International Journal of Research
Available at https://pen2print.org/index.php/ijr/
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 23
December 2018

Phase Level Scheduler for Map Reduce Using Grained Resource

Ch.Sireesha & K.Sreehari
1 PG Scholar, Dept of CSE, Prakasam Engineering College, Prakasam(Dt), AP, India.
2 Asst Professor , Dept of CSE, Prakasam Engineering College, Prakasam(Dt), AP, India.

Abstract:

MapReduce is one of the important concepts of Hadoop that is used for data handling used by big companies today such as Google and Facebook. Here we divide each job into the map and reduce phases and try to complete the execution of the assigned task in a parallel form. In this paper, we suggest that it would be more efficient if we make the scheduler to work at the phase-level instead of the task-level. The reason is because the task demands a lot of requirements during its lifetime. For this very purpose, we introduce the concept called PRISM, which is aphase and information-aware scheduler for MapReduce and in this concept we divide the tasks into unequal parts called as phases and apply phase-level scheduling to these phases and achieve efficient resource usage.

1. INTRODUCTION

Today, companies depend entirely on large-scale data analysis, so they can make critical business decisions day by day. This is directed to the development of Map-Reduce, i.e.a parallel programming model that has become equivalent with large-scale and data-intensive calculations. Map-Reduce consists of a job, which is a collection of Map and Minimize
activities. These activities can be synchronously programmed on several machines, with a substantial reduction in work time.An essential component of a Map-Reduce system is your task planner. The main role of the activity planning program is to create a mapping and reduction planning activity that includes one or more jobs, minimizes job completion time and maximizes resource utilization. In many situations, the containment of heavy resources and the time of completion of long processes take place due to a planning with too many tasks performed simultaneously on a single machine. On the contrary, hunger occurs because of the improper use of resources and also because of a planning with very few simultaneous activities in a single machine.

The problem of job scheduling becomes significantly simpler to solve, assuming that all map activities (and all reduction activities) have consistent resource requirements, such as CPU, memory, disk, and network bandwidth. However, this hypothesis is used to simplify the programming problem with current systems of Map Reduction, such as Hadoop Map-Reduce Version 1.x. This system uses a simple slot-based resource allocation scheme, in which the physical resources of each machine take on the amount of indistinguishable slots that can be allocated to activities.

This document offers PRISM, which is a fine-stage programmer and resource information for map reduction clusters. PRISM realizes the conscious planning of resources at the level of the phases. Specifically, this document shows that, for Map-Reduce applications, the consumption of resources of the activity during the execution time can vary considerably from

one phase to another. Therefore, it is possible that the planner has a greater degree of parallelism even if it avoids the containment of resources, only when it comes to the demand for resources at the phase level. Therefore, in the end, this document has developed a phase-level programming algorithm with the aim of obtaining high work performance together with the appropriate use of resources.

2. Existing System

2.1 Hadoop MapReduce

MapReduce [10] is a parallel computing model for large-scale data -intensive computations. A MapReduce job consists of two types of tasks, i.e. the map task and the reduce task. A map task takes a keyvalue block as the input that is stored in the underlying distributed file system and runs a user-specified map function to of key-value output. Subsequently, a reduce task is responsible for collecting and applying specified reduce function on the collected key value pairs to produce the final output. Currently, the most popular implementation of MapReduce is Apache Hadoop MapReduce [1]. A Hadoop cluster consists of a collection of machines where one node will act as a master node and all the remaining n-1 nodes act as the slave node. The slave nodes execute the tasks assigned by the master node. The master node runs a resource manager (also known as a job tracker) that is responsible for scheduling tasks on slave nodes. Each slave node runs a local node manager (also known as a task tracker) that is responsible for launching and allocating resources for each task. To do so, the task tracker launches a Java Virtual Machine (JVM) that executes the corresponding map or reduce task. The original Hadoop MapReduce (i.e. version 1.*x* and earlier) adopts a slot-based resource allocation scheme. The scheduler assigns tasks to be executed to each machine based on the availability of the resources on each machine. The number of map and educe slots determine how the data are divided and allocated to each machine. As a Hadoop cluster is usually a multi-user system, many users can simultaneously submit jobs to the cluster. The job scheduling is performed by the resource manager in the master node, which maintains a list of jobs in the system. Here each slave node performs a small job and informs its completion via a heartbeat message (usually between 1-3 seconds) to the master node. The resource scheduler will use the provided information to make scheduling decisions. Today there are two commonly used schedulers that are: Capacity scheduler [2] and Fair scheduler [3]. These schedulers function on at task level.

2.2 MapReduce Job Phases

Current Hadoop job schedulers perform as task-level scheduling where initially a task given by the user to execute is divided into blocks or chunks which are of unequal size this is the map phase. In particular, a map task can be divided into 2 main phases: *map* and *merge*2. The Hadoop Distributed File System (HDFS) [4], where data blocks are stored across multiple slave nodes. In the map phase, a mapper fetches an input data block from the Hadoop Distributed File System (HDFS) [4] and applies the user - as with the Hadoop implementation, defined a map function on each record. The map function generates records that are serialized and collected into a buffer. When the buffer becomes full (i.e., content size exceeds a pre-specified threshold), the content of the buffer will be written to the local disk. Lastly, the mapper executors a merge phase to group the output records based on the intermediary keys, and store the records

in multiple files so that each file can be fetched a corresponding reducer. Similarly, the execution of a reduce task can be divided into 3 phases: *shuffle*, *sort*, and *reduce*. In the shuffle phase, the reducer fetches the output file from the local storage of each map task and then places it in a storage buffer that can be either in memory or on disk depending on the size of the content. At the same time, the reducer also launches one or more threads to perform local merge sort in order to reduce the running time of the subsequent sort phase. Once all the map output records have been collected, the sort phase will perform one final sorting procedure to ensure all collected records are in order. Finally, in 1. Other resources such as disk and network I/O are yet to be supported by Hadoop Yarn.

2. We use the same phase names

3. PRISM

3.1 Prism Architecture

As it is cleared from the definition that, PRISM is a resource information-aware Map Reduce scheduler that distributes tasks into phases in a fine-grained manner, where each phase has a persistent resource usage profile and implements scheduling at the level of phases. During the execution time of a task, resource usage analysis may lead to ineffective scheduling decisions. Because of this, at run-time, if the resource allotted to a task is higher than the existing resource usage, then the idle resources are wasted. On the other hand, if the resources allotted to the task is much less than the actual resource demand, then the resource can suffer from a situation called, bottleneck, which may slow down task execution.

Therefore, a fine-grained, phase-level scheduling mechanism has been introduced. This allocates the resources according to the demand of the phase that each task is currently executing. Due to this fine-grained resource allocation, not a single task suffers from either bottleneck or starvation problem.

An overview of the PRISM architecture is shown in Fig. 1. PRISM comprises of four main modules: resource manager, local node managers, a job progress monitor and a phase-based scheduler. Initially, Resource Manager (also known as a job tracker), is responsible for scheduling tasks on each local node. Then, Local Node Manager, (also known as a task tracker) that coordinate phase transitions with the scheduler. Next is Job Progress Monitor, which is responsible to capture phase-level progress information. Finally, Phase-Based Scheduler, i.e., a fine-grained, phase-level scheduling mechanism that allocates resources according to the demand of executing phase (neither overflow nor underflow).

3.2 Phase-Level Scheduling Mechanism

In this mechanism, there are some steps which are followed during the execution of PRISM. These steps are:

(Step 1): Each local node manager sends a heartbeat message to the phase-based scheduler periodically. As soon as a task requests to be scheduled, then the scheduler immediately responses to the heartbeat message with a task scheduling request.

(Step 2): Then, the local node manager initiates the task.

(Step 3): As and when a task completes implementing a particular phase (shuffle phase), then the task requests the local node manager for permission to start the next phase (e.g. reduce phase).

(Step 4): The local node manager then forwards this permission request to the phase-based scheduler.

(Step 5): Finally, once the task is permitted to execute the next phase (reduce phase), the local node manager grants permission to process that task and once the task is completed; the task

status is received by the local node manager and then dispatched to the phase-based scheduler.

PRISM requires constant phase-level resource information for each job to perform phase-level scheduling. In this way, the entire task is implemented. Each phase travels through all the above steps and finally get completed successfully.

4. CONCLUSION

In this Paper, Map-Reduce is used as a popular programming model to calculate data-intensive jobs. PRISM, which is a fine-grained resource allocation / reduction planner, divides tasks into phases and also performs phase-level programming. Due to the use of this phase-level programming, there is an improvement in the use of resources.The planning algorithm used by PRISM contributes to minimizing work execution time compared to current Hadoop programmers. In general, PRISM achieves high work performance. Finally, the future purpose of this document will be to improve the scalability of PRISM through the use of distributed programmers.

REFERENCES

[1] Hadoop Map Reduce distribution [Online]. Available: http://hadoop.apache.org, 2015.

[2] Hadoop Fair Scheduler [Online]. Available: http://hadoopapache.org/docs/r0.20.2/fair_scheduler.html, 2015.

[3] Hadoop Distributed File System [Online]. Available: hadoop.apache.org/docs/hdfs/current/, 2015.

[4] The Next Generation of Apache Hadoop MapReduce [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html, 2015.

[5] R. Boutaba, L. Cheng, and Q. Zhang, "On cloud computational models and the heterogeneity challenge," J. Internet Serv. Appl., vol. 3, no. 1, pp. 1–10, 2012.

[6] T. Condie, N. Conway, P. Alvaro, J. Hellerstein, K. Elmeleegy, and R. Sears, "MapReduce online," in Proc. USENIX Symp. Netw. Syst. Des. Implementation, 2010, p. 21.

[7] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," Commun. ACM, vol. 51, no. 1, pp. 107–113, 2008.

[8] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in Proc. USENIX Symp. Netw. Syst. Des. Implementation, 2011, pp. 323–336.

[9] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. Cetin, and S. Babu, "Starfish: A self-tuning system for big data analytics," in Proc. Conf. Innovative Data Syst. Res., 2011, pp. 261–272.

[10] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, and K. Talwar, "Quincy: Fair scheduling for distributed computing clusters," in Proc. ACMSIGOPS Symp. Oper. Syst. Principles, 2009, pp. 261–276.

[11] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang. "Multi-resource allocation: Flexible tradeoffs in a unifying framework," in Proc. IEEE Int. Conf. Comput.Commun., 2012, pp. 1206–1214.

[12] J. Polo, C. Castillo, D. Carrera, Y. Becerra, I. Whalley, M. Steinder, J. Torres, and

E. Ayguad_e, "Resource-aware adaptive scheduling for MapReduce clusters," in Proc. ACM/IFIP/USENIX Int. Conf. Middleware, 2011, pp. 187–207.

[13] A. Rasmussen, M. Conley, R. Kapoor, V. T. Lam, G. Porter, and A. Vahdat, "ThemisMR: An I/O-Efficient MapReduce," in Proc. ACM Symp. Cloud Compute. 2012, p. 13.

[14] A. Verma, L. Cherkasova, and R. Campbell, "Resource provisioningframework for MapReduce jobs with performance goals," in Proc. ACM/IFIP/USENIX Int. Conf. Middleware, 2011, pp. 165–186.

[15] D. Xie, N. Ding, Y. Hu, and R. Kompella, "The only constant is change: Incorporating time-varying network reservations in data centers," in Proc. ACM SIGCOMM, 2012, pp. 199–210.

[16] Y. Yu, M. Isard, D. Fetterly, M. Budiu, _ U. Erlingsson, P. Gunda, and J. Currey, "DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language," in Proc. USENIX Symp. Oper. Syst. Des. Implementation, 2008, pp. 1–14.

[17] M. Zaharia, D. Borthakur, J. SenSarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in Proc. Eur. Conf. Comput. Syst., 2010, pp. 265–278.

[18] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in Proc. USENIX Symp. Oper. Syst. Des. Implementation, 2008, vol. 8, pp. 29–42.