Improving the Performance of Hadoop Map reduce using Dynamic Slot Configurations

P. Anusha, Dr. A. Venkataramana
Department of CSE,
GMR Institute of Technology, Rajam, AP-India

Abstract

Hadoop have become the adequate platform for scalable analysis on large data sets with the map reduces technology. Primarily in Hadoop is how to minimize the completion length of a set of MapReduce jobs. Hadoop only allows static slot configuration, i.e., fixed numbers of map slots and reduce slots throughout the lifetime of a cluster. This static configuration may lead to low system resource utilizations as well as long computation length. So has to overcome this problem, this project proposes effective schemes which use Dynamic Slot Configurations for Homogeneous and Heterogeneous Clusters using Hadoop Map reduce. By leveraging the workload information of recently completed jobs, the proposed scheme dynamically allocates resources to map and reduce tasks. This scheme is more effective under both simple workloads and more complex mixed workloads.

Keywords: MapReduce jobs, Hadoop scheduling, reduced makespan, slot configurations.

I. INTRODUCTION

An increasing number of popular applications become data-intensive in nature. In the past decade, the World Wide Web has been adopted as an ideal platform for developing data-intensive applications, since the communication paradigm of the Internet is sufficiently open and powerful. Representative data-intensive Web applications include search engines, online auctions, webmail, and online retail sales, to name just a few. Data-intensive applications like data mining and web indexing need to access ever-expanding data sets ranging from a few gigabytes to several terabytes or even petabytes. Google, for example, leverages the MapReduce model to process approximately twenty petabytes of data per day in a parallel fashion [3].

The MapReduce programming framework can simplify the complexity of running parallel data processing functions across multiple computing nodes in a cluster, because scalable MapReduce helps programmers to distribute programs and have them executed in parallel. MapReduce automatically handles the gathering of results across the multiple machines and return a single result or set. More importantly, the MapReduce platform can offer fault tolerance that is entirely transparent to programmers. Right now, MapReduce is a practical and attractive programming model for parallel data processing in high-performance cluster computing environments. Hadoop a popular open-source implementation of the Google’s MapReduce model [is primarily developed by Yahoo [4]. Hadoop is used by Yahoo’s servers, where hundreds of terabytes of data are generated on at least 10,000 processor cores [6].

Facebook makes use of Hadoop to process more than 15 terabytes of new data per day. In addition to Yahoo and Facebook, a wide variety of websites like Amazon and Last.fm are employing Hadoop to manage massive amount of data on a daily basis [5]. Apart from Web data intensive applications, scientific data-intensive applications (e.g., seismic simulations and natural language processing) take maximum benefits from the Hadoop system [1][5].

A Hadoop system basically consists of two major parts. The first part is the Hadoop MapReduce engine MapReduce [3]. The second component is HDFS Hadoop Distributed File System [2], which is inspired by Google’s GFS (i.e., Google File System). Currently, HDFS divides files into blocks that are replicated among several different computing nodes with no attention to whether the blocks are divided evenly. When a job is initiated, the processor of each node works with the data on their local hard disks. In the initial phase of this dissertation research, we investigate how Hadoop works with its parallel file system, like Lustre. Lustre divides a large file into small pieces, which are evenly distributed across multiple nodes. When the large file is accessed, high aggregated I/O bandwidth can be achieved by accessing the multiple nodes in parallel. The performance of cluster can be improved by Hadoop, because multiple nodes work concurrently to provide high throughput. Although Hadoop is becoming popular as a high-performance computing platform for data-intensive applications, increasing evidence has shown that performance of data-intensive applications can be severely limited by a combination of a persistent lack of high disk and network-I/O bandwidth and a significant increase in I/O activities.

In other words, performance bottlenecks for data-intensive applications running in cluster environments are caused by disk- and network-I/O rather than CPU or memory performance. There are multiple reasons for this I/O performance problem. First, the performance gap between processors and I/O subsystems in clusters is rapidly widening. For example, processor performance has seen an annual increase of approximately 60% for the last two decades, while the overall performance improvement...
of disks has been hovering around an annual growth rate of 7% during the same period of time. Second, the heterogeneity of various resources in clusters makes the I/O bottleneck problem even more pronounced.

II. REALATED WORK

Zaharia et al. [3] proposed a lengthen scheduling to further fortify the performance of the reasonable scheduler by increasing information locality. Verma et al. [4] introduced a heuristic to decrease the lifetime of a suite of independent MapReduce jobs by making use of the classic Johnson’s algorithm. Another category of schedulers additional consider user level objectives at the same time improving the performance. ARIA, a deadline conscious scheduler, used to be recently proposed in [6], which invariably schedules a job with the earliest time limit and makes use of the Lagrange’s procedure to seek out the minimum quantity of slots for each job with the intention to meet the predefined time limit.

In a similar way, Polo et al. [7] estimated the assignment execution instances based on the common execution times of the accomplished duties instead of the job profiles. Undertaking execution instances were then used to calculate the number of slots that a job needed to fulfill its time limit. Special closing date and locality aware scheduling algorithms are evaluated with empirical analysis for Hadoop process in [9]. Even though these cut-off date mindful schedulers aid user-level targets, their systems are still based on static slot configurations, i.e., having a constant quantity of map slots and decrease slots per node for the duration of the lifetime of a cluster.

First-class-grained useful resource conscious management is a further fundamental path for bettering performance in Hadoop. RAS [10] leverages present profiling information to dynamically assess the number of job slots and their placement within the cluster. The purpose of this approach is to maximize the useful resource utilization of the cluster and to meet job completion time points in time. Extra just lately, [11] introduces a nearby resource manager at each TaskTracker to notice venture resource utilization and predict undertaking finish time, and a worldwide resource manager on the JobTracker to coordinate the useful resource assignments to every mission; and [9] addresses the cluster useful resource utilization main issue via constructing a dynamic cut up model of resource utilization.

III. SYSTEM APPROACH

The system architecture is depicted in fig.2 whereby Hadoop cluster, data are distributed to individual nodes and stored in their disks. To execute a map task on a node, make its input data available on that node ‘and it starts to execute the file by assigning the task to the task tracker in the master node The master node invokes the MapReduce scheduler to assign tasks to slave node. A slave node who is assigned a map tasks reads the contents of the corresponding input data block, parses the input key pairs out of block, and parses each pair to user defined map functions. The reduce task then reads the intermediate data and invokes the reduce function to produce output. The key idea of this new mechanism, named TuMM, is to manually the slot assignment ratio between maps and to reduce tasks in a cluster as a tunable especially at the end for reducing the makespan of Map Reduce jobs. There are two major components introduced by TuMM is

A. Dynamic slot configuration under Homogeneous environments

The default Hadoop cluster uses static slot configuration and does not perform well for varying workloads. The inappropriate setting of $s_m$ and $s_r$ may lead to extra overhead because of the following two cases:

- If job $j_i$’s map phase is completed advanced than job $j_i$’s reduce phase, then the reduce slots will be idle for the interval period of $(st(i)+w_m(i))-ft(i-1)$, see Fig. 1(a).

- If job $j_i$’s map phase is completed earlier than the job $j_{i+1}$’s reduce phase, then $j_{i+1}$’s reduce tasks have to wait for a period of $(f(i+1)-st(i+1)+w_m(i))$ until reduce slots are released by $j_{i+1}$, see Fig. 1(b). Our intuition is to eliminate the two undesired cases mentioned above by aligning the completion of $j_i$’s map phase and $j_{i+1}$’s reduce phase, see Fig. 1(c).

B. Workload monitor(WM)

The WM that of power in the Job Tracker periodically collects the execution time information of recently for the completion of tasks and implement the present map and reduce workloads in the cluster.

C. Slot Assigner

To estimates the SA modules decides and adjust the slot ratio between map and reduce tasks for each and every slave node. The slave node has to perform the task that assign to it and it has to inform to the job Tracker if it done the execution of tasks and it has been reallocated.

When deploying a Hadoop cluster as depicted in above 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.

As illustrated in Fig. 3 case 1, the total execution time of the map phase takes slot on node 2. However, the map phase execution time can be improved to 3 seconds if we change the slot configures on these two nodes, i.e., 3 map slot on node 1 and 1 map slots 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.

D. Algorithm Design

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:

- The average execution time of map/reduce tasks.
- The average execution of map/reduce tasks that ran on node i, i.e., tim/tir. 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 (sim) and reduce tasks (sir) on node i. Such that the ratio of slot assignments is equal to the ratio of remaining map and reduce workloads (i.e., 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, refer in Algorithm. 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, refers in Algorithm.

**Input:** Number of slave nodes in cluster: \( k \)

**Total numbers of running map/reduce tasks:**

**When** receive heartbeat message from node \( x \) with the number of running map/reduce tasks on node \( x \):**

**Initialize** assignment of slots for node \( x \):

- if
  - \( +1 \)
- else
  - \( +1 \)

if assign a map task to node \( x \);
else assign a reduce task to node \( x \).

**IV. WORKING RESULTS**

From the observed results, it is found that dynamic slot Configuration enhances the performance of MapReduce tasks. Observing all the iterations, when the map tasks are increasing the completion time will be decreased and memory usage increased. See the iteration 7 that having the Slot Ratio 5:1. Now we have 5 map slots and 1 reduce slot that means each map task taken as one slot. So the performance of MapReduce tasks is increased.

![Fig 4. Outcome of Dynamic slot configuration](image)

**V. CONCLUSION**

All observations are calculated with respect to the original Hadoop. It is observed that, even under the optimized map/reduce slot configuration for the original Hadoop, our Dynamic Slot Configuration system can still further improve the performance of MapReduce jobs significantly. Slot where dynamically map and reduce tasks performances are used to reach high rate in performance and assignment of tasks to slots are done dynamically. To achieve efficiency for tasks performed in slots which is crucial. Speculative execution performance balance used to raise performance of group of jobs and prescheduling used to achieve data locality at high rate overcome loop holes and do the required operation in positive direction.

**REFERENCES**


