Research on Scheduling Scheme for Hadoop clusters

Jiong Xie[1,2], FanJun Meng[3], HaiLong Wang[3], JinHong Cheng[1],
Hongfang Pan[1], and Xiao Qin[2]

[1] Inner Mongolia electric power information and communication center, China
[2] Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849-5347 Email: {Jzx0009, Xzq0001}@auburn.edu
[3] Computer & Information Engineering College, Inner Mongol Normal University, China

Abstract. In this paper, we import a prefetching mechanism into MapReduce model while retaining compatibility with the native Hadoop. Given a data-intensive application running on a Hadoop cluster, our approach estimates the execution time of each task and adaptively preloads an amount of data to the memory before the new task is assigned to the computing node.

Keywords: MapReduce; Prefetching; Schedule.

1 Introduction

In the past decade, the MapReduce framework has been employed to develop a wide variety of data-intensive applications in large-scale systems. In this paper, we focus on a new reshuffling scheme to further improve Hadoop's system performance.

In the past decade, the MapReduce framework has been employed to develop a wide variety of data-intensive applications in large-scale systems. In this paper, we observe the data movement and task process pattern of MapReduce, and design a prefetching mechanism to solve this problem so as to improve the performance.

Four factors make predictive scheduling and prefetching desirable and possible:
1) the underutilization of CPU processes
2) the growing importance of MapReduce performance
3) the Hadoop file distribution model offer data storage information
4) interaction between master node with slave nodes.

Our preliminary results show that CPU and IO workload are underutilized while they are working on a data-intensive application. In MaReduce model, HDFS(Hadoop Distributed File System) is tuned to support large files. HDFS spilt the big files to several partitions and distribute to hundreds of nodes in a single cluster, the size of which by default is 64MB. However, because of the big block size, the transfer time of data access dominate the whole process time, and the stall for IO is also a significant factor in the process time. This suggests that prefetching is needed to boost IO performance and improve the performance.

The second factor encourage our prefetching mechanism is that ever-faster CPUs are processing data more quickly than the data provided. Simply increasing the cache does not improving the IO-subsystem performance and CPU performance [3]. In the MapReduce model, before a computing node launches a new task, it should ask the master node to assign one, and then the master node let the computing node know what the next task is and where the required data is located.

The computing node does not gather the required data and process until receiving this information. In this way, the CPU should wait a long period while the computing node communicates with the master node. Prefetching strategies are needed to parallelize these workloads so as to avoid the idle point.

2 Design and Implementation

We design a predictive scheduler -a flexible task scheduler- to predict the most appropriate task trackers to which future tasks should be assigned. Once the scheduling decisions are predicted ahead of time, DataNodes can immediately start loading < key; value > pairs. Our predictive scheduler allows DataNodes to explore the underutilized disk bandwidth by preloading < key; value > pairs. The algorithm for predicting stragglers in the native Hadoop is inadequate, because the original algorithm uses a single heuristic variable for prediction purpose. The native Hadoop randomly assigns tasks and mispredicts stragglers in many cases. To address this problem, we develop a predictive scheduler by designing a prediction algorithm integrated with the native Haddop. Our predictive scheduler seeks stragglers and predicts candidate data blocks. The prediction results on the expected data are sent to corresponding tasks. The prediction decisions are made by a prediction module during the prefetching stage.

We seamlessly integrate the predictive scheduler with the prefetching module. Below let us describe the structure of the prefetching module, which consists of a single prefetching manager and multiple worker threads. The role of the prefetching manager is to monitor the status of worker threads and to coordinate the prefetching process with tasks to be scheduled.

When the job tracker receives a job request from a Hadoop application, the job tracker places the job in an internal queue and initializes the job [5]. The job tracker divides a large input file to several fixed-size blocks and creates one map task for each block. Thus, the job tracker partitions the job into multiple tasks to be processed by task trackers. When the job tracker receives a heartbeat message from an idle task tracker, the job tracker retrieves a task from the queue and assigns the task to the idle task tracker. After the task tracker obtains the task from the job tracker, the task is running on the task tracker.

Figure **1** shows that the Hadoop system applies the following basic steps to launch a task. First, the job tracker localizes the job JAR by copying the job from the shared file system to the task tracker's file system. The job tracker also copies any required files by the Hadoop application from the jobtracker node distributed cache to the local disk. Second, the job tracker creates a local working directory for the task, and un-jars

the contents of the JAR into this directory. Last, an instance of TaskRunner is created to launch a new Java Virtual Machine to run the task.

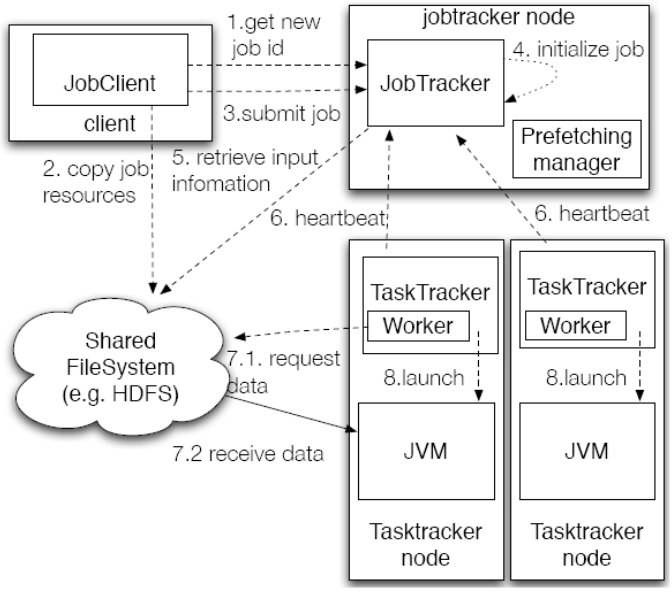

Fig. 2. The basic steps to launch a task in Hadoop

In our design, the above task launching procedure is monitored by the prediction module. Specifically, the prediction module in the scheduler predicts the following events. 1) finish times of tasks currently running on nodes of a Hadoop cluster; 2) pending tasks to be assigned to task trackers; and 3) launch times of the pending tasks.

Upon the arrival of a request from the Job tracker, the predictive scheduler triggers the prefetching module that forces preload worker threads to start loading data to main memory. The following three issues must be addressed in the prefetching module.

When to prefetch. The prefetching module controls how early to trigger prefetching actions. Before a block finishes, the subsequent block will be loaded into the main memory of the node. The prediction module assists the prefetching module to estimate the execution time of processing each block in a node. Please note that the block processing time of an application on different nodes may vary in a heterogeneous cluster. The estimates are calculated by statistically measuring the processing times of blocks on all the nodes in a cluster. This statistic measuring can be performed offline.

What to prefech. The prefetching module must determine blocks to be prefetched. Initially, the predictive scheduler assigns two tasks to each task tracker in a node. When the prefetching module is triggered, it proactively contacts the job tracker to seek required information regarding data to be processed by subsequent tasks.

How much to prefetch. When the prefetching action is triggered, the prefetching module automatically fetches data form disks. Due to the large block size in HDFS, we intend not to make our prefetching module very aggressive. Thus, there is only one block being prefetched at a time.

The most important part of the prefetching work is to synchronize two resources in the MapReduce system: the computing task and the data block. The scheduler in the MapReduce always collects all the running task information and constructs a RunningTaskList. It separately caches the different types of tasks in a map task list and a reduce task list. The worker thread in each running node can finish the data loading job all by itself before the task is received.

3 Related Work

MapReduce[1] is useful in a wide range of applications including: distributed grep, distributed sort, web link-graph reversal, web access log stats and so on. Moreover, the MapReduce model has been adapted to several computing environments like multi-core and many-core systems.

Scheduling Algorithms Performance of Hadoop systems can be improved by efficiently scheduling tasks on Hadoop clusters. Many scheduling algorithms might be adopted in Hadoop. For example, [2] implemented a new scheduling algorithm called LATE in Hadoop to improve Hadoop system performance by speculatively executing tasks that decrease responses time the most.

4 Conclusion

In this paper, we observed that the task processing procedure in Hadoop may introduce data transfer overhead in a cluster. Realizing that the data transfer overhead is caused by the data locality problem in Hadoop, we proposed a predictive scheduling and prefetching mechanism or PSP for short to hide data transfer overhead.

References

1. J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. OSDI ’04, pages 137–150, 2008.
2. M.Zaharia, A.Konwinski, A.Joseph, Y.zatz, and I.Stoica. Improving mapreduce performance in heterogeneous environments. In OSDI’08: 8th USENIX Symposium on Operating Systems Design and Implementation, October 2008.
3. R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed prefetching and caching. SIGOPS Oper. Syst. Rev., 29:79–95, December 1995.
4. Sangwon Seo, Ingook Jang, Kyungchang Woo, Inkyo Kim,et. al. Hpmr: Prefetching and pre-shuffling in shared mapreduce computation environment. In Proceedings of 11th IEEE International Conference on Cluster Computing, pages 16–20. ACM, 2009.
5. Tom White. Hadoop The Definitive Guide. O’Reilly, 2009.