Dynamic Slot Allocation Technique for MapReduce WorkLoad

M.S.Harini Lakshmi¹, S.Fathima Nisha², D.Manjula³ and A. Sheelavathi⁴
¹²³Final year IT, Saranathan College Of Engineering.
⁴Assistant Professor/IT, Saranathan College Of Engineering.

Abstract—MapReduce is a popular parallel Computing paradigm, for large-scale data processing in cluster and data centers. However, the slot utilization can be low, especially when Hadoop Fair scheduler is used, due to the pre-allocation of slots among map and reduce tasks, and the order that map tasks followed by reduce task in a typical MapReduce environment. To address this problem, we propose to allow slots to be dynamically allocated to either map or reduce tasks depending on their actual requirement. Specifically, we have proposed two types of Dynamic Hadoop Fair scheduler (DHFS), for different levels of fairness (i.e., cluster and pool level). The experimental results show performance significantly while guaranteeing the fairness.

Index Terms—Map reduces Hardtop, Fair Scheduler, Dynamic Scheduling, Slots allocation.

I. INTRODUCTION

In recent years, Map Reduce has become the parallel computing paradigm of choice for large-scale data processing in clusters and data centers. A Map Reduce job consists of a set of map and reduce tasks, where reduce tasks are performed after the map tasks. Hardtop, an open source implementation of Map Reduce, has been deployed in large clusters containing thousands of machines by companies such as Yahoo! And Facebook to support batch processing for large jobs submitted from multiple users (i.e., Map Reduce workloads). In a Hardtop cluster, the compute resources are abstracted into map (or reduce) slots, which are basic compute units and statically configured by administrator in advance. Due to 1) the slot allocation constraint assumption that map slots can only be allocated to map tasks and reduce slots can only be allocated to reduce tasks, and 2) the general execution constraints that map tasks are executed before reduce tasks, we have two observations: (I). there are significantly different performance and system utilization for a Map Reduce workload under different job execution orders and map/reduce slots configurations, and (II). Even under the optimal job submission order as well as the optimal map/reduce slots configuration, there can be many idle reduce (or map) slots while map reduce slots are not enough during the computation, which adversely affects the system utilization and performance. In our work, we address the problem of how to improve the utilization and performance of Map Reduce cluster without any prior knowledge or information (e.g., the arriving time of Map Reduce jobs, the execution time for map or reduce tasks) about Map Reduce jobs. Our solution is novel and straightforward: we break the former first assumption of slot allocation constraint to allow (1). Slots are generic and can be used by map and reduce tasks. (2). Map tasks will prefer to use map slots and likewise reduce tasks prefer to use reduce slots. In other words, when there are insufficient map slots, the map Tasks will use up all the map slots and then borrow unused reduce slots. Similarly, reduce tasks can use unallocated map slots if the number of reduce tasks is greater than the number of reduce slots. In this paper, we will focus specifically on Hardtop Fair Scheduler (HFS). This is because cluster utilization and performance for the whole Map Reduce jobs under HFS are much poorer (or more serious) than that under FIFO scheduler. But it is worth mentioning that our solution can be used for FIFO scheduler as well. HFS is a two-level hierarchy, with task slots allocation across "pools" at the top level, and slots allocation among multiple jobs within the pool at the second level. We propose two types of Dynamic Hardtop Fair Scheduler (DHFS), with the consideration of different levels of fairness (i.e., pool level and cluster-level). They are as follows:

- Pool-independent DHFS (PI-DHFS). It considers the dynamic slots allocation from the cluster-level, instead of pool-level. More precisely, it is a typed phase-based dynamic scheduler, i.e., the map tasks have priority in
the use of map slots and reduce tasks have priority to reduce slots (i.e., intra-phase dynamic slots allocation). Only when the respective phase slots requirements are met can excess slots be used by the other phase (i.e., inter phase dynamic slots allocation). 

- Pool-dependent DHFS (PD-DHFS). It is based on the assumption that each pool is selfish, i.e., each pool will always satisfy its own map and reduce tasks with its shared map and reduce slots between its map-phased pool and reduce-phased pool (i.e., intra-pool dynamic slots allocation) first, before sharing the unused slots with other overloaded pools (i.e., inter-pool dynamic slots allocation).

We have designed and implemented the two DHFSs on top of default HFS. We evaluate the performance and fairness of our proposed algorithms with synthetic workloads. Both schedulers, PI-DHFS and PD-DHFS, have shown promising results. The experimental results show that the proposed DHFS can improve the system performance significantly (by 32% - 55% for a single job and 44% - 68% for multiple jobs) while guaranteeing the fairness. Organization. The rest of the paper is organized as follows. Section II reviews the Map Reduce background and related work. Section III introduces our two types of Dynamic Hardtop Fair Scheduler, namely, PI-DHFS and PD-DHFS. Section IV reports on the performance improvement of proposed DHFS obtained from our experiments. Section V discusses fairness and slots movement for PI-DHFS and PD-DHFS. Finally, Section VI concludes the paper and gives our future work.

II. LITERATURE REVIEW

BIGDATA: The term "Big Data" has launched a veritable industry of processes, personnel and technology to support what appears to be an exploding new field. Giant companies like Amazon and Walmart as well as bodies such as the U.S. government and NASA are using Big Data to meet their business and/or strategic objectives. Big Data can also play a role for small or medium-sized companies and organizations that recognize the possibilities (which can be incredibly diverse) to capitalize upon the gains. Map Reduce is a popular programming model for processing large data sets, initially proposed by Google [16]. Now it has been a de facto standard for large scale data processing on the cloud. Hardtop is an open-source java implementation of Map Reduce. When a user submits jobs to the Hardtop cluster, Hardtop system breaks each job into multiple map tasks and reduces tasks.

Each map task processes (i.e. scans and records) a data block and produces intermediate results in the form of key-value pairs. Generally, the number of map tasks for a job is determined by input data. There is one map task per data block. The execution time for a map task is determined by the data size of an input block. The reduce tasks consists of shuffle/sort/reduce phases. In the shuffle phase, the reduce tasks fetch the intermediate outputs from each map task. In the sort/reduce phase, the reduce tasks sort intermediate data and then aggregate the intermediate values for each key to produce the final output. The number of reduce tasks for a job is not determined, which depends on the intermediate map outputs. We can empirically set the number of reduce tasks for a job to be 0.95x or 1.75x reduce tasks capacity. There are several job schedulers for Hardtop, i.e., FIFO, Hadoop Fair Scheduler. Capacity Scheduler. The job scheduling in Hardtop is performed by the job Tracker (master), which manages a set of task Trackers (slaves). Each task Tracker has a fixed number of map slots and reduces slots, configured by the administrator in advance. Typically, there is one slot per CPU core in order to make CPU and memory management on slave nodes easy. The task Trackers reports periodically to the job Tracker the number of free slots and the progress of the running tasks. The job Tracker allocates the free slots to the tasks of running jobs. In particular, the map slots can only be allocated to map tasks and reduce slots can only be allocated to reduce tasks. Hardtop Fair Scheduler is a multi-user Map Reduce job scheduler that enables organizations to share a large cluster among multiple users and ensure that all jobs get roughly an equal share of slot resources at each phase. It organizes jobs into pools and shares resources fairly across all pools based on max-min fairness. By default, each user is allocated a separate pool and, therefore, gets an equal share of the cluster no matter how many jobs they submit. Each pool consists of two parts: map-phase pool and reduce-phase pool. Within each map/reduce-phase pool, fair sharing is used to share map/reduce slots between the running jobs at each phase. Pools can also be given weights to share the cluster none proportionally in the configuration file.
III. ALGORITHMS

This proposed system implement two algorithms for dynamic job slot configuration.

1. Make span
2. Total completion time

1. Make span:
   Make span is defined as the time period since the start of the first job until the completion of the last job for a set of jobs. It considers the computation time of jobs and is often used to measure the performance and utilization efficiency of a system.
   The first class of algorithms focuses on the job ordering optimization for a Map Reduce workload under a given map/reduce slot configuration. The slot configuration can have a significant impact on performance for map reduce work load.
   It can be used to measure the satisfaction to the system from a single job’s perspective. So far, we focus only on the optimization of make span and Map Reduce workloads.
   The make span is affected primarily by the positions of large-size jobs. In contrast, the total completion time is mainly influenced by the positions of small-size jobs.
   The algorithm shortest processing time first (SPTF) is optimal for the total completion time on a single machine where there is one task per job and no precedence constraints

2. Total completion time
   In contrast, our second class of algorithms considers the scenario that we can perform optimization for map/reduce slot configuration for a Map Reduce workload.
   Total completion time is referred to as the sum of completed time periods for all jobs since the start of the first job.
   It is a generalized make span with queuing time (i.e., waiting time) included. We can use it to measure the satisfaction to the system from a single job’s perspective through dividing the total completion time by the number of jobs (i.e., average completion time).

Advantages:
- Independent job slot configuration is allowed.
- Maximum make span is used.
- Optimization process of map reduces easily...
- Low Cost.

So finally by using these two algorithms we could achieve the high performance because the time taken is first calculated and then the map reduces is done. Once the job is uploaded it is splatted as a cluster and then the mapping is done so that the server does not require check/map every time. The main difference between Map Reduce and traditional 2HFS is that Map Reduce jobs can run multiple maps and reduce tasks concurrently in each phase, whereas 2HFS allows at most one task to be processed at a time. The batch job ordering problem has been extensively studied in the high performance computing. The previous works all focused on the single-stage parallelism, where each job only has a single stage. Minimizing the make span for 2HFS is strongly NP-hard when at least one stage contains multiple process. A set of general low-level optimizations including improving I/O speed, utilizing indexes, using fingerprinting for faster key comparisons, and block size tuning. An I/O-efficient Map Reduce system called job optimization.

Architecture Diagram:
guaranteeing the fairness. The core technique is dynamically allocating map (or reduce) slots to map and reduce tasks. Two types of DHFS are presented, namely, I-DHFS and PD-DHFS, based on fairness for cluster and pools, respectively. The experimental results show that our proposed DHFS can improve the performance and utilization of the Hardtop cluster significantly. As for future work, we are interested in extending our dynamic slot allocation algorithms to environments. Cluster/cloud has become heterogeneous with different architectures. We plan to extend our previous study to handle the slot configuration on CPUs and GPUs.

REFERENCES


