

Due to cluster resource competition and task scheduling policy, some map tasks are assigned to nodes without input data, which causes significant data access delay. Data locality is becoming one of the most critical factors to affect performance of MapReduce clusters. As machines in MapReduce clusters have large memory capacities, which are often underutilized, in-memory prefetching input data is an effective way to improve data locality. However, it is still posing serious challenges to cluster designers on what and when to prefetch. To effectively use prefetching, we have built HPSO (High Performance Scheduling Optimizer), a prefetching service based task scheduler to improve data locality for MapReduce jobs. The basic idea is to predict the most appropriate nodes for future map tasks based on current pending tasks and then preload the needed data to memory without any delaying on launching new tasks. To this end, we have implemented HPSO in Hadoop-1.1.2. The experiment results have shown that the method can reduce the map tasks causing remote data delay, and improves the performance of Hadoop clusters.

1. Introduction

MapReduce [8] has been highly successful as a parallel distributed processing framework in implementing big data applications on commodity cloud computing plateforms such as Amazon EC2 and Windows Azure. MapReduce divides a computation into multiple small tasks and lets them run on different machines in parallel. It enables hiding the details of the underlying parallel processing to provide a simple programming interface for developing distributed application. There are many different implementations of MapReduce framework such as Hadoop [20], Disco [14], Phoenix [22], etc.

In most state-of-the-art cloud cluster systems, a key challenge is to increase the utilization of MapReduce clusters. If map tasks are scheduled to nodes without input data, these tasks will issue remote I/O operations to copy the data to local nodes. This data transfer delay is primarily on the execution time cost of map phase, while map phase often dominates the execution time of the MapReduce jobs. So data locality becomes one critical factor to affect performance of MapReduce framework. In practice, not only clusters are shared by multiple users, but also there is a limitation in the number of nodes a user can use. In this case, it is not easy to guarantee good data locality to all map tasks. The process will cause remote data access delay, thus degrading the performance of MapReduce. Workloads from Facebook and Microsoft Bing datacenters show that this remote I/O operations phase constitutes 79% of a job's duration and consumes 69% of the resources [3]. So in this paper, we focus on the optimizing the map phase. Obviously, the performance of MapReduce clusters is closely tied to its task scheduler. Zaharia [23,24] proposed a delay scheduling algorithm to reduce the map tasks executing remote I/O operations. A next-k-node scheduling method is proposed to improve the data locality [28]. However, in both methods task fairness withered as the cost.

Data prefetching [5] is a data access latency hiding technique, which decouples and overlaps data transfers and computation. And machines in MapReduce clusters have large memory capacities, which are often underutilized; the median and 95 percentile memory utilizations in the Facebook cluster are 10 and 42%, respectively [3]. In light of this trend, we investigate memory locality to speed-up MapReduce jobs by prefetching and caching their input data. Seo et al. [16] designed a intra-block and inter-block prefetching scheme to improve data locality of map tasks, which are assigned to nodes without input data. A data prefetching mechanism in heterogeneous or shared environments [9] is proposed. However, both techniques not only cannot reduce the occurrence of such map tasks, but also do not consider the remote access delay of the first data block split.

The prefetching accuracy is the key factor that affects performance. In MapReduce clusters, task scheduler determines the mapping between tasks and nodes. To this end, we design HPSO, a

* Corresponding author. Tel.: +86 15862451676.
E-mail addresses: mmsun@mail.ustc.edu.cn (M. Sun), zhuangh@mail.ustc.edu.cn (H. Zhuang), liclong@mail.ustc.edu.cn (C. Li), local@mail.ustc.edu.cn (K. Lu), xhzhou@ustc.edu.cn (X. Zhou).

http://dx.doi.org/10.1016/j.asoc.2015.04.039
1568-4946/

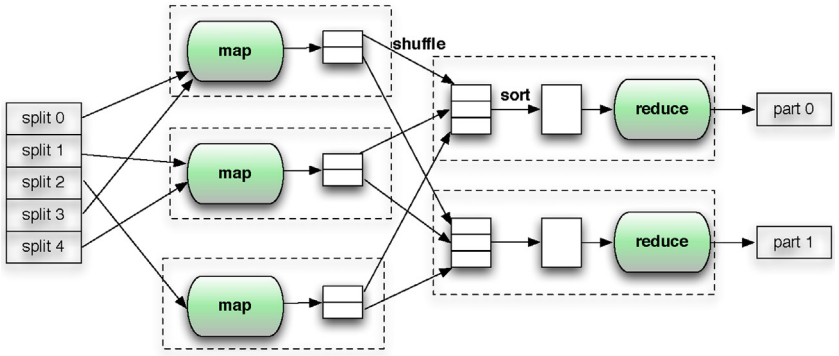

Fig. 1. The MapReduce framework.

prefetching service based task scheduler to improve performance for MapReduce clusters. The method first predicts the execution time of map tasks and further evaluates the sequence that nodes free busy slots. According to this node sequence, HPSO predicts and assigns the most suitable map tasks to nodes ahead of time. Once such scheduling decisions are made, nodes preload the related input data from remote nodes or local disk to memory before tasks is launching. In this way, input data prefetching is carried out concurrently with data processing, thus data transfer overhead is overlapped with data processing in the time dimension. In summary, in this paper we claim following contributions:

- We provide a novel prefetching mechanism to coordinately manage prefetching input data blocks.
- We exploit task scheduler to determine what and when to prefetch. This method can greatly improve the efficiency of map tasks.
- We have built the High Performance Scheduling Service(HPSO), which combines the task scheduler, prediction, and prefetching mechanism together to exploit data locality and reduce network overhead. To evaluate its performance, we have built a Hadoop cluster system. Compared with the native Hadoop, our implementation of HPSO has demonstrated performance.

The remainder of the paper is structures as follows. Section 2 introduces some technology background necessary to understand the MapReduce and scheduler, and motivation. Section 3 describes the prefetching technique. The design and implementation of HPSO are illustrated in Section 4. Section 5 evaluates HPSO. Related work is described in Section 6. Finally, conclusion is given in Section 7.

2. Background and motivation

In this section, we introduce the background of MapReduce programming framework, hadoop scheduler mechanism and data locality problem.

2.1. MapReduce programming framework

Computations in MapReduce framework are divided into map and reduce phases, separated by an internal grouping of the intermediate results as shown as Fig. 1. The power of MapReduce is that the map and reduce functions are executed in parallel over hundreds or thousands of processors with minimal effort by the user. After user submits a job, MapReduce jobs run as follows. Firstly, input data is divided into several fixed-size splits, each of which runs a map task. Then after all map tasks are finished, the intermediate data is reassigned to reduce tasks according to different keys generated in map phase. Reduce phase can be divided into three parts, shuffle, sort and reduce function. The shuffle phase transfers intermediate data generated by map tasks to the corresponding reduce tasks. The sort phase merges all the intermediate data belonging to the reduce task and maintains sort. Then reduce function is called for each key in the sort phase, and directly writes output to distributed file system.

In our paper, we chose Hadoop since it is an open-source implementation of MapReduce model. Furthermore it has been used by many companies such as Yahoo!, Amazon, Facebook duo to its high performance, reliability, and availability. Specially, Hadoop manages computing resources by the term of slot, the basic resource allocation unit. The precise number of slots of each slave in Hadoop cluster depends on the number of cores and the amount of memory. Each slave node provides a number of slots for map tasks and a number of slots for reduce tasks, and these are set independently. Each slot can only run a task simultaneously.

Hadoop Distributed File System (HDFS) [17] is a highly fault-tolerant distributed file system designed to provide high bandwidth for MapReduce by replicating and partitioning files across many nodes. Files in HDFS are automatically partitioned to chunks (64 Mb by default). In MapReduce framework, each map function executes on one chunk.

2.2. Hadoop scheduler

Fig. 2 illustrates the work mechanism of Hadoop running a MapReduce job. Hadoop clusters have one JobTracker, which coordinates the job run, and a number of TaskTrackers, which are in charge of running jobs and periodically heartbeat the JobTracker. First, when JobTracker receives a client submitted job, it puts it into an internal queue from where the job scheduler will pick it

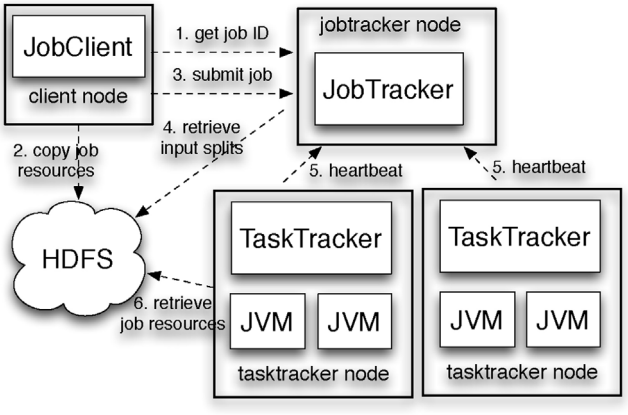

Fig. 2. How Hadoop runs a MapReduce job.