SLOT CONFIGURATIONS BASED ON SELF-ADJUSTING FOR HETEROGENEOUS HADOOP CLUSTERS

Tejaswini M¹

¹M.Tech, Department of Computer science and Engineering, AMC Engineering College, Bangalore

ABSTRACT:

The open source Hadoop and Mapreduce framework are the defacto platform for scalable analysis on large data sets. How to reduce the completion length of a set of MapReduce jobs is one of the primary concerns in Hadoop. The current open source Hadoop allows only static slot configuration, like fixed numbers of map slots and reduce slots throughout the cluster lifetime. Such static configuration may lead to long completion length as well as low system resource utilizations. Propose new schemes which use slot ratio between map and reduce tasks as a tunable knob for minimizing the completion length (i.e., makespan) of a given set. By leveraging the workload information of recently completed jobs, schemes dynamically allocates resources (or slots) to map and reduce tasks.

Keywords: Mapreduce job, Hadoop scheduling, reduced makespan, slot configuration

[1] INTRODUCTION

MapReduce[1] for processing big data in parallel. Its open source implementation Apache Hadoop [2] has popular platform for data processing and information analysis. With the rise of cloud computing, It is now convenient for a regular user to launch a MapReduce cluster on the cloud, e.g., AWS MapReduce, for data-intensive applications. How to improve the performance of a MapReduce cluster becomes a focus of research and development [3–11]. As a complex system, Hadoop is configured with a large set of system parameters. While it provides the flexibility to customize the cluster for different applications, it is challenging for users to understand and set the optimal values for those parameters. In this paper, aim to develop algorithms for adjusting a basic system parameter with the goal to improve the performance (i.e., reduce the makespan) of a batch of MapReduce jobs.
A Hadoop cluster has single master node and multiple slave nodes. The master node runs the JobTracker routine which is responsible for scheduling jobs and coordinating the execution of tasks of each job. Each slave node runs the TaskTracker for hosting the execution of MapReduce jobs. The concept of “slot” is used to indicate the capacity of accommodating tasks on each node. In a Hadoop system, a slot is assigned as a map slot or a reduce slot serving map tasks or reduce tasks, respectively. At any given time, only one task can be running per slot. The number of available slots per node indeed provides the maximum degree of parallelization in Hadoop. Here shown that the slot configuration has a significant impact on system performance. The Hadoop framework use fixed numbers of map slots and reduce slots at each node as the default setting throughout the lifetime of a cluster. The values in this fixed configuration are usually heuristic numbers without considering job characteristics. Therefore, this static setting is not well optimized and may hinder the performance improvement of the entire cluster. In this work, propose and implement a new mechanism to dynamically allocate slots for map and reduce tasks. The primary goal of the new mechanism is to improve the completion time (i.e., the makespan) of a batch of MapReduce jobs while retain the simplicity in implementation and management of the slot-based Hadoop design.

The key idea of this new mechanism, named TuMM, is to automate the slot assignment ratio between map and reduce tasks in a cluster as a tunable knob for reducing the makespan of MapReduce jobs. The Workload Monitor (WM) and the Slot Assigner (SA) are the two major components introduced by TuMM. The WM that resides in the JobTracker periodically collects the execution time information of recently finished tasks and estimates the present map and reduce workloads in the cluster. The SA module takes the estimation to decide and adjust the slot ratio between map and reduce tasks for each slave node. With TuMM, the map and reduce phases of jobs could be better pipelined under priority based schedulers, and thus the makespan is reduced. Further the dynamic slot assignments in heterogeneous environments, and propose a new version of TuMM, named H TuMM, which sets the slot configurations for each individual node to reduce the makespan of a batch of jobs.

[2] SYSTEM MODEL AND DYNAMIC SLOT CONFIGURATION UNDER HETEROGENEOUS ENVIRONMENTS

Heterogeneous environments are fairly common in today’s cluster systems. For example, system managers of a private data center could always scale up their data center by adding new physical machines. Therefore, physical machines with different models and different resource capacities can exist simultaneously in a cluster
When deploying a Hadoop cluster [Figure 1] in such a heterogeneous environment, tasks from the same job may have different execution times when running on different nodes. In this case, a task’s execution time highly depends on a particular node where that task is running. A job’s map tasks may run faster on a node which has faster cpu per slot while its reduce tasks may experience shorter execution times on the other nodes that have more memory per slot. Estimating the remaining workloads and deciding the slot configuration in heterogeneous Hadoop cluster becomes more complex.

For example, consider a Hadoop job with 7 map tasks and a Hadoop cluster with two heterogeneous nodes such that node 1 is faster than node 2. Consider a cluster configured with 4 map slots in total, and one map task of that job takes 1 second and 2 seconds to finish on node 1 and node 2, respectively. We note that in this heterogeneous Hadoop cluster, various slot configurations will yield different performance (e.g., the execution time) of this job.
Figure 2. Illustrating a Hadoop job with 7 map tasks running in a heterogeneous Hadoop cluster with 2 nodes and 4 map slots in total. The map phase of that job run faster when have (c) 3 map slots on Node 1 and 1 map slot on Node 2, than when have (a) 2 map slot on Node 1 and 2 map slots on Node 2, and (b) 1 map slot on Node 1 and 3 map slots on Node 2.

As illustrated in [Figure 2] case 1, the total execution time of the map phase takes 4 seconds on node 2. However, the map phase execution time can be improved to 3 seconds if we change the slot configuration on these two nodes, i.e., 3 map slots on Node 1 and 1 map slot on Node 2. This situation indicates that it is harder to predict the time needed to finish the map phase or reduce phase in the heterogeneous environment, and evenly distribute the map (or reduce) slot assignments across the cluster will no longer work well. Which utilizes the overall workload information to set the slot assignments over the entire cluster does not work well any more when the nodes in the cluster become heterogeneous. New version of TuMM, named H TuMM, which dynamically sets the slot configurations for each node in a heterogeneous Hadoop cluster in order to reduce the makespan of Hadoop jobs.

2.1 Algorithm Design: H TuMM H TuMM shares the similar idea of TuMM, i.e., dynamically assign slots to map and reduce tasks to align the process of map and reduce phase based on the collected workload information. The key difference of H TuMM is to set the slot configurations for each node individually in a heterogeneous cluster, i.e., each of those nodes will have different slot assignment ratio between map and reduce tasks. To accomplish it, H TuMM collects the workload information on the entire cluster and on each individual node as well: when a map/reduce task is finished on node i, the workload collector updates

(1) The average execution time of map/reduce tasks, i.e., $t_m/t_i$;
(2) The average execution of map/reduce tasks that ran on node i, i.e., $t_m^i/t_i^i$. Based on the collected workload information, H TuMM performs slot assignment for each node. Once a slot in node i becomes available, H TuMM first updates the slot assignments to map tasks ($s_{im}$) and reduce tasks ($s_{ir}$) on node i. Such that the ratio of slot assignments (i.e., $s_{im}/s_{ir}$) is equal to the ratio of remaining map and reduce workloads (i.e., $t_m^i*n_m^i/t_r^i*n_r^i$). Therefore, map and reduce phases running on that node are aligned. If there is one remaining slot, in this case, the free slot will be assigned to a map (resp. reduce) task if map (resp. reduce) tasks run relatively faster on this node compared to the average execution time across the entire cluster in order to improve the efficiency, see line 3-7 in Algorithm 1. When the slot assignment on the specific node is determined, the JobTracker can assign tasks based on the new slot configuration and the number of currently running tasks on that node (i.e., $rt_m^i$ and $rt_r^i$), see line 8-11 in Algorithm.

Algorithm for Node i 0:
0: **Input:** Average task execution time on node $i$ and across the cluster, and the remaining task number of current running jobs;
0: **When Node** $i$ has free slots and ask for new task assignment through the heartbeat message;
1: $s_{m}^{i} \leftarrow \left[ S^{i} \times \frac{t_{m}^{i} \times n_{m}^{i}}{t_{m}^{i} \times n_{m}^{i} + t_{r}^{i} \times n_{r}^{i}} \right]$;
2: $s_{r}^{i} \leftarrow \left[ S^{i} \times \frac{t_{r}^{i} \times n_{r}^{i}}{t_{m}^{i} \times n_{m}^{i} + t_{r}^{i} \times n_{r}^{i}} \right]$;
3: **if** $s_{m}^{i} + s_{r}^{i} \leq S^{i}$ **then**
4: **if** $\frac{r_{m}}{r_{r}} > \frac{t_{m}}{t_{r}}$ **then**
5: $s_{r} \leftarrow S^{i} - s_{m}^{i}$;
6: **else**
7: $s_{m}^{i} \leftarrow S^{i} - s_{r}^{i}$;
8: **if** $(s_{m}^{i} - r_{m}^{i}) > (s_{r}^{i} - r_{r}^{i})$ **then**
9: assign a map task to node $i$;
10: **else**
11: assign a reduce task to node $i$;

### [3] RELATED WORKS

An important direction for improving the performance of a Hadoop system is job scheduling. The default FIFO scheduler does not work well in a shared cluster with multiple users and a variety of jobs. Fair [12] and Capacity [13] schedulers were proposed to ensure that each job can get a proper share of the available resources; and Quincy [5] addressed the scheduling problem with locality and fairness constraints. Zaharia et al. [3] proposed a delay scheduling to further improve the performance of the Fair scheduler by increasing data locality. Verma et al. [4] introduced a heuristic to minimize the makespan of a set of independent MapReduce jobs by applying the classic Johnson’s algorithm.

### [4] CONCLUSION

In this paper presented a novel slot management scheme, named TuMM, to enable dynamic slot configuration in Hadoop. The main objective of TuMM is to improve resource utilization and reduce the makespan of multiple jobs. To meet this goal, the presented scheme introduces two main components: Workload Monitor periodically tracks the execution information of recently completed tasks and estimates the present workloads of map and reduce tasks and Slot Assigner dynamically allocates the slots to map and reduce tasks by leveraging the estimated workload information.
REFERENCES


