Performance Improvement of MapReduce Framework by Identifying Slow TaskTrackers in Heterogeneous Hadoop Cluster

Nenavath Srinivas Naik, Atul Negi and V.N. Sastry

Abstract MapReduce is presently recognized as a significant parallel and distributed programming model with wide acclaim for large scale computing. MapReduce framework divides a job into *map*, *reduce* tasks and schedules these tasks in a distributed manner across the cluster. Scheduling of tasks and identification of "slow TaskTrackers" in heterogeneous Hadoop clusters is the focus of recent research. MapReduce performance is currently limited by its default scheduler, which does not adapt well in heterogeneous environments. In this paper, we propose a scheduling method to identify "slow TaskTrackers" in a heterogeneous Hadoop cluster and implement the proposed method by integrating it with the Hadoop default scheduling algorithm. The performance of this method is compared with the Hadoop default scheduler. We observe that the proposed approach shows modest but consistent improvement against the default Hadoop scheduler in heterogeneous environments. We see that it improves by minimizing the overall job execution time.

Keywords Hadoop · MapReduce · Job scheduling · TaskTracker · Heterogeneous environments

N.S. Naik (✉) · A. Negi
School of Computer and Information Sciences, University of Hyderabad,
Hyderabad 500046, India
e-mail: srinuphdcs@gmail.com

A. Negi
e-mail: atulcs@uohyd.ernet.in

V.N. Sastry
Institute for Development and Research in Banking Technology,
Hyderabad 500057, India
e-mail: vnsastry@idrbt.ac.in

© Springer India 2016

A. Nagar et al. (eds.), *Proceedings of 3rd International Conference on Advanced Computing, Networking and Informatics*, Smart Innovation, Systems and Technologies 44, DOI 10.1007/978-81-322-2529-4_49

1 Introduction

Efficiently storing, querying, analyzing, interpreting, and utilizing these huge data sets presents one of the impressive challenges to the computing industry and the research community [1]. A large number of organizations across the world use Apache Hadoop, created by Doug Cutting, which is an open source implementation of the MapReduce framework and processes massive amounts of data in-parallel on large clusters of commodity systems. Take Yahoo, for example. Uses a Hadoop cluster of 4,000 nodes, having 30,000 CPU cores, and 17 petabytes of disk space [2]. The structure of MapReduce is based on the master-slave architecture [3]. A single master node monitors the status of all slave nodes in the cluster and allocates jobs to them. The benefits of MapReduce framework are the capability of fault tolerance and appropriate distribution of tasks to multiple processing nodes in the cluster [4].

The basic assumption of Hadoop framework is that the nodes of the cluster are homogeneous [5]. Several issues which will directly affect the performance of MapReduce framework are node heterogeneity, stragglers, data locality and "slow TaskTrackers" [6]. These issues have been undervalued by researchers in most of the proposed MapReduce scheduling algorithms, which leads to poor performance of Hadoop [7]. Minimizing the execution time of a job by appropriately assigning tasks to the available nodes is a common goal of the MapReduce schedulers and it is likewise a significant research topic because it betters the performance of MapReduce framework [8].

In this research work, we address the problem of identifying "slow TaskTrackers" in the heterogeneous Hadoop cluster by integrating it with the Hadoop default scheduler. The proposed work helps the JobTracker not to schedule any task on these identified "slow TaskTrackers" instead schedule on the remaining TaskTrackers, which minimizes the job execution time and certainly improves the overall performance of the MapReduce framework in heterogeneous environments. Throughout this paper by "slow TaskTracker" we are referring to a TaskTracker which has some tasks under it that are running slower relative to other tasks.

The rest of the paper is structured as follows. A background of the MapReduce framework and the Hadoop's default scheduler as related work is given in Sect. 2. Procedure for identifying "slow TaskTrackers" in the heterogeneous Hadoop cluster is given in Sect. 3 and Sect. 4 conducts a performance evaluation of the proposed work. Finally, we conclude the paper and give some outlines of our future research work in Sect. 5.

2 Related Work

This section provides a brief view of the MapReduce framework and explains about the Hadoop default scheduling algorithm with its limitations.

2.1 Basic Concepts in MapReduce

In Hadoop cluster, HDFS (Hadoop Distributed file system) contains one single NameNode called master node and a number of DataNodes called worker nodes [9]. NameNode maintains the meta-data information about the locations of data chunks and DataNode stores the chunks of data in the cluster. For running a job in the cluster, MapReduce component is used, which contains one JobTracker and a series of TaskTrackers [10]. JobTracker manages the jobs and assigns tasks to the TaskTrackers and TaskTracker processes the tasks on the corresponding node in the cluster [11].

Scheduling of the MapReduce system has following stages while scheduling a job in the cluster [12].

1. The Hadoop framework first breaks the input data file into M pieces of identical data sizes and then distributed in the cluster.
2. The master node will pick up the idle worker nodes and allocates them M *map* tasks. After intermediate output is produced by *map* tasks, the master node will allocates R *reduce* tasks to the worker nodes which are idle.
3. The intermediate (key, value) pairs from the *map* function are buffered to local disks at regular intervals.
4. The above buffered pairs are split into R regions by (*map*) worker using a partition function (default is *hash* (intermediate key) mod R), so that same intermediate (key, value) pairs go to one partition.
5. Reducers will read the data from the *map* workers using remote procedure calls, then it sorts and groups the data by intermediate key so that all values of the same key are collected together.
6. After complete execution of the *map* and *reduce* tasks, the outcomes will be fed back to the user by the master node.

2.2 Hadoop Default Scheduling Algorithm

The progress score (PS) of a task t is denoted by PS_t, which is calculated using (1) for *map* tasks and (2) for *reduce* tasks [13].

$$PS_t = M/N \tag{1}$$

$$PS_t = (1/3)(K + M/N) \tag{2}$$

where, M is the number of (key, value) pairs that have been processed successfully, N is the overall number of (key, value) pairs and K is the stage (shuffle, sort and merge) value in a *reduce* phase.

The average progress score of a job PS_{avg} is calculated using (3), $PS[i]$ is the progress score of a task t_i and n is the number of executable tasks in a job.

$$PS_{avg} = \sum_{i=1}^{n} PS[i]/n \tag{3}$$

Limitations of Hadoop Default Scheduler [13]

1. The *map* and *reduce* task weights in different stages are $M_1 = 1$, $M_2 = 0$ and ($R_1 = R_2 = R_3 = 1/3$) but these weights will change when tasks run in a heterogeneous environment.
2. Default scheduler cannot identify the "slow TaskTrackers" in a heterogeneous Hadoop cluster.
3. Default scheduler unobserved the accurate straggler tasks which need to be re-executed in the cluster.

3 Proposed Method for Identifying Slow TaskTrackers in Heterogeneous Hadoop Cluster

The performance of distributed and parallel systems like MapReduce is closely related to its Task scheduler. If a task is scheduled on a "slow TaskTracker" then the overall execution time of a job will be increased. Finding "slow TaskTrackers" in heterogeneous Hadoop cluster is an interesting research problem because the efficient way of finding it can significantly *reduce* the overall job execution time and thus improves the performance of the MapReduce framework in heterogeneous environments.

The Progress score of a TaskTracker in the cluster is calculated using (4)

$$PSTT_i = \sum_{j=1}^{t} PS_j/t \tag{4}$$

Here, the progress score of ith TaskTracker is $PSTT_i$, PS_j is the progress score of a task calculated based on how much a task's (key, value) pairs have been finished per second, which is calculated as in Hadoop default scheduler and t is the number of tasks on the ith TaskTracker in the cluster.

The average progress score of all TaskTrackers in the Hadoop cluster for a given job is calculated using (5)

$$APSTT = \sum_{i=1}^{T} PSTT_i/T \tag{5}$$

Here, *APSTT* is the average progress score of all TaskTrackers in the cluster and *T* is the number of TaskTrackers present in the Hadoop cluster.

We can find the "slow TaskTrackers" present in the cluster using (6)

$$PSTT_i > APSTT(TTTh + 1) \tag{6}$$

For the *i*th TaskTracker, if it satisfies the above equation, then we can say that particular TaskTracker is a "slow TaskTracker" otherwise it is the fast TaskTracker in the heterogeneous Hadoop cluster.

TaskTracker Threshold (*TTTh*) is in the range [0,1] is used to categorize the TaskTrackers in the Hadoop cluster into slow and fast. According to (6), if *TTTh* is too small then it will categorize some fast TaskTrackers to be "slow TaskTrackers" and if *TTTh* is too large then it will categorize some "slow TaskTrackers" to be fast TaskTrackers. Thus, we have chosen 0.5 as an appropriate value for *TTTh* in our experiments.

Input: The set of TaskTrackers present in the heterogeneous Hadoop cluster.
Output: The set of "slow TaskTrackers".

Algorithm 1 Identifying slowTaskTrackers

```
1: set $slowTaskTrackers$
2: for each $TaskTracker\ i$ in the $cluster$ do
3:     for each running $task\ j$ of the $job$ do
4:         if $task\ j$ is a $Map$ task then
5:             $ProgressScore_j \leftarrow M/N$
6:         else
7:             $ProgressScore_j \leftarrow 1/3 * (K + M/N)$
8:         end if
9:     end for
10:    $PSTT_i = \sum_{j=1}^{t} PS_j/t$
11: end for
12: $APSTT = \sum_{i=1}^{T} PSTT_i/T$
13: for each running $task\ i$ of the $job$ do
14:     if $PSTT_i > APSTT(TTTh + 1)$ then
15:         $slowTaskTrackers.add(i^{th}\ TaskTracker)$
16:     end if
17: end for
18: return $slowTaskTrackers$
```

4 Evaluation

In this section, we now briefly discuss the experimental environment, workload description and then explains the performance analysis of our proposed method on a heterogeneous Hadoop cluster.

4.1 Experimental Environment

We followed numerous stages to establish the experimental setup required to conduct our experiments and considered heterogeneous nodes in a Hadoop cluster as presented in Table 1, it has different Hadoop cluster hardware environment and configurations. We used Hadoop cluster of five heterogeneous nodes to evaluate our proposed method for finding "slow TaskTrackers". One of the nodes was chosen as a master node which runs the Hadoop distributed file system (NameNode) and MapReduce runtime (JobTracker). The remaining four nodes were worker nodes (DataNodes and TaskTrackers). The nodes were interconnected by Ethernet switch. All systems in the cluster use Ubuntu 14.04 operating system, JDK version 8, and Hadoop 1.2.1 version for performance evaluation.

In our experiments, we evaluate the proposed scheduling method using Hi-Bench benchmark suite [14] because it is a new, realistic and comprehensive benchmark suite for Hadoop.

4.2 Workload Description

We evaluate our proposed method using three different job types: Sort, WordCount, and TeraSort, that simulate micro benchmarks of Hi-Bench benchmark suite. These

Table 1 Hadoop evaluation environment

Node	Hardware configuration	Hadoop configuration
Master node	Intel Xeon(R) CPU E3110 @ 3.00 GHz, 4 GB RAM, 500 GB Disk space	
Slave node 1	Intel core i3-3220 CPU @ 3.30 GHz, 2 GB RAM, 500 GB Disk space	3 *map* and 1 *reduce* slots per node
Slave node 2	Intel core 2 duo CPU E7500 @ 2.93 GHz, 2 GB RAM, 320 GB Disk space	2 *map* and 1 *reduce* slots per node
Slave node 3	Intel Pentium CPU G640 @ 2.80 GHz, 2 GB RAM, 500 GB Disk space	1 *map* and 1 *reduce* slots per node
Slave node 4	Intel Core 2 Duo Processor P8400 @ 2.26 GHz, 3 GB RAM, 250 GB Disk space	2 *map* and 1 *reduce* slots per node

micro benchmarks show the key characteristics of MapReduce clearly and widely used by the Hadoop research community to evaluate the scheduling algorithms in their experiments. We briefly describe the micro-benchmarks as below [14]:

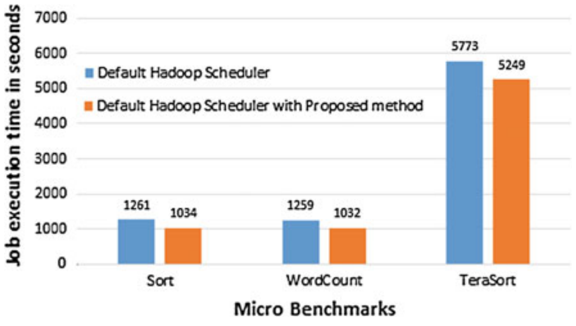
1. The WordCount workload counts the word frequencies from textual data. It is mostly CPU bound (particularly during the *map* phase), causing high CPU usage, light disk or network I/O.
2. The Sort workload depends on the Hadoop framework to sort the final results. It is mostly I/O bound, having moderate CPU usage and heavy disk I/O.
3. The TeraSort workload is very high CPU utilization and moderate disk I/O during the *map* and shuffle phases, and moderate CPU usage and heavy disk I/O during the *reduce* phase.

4.3 Performance Analysis of Our Proposed Method

In order to evaluate the performance, we have integrated our proposed method with the Hadoop default scheduling algorithm to identify the "slow TaskTrackers" in the heterogeneous Hadoop cluster. We compared our proposed method with the Hadoop default scheduler because it is a simple, fast algorithm, extensively used in numerous recent Hadoop clusters and it has no procedure to find the "slow TaskTrackers" and assumes nodes in the cluster as homogeneous. We presented our performance improvement by comparing the proposed method with the Hadoop default scheduler and performed Sort, WordCount, TeraSort benchmarks under heterogeneous environments by considering the Job execution time as a metric for the evaluation.

In our experiments, we presented how "slow TaskTrackers" effect the execution time of a job and performed three micro benchmarks over the MapReduce job execution time metric for performance evaluation in the heterogeneous Hadoop cluster. Figure 1 shows the performance comparison of the Default Hadoop scheduler and Default Hadoop scheduler with the proposed method. In all of these different workloads (Sort, WordCount and TeraSort), our proposed method achieves the best in terms of minimum job execution time compared to the Hadoop default scheduling algorithm in the heterogeneous environments.

Fig. 1 Comparison of job execution time for different workloads



5 Conclusion and Future Work

In this paper, we proposed a scheduling method and integrated it with the Hadoop default scheduler, which aims to find the "slow TaskTrackers" in the heterogeneous Hadoop cluster and it predicts the JobTracker in such a way that it will not schedule any new tasks on the identified "slow TaskTrackers" in the cluster. In this proposed method, when a JobTracker schedules a task on the TaskTracker, first it identifies the "slow TaskTrackers" present in the Hadoop cluster, then it will not schedule the tasks on those particular "slow TaskTrackers" instead schedules on the remaining TaskTrackers in the Hadoop cluster. Our proposed method shows the best performance in terms of job execution time compared to the Hadoop default scheduler when executing the Sort, Word Count, and TeraSort benchmarks and thus it improves the performance of the MapReduce framework in the heterogeneous environments by minimizing the overall job execution time.

As part of the future research work, we would like to further identify the "slow TaskTrackers" in each of the *map* and *reduce* phases of the MapReduce framework in heterogeneous environments.

Acknowledgments Nenavath Srinivas Naik express his gratitude to Prof. P.A. Sastry (Principal), Prof. J. Prasanna Kumar (Head of the CSE Department) and Dr. B. Sandhya, MVSR Engineering College, Hyderabad, India for hosting the experimental test bed.

References

1. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**, 107–113 (2008)
2. Dean, J., Ghemawat, S.: MapReduce: a flexible data processing tool. Commun. ACM **53**(1), 72–77 (2010)
3. Rasooli, A., Down, D.G.: An adaptive scheduling algorithm for dynamic heterogeneous hadoop systems. In: Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research, pp. 30–44. Canada (2011)
4. Zaharia, M., Borthakur, D., Sarma, J.S., Elmeleegy, K., Shenker, S., Stoica, I.: Job Scheduling for Multi-User MapReduce Clusters. Technical Report, University of California, Berkeley (2009)
5. Dawei, J., Beng, C.O., Lei, S., Sai, W.: The Performance of MapReduce: An In-depth Study. VLDB (2010)
6. Zaharia, M., Konwinski, A., Joseph, A.D., Katz, R., Stoica, I.: Improving mapreduce performance in heterogeneous environments. In: 8th Usenix Symposium on Operating Systems Design and Implementation, pp. 29–42. ACM Press, New York (2008)
7. Tan, J., Meng, X., Zhang, L.: Delay Tails in Mapreduce Scheduling. Technical Report, IBM T. J. Watson Research Center, New York (2011)
8. Ekanayake, J., Pallickara, S., Fox, G.: MapReduce for data intensive scientific analyses. In: Proceedings of the 2008 IEEE Fourth International Conference on eScience, pp. 277–284 (2008)
9. Rasooli, A., Down, D.G.: A hybrid scheduling approach for scalable heterogeneous Hadoop systems. In: Proceeding of the 5th Workshop on Many-Task Computing on Grids and Supercomputers, pp. 1284–1291 (2012)

10. Nanduri, R., Maheshwari, N., Reddyraja, A., Varma, V.: Job aware scheduling algorithm for mapreduce framework. In: Proceedings of the 3rd International Conference on Cloud Computing Technology and Science, pp. 724–729, Washington, USA (2011)
11. Zhenhua, G., Geo, R.F., Zhou, M., Yang, R.: Improving resource utilization in MapReduce. In: IEEE International Conference on Cluster Computing, pp. 402–410 (2012)
12. Rasooli, A., Down, D.G.: COSHH: a classification and optimization based scheduler for heterogeneous Hadoop systems. J. Future Gener. Comput. Syst. 1–15 (2014)
13. Naik, N.S., Negi, A., Sastry, V.N.: A review of adaptive approaches to MapReduce scheduling in heterogeneous environments. In: IEEE International Conference on Advances in Computing, Communications and Informatics, pp. 677–683, Delhi, India (2014)
14. Shengsheng, H., Jie, H., Jinquan, D., Tao, X., Huang, B.: The HiBench benchmark suite: characterization of the MapReduce-based data analysis. In: IEEE 26th International Conference on Data Engineering Workshops, pp. 41–51 (2010)