International Journal of Computer Sciences
December'2015 Issue
ISSN Number: 4142 3453

Automatic Slot Configurations for Heterogeneous Hadoop Clusters

Ch Srikanth[1] , Koona Hemanath[2], Y.Subba Rayudu[3]

[1] Assist.Prof of cse Department Institute of Aeronautical Engineering hyd

[2] Assistant Professor of Cse Deparment , MLR institute of Technology,Hyd

[3] Assistant Professor of Cse Deparment ,Institute of Aeronautical Engineering ,Hyd

***Abstract:** In the era of big data, one of the most significant research areas is cluster computing for large-scale data processing. Many cluster computing frameworks and cluster resource management schemes were recently developed to satisfy the increasing demands on large volume data processing. Among them, Apache Hadoop became the de facto platform that has been widely adopted in both industry and academia due to its prominent features such as scalability, simplicity and fault tolerance. The original Hadoop platform was designed to closely resemble the MapReduce framework, which is a programming paradigm for cluster computing proposed by Google. Recently, the Hadoop platform has evolved into its second generation, Hadoop YARN, which serves as a unified cluster resource management layer to support multiplexing of different cluster computing frameworks. A fundamental issue in this field is that given limited computing resources in a cluster, how to efficiently manage and schedule the execution of a large number of data processing jobs. Therefore, in this dissertation, we mainly focus on improving system efficiency and performance for cluster computing platforms, i.e., Hadoop MapReduce and Hadoop YARN, by designing the following new scheduling algorithms and resource management schemes*

***Keywords:** MapReduce jobs, Hadoop scheduling, reduced makespan, slot configuration*

I. INTRODUCTION

The past decade has seen the rapid development of cluster computing platforms, as growing data volumes require more and more scalable applications. In the age of big data, the data that needs to be processed by many companies and research projects is difficult to fit into traditional database and software techniques due to its increasing volume, velocity, and variety. For example, Google reported to process more than 20 PB of data per day in 2008 [1], and Facebook reported that they process between 10-15 TB of compressed data every day in 2010 [2]. This amount of data definitely cannot be handled by a single computer. It is also not cost efficient and scalable to process such big data with a single high performance super computer. Therefore, many paradigms are designed for efficiently processing big data in parallel with commercial computer clusters. Among them, MapReduce [1] and its open source implementation Apache Hadoop [3] have emerged as the de facto platform for processing large-scale semistructured and unstructured data. Hadoop MapReduce has been widely adopted by many companies and institutions [4] mainly due to the following advantages. First, Hadoop is easy for both administrators and developers to deploy and develop new applications. Moreover, it is scalable. A Hadoop cluster could be easily scaled from a few nodes to thousands of nodes. Last but not least, a Hadoop MapReduce cluster is fault tolerant to node failures, which greatly improves the availability of Hadoop platforms. MapReduce [1] has become the leading paradigm in recent years for parallel big data processing. Its open source implementation Apache Hadoop [2] has also emerged as a popular platform for daily data processing and information analysis. With the rise of cloud computing, MapReduce is no longer just for internal data process in big companies. It is now convenient for a regular user to launch a MapReduce cluster on the cloud, e.g., AWS MapReduce, for data-intensive applications. When more and more applications are adopting the

MapReduce framework, how to improve the performance of a MapReduce cluster becomes a focus of research and development. Both academia and industry have put tremendous efforts on job scheduling, resource management, and Hadoop applications [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, we 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 classic Hadoop cluster includes a 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 daemon 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. Our experiments have shown that the slot configuration has a significant impact on system performance. The Hadoop framework, however, uses 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.

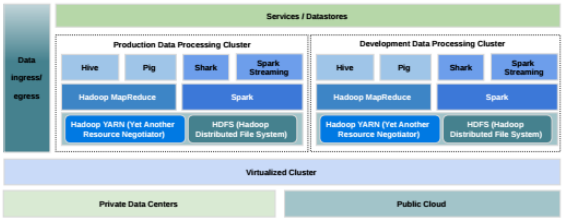

Figure 1.1: Typical deployment of large-scale data processing systems.

II. FEATURES OF CLUSTER COMPUTING PLATFORMS AND APPLICATIONS

In this section, we discuss some prominent features of current cluster computing frameworks for large-scale data processing and cluster computing applications. Our work is motivated by these features. (1) Diversity of workloads. Many cluser computing platforms, such as Hadoop, were designed for optimizing a single large job or a batch of large jobs. However, actual workloads are usually much more complex in real world deployed platforms. The complexity is reflected in three aspects. First, a large-scale data processing cluster, once established, is no longer dedicated to a particular job, but to multiple jobs from different applications or users. For example, Facebook [2] allows multiple applications and users to submit their ad hoc queries to the shared Hive-Hadoop clusters. Second, data processing service is becoming prevalent and open to numerous clients from the Internet, like today's search engines service. For example, a smartphone user may send a job to a MapReduce cluster through an App asking for the most popular words in the tweets logged in the past three days. Third, the characteristics of data processing jobs vary a lot. It is essentially caused by the diversity of user demands. Recent analysis on MapReduce workloads of current enterprise clients [13], e.g., Facebook and Yahoo!, has revealed the

diversity of MapReduce job sizes which range from seconds to hours. Overall, workload diversity is common in practice when jobs are submitted by different users. For example, some users run small interactive jobs while other users submit large periodical jobs; on the other hand, some users run jobs for processing files with similar sizes while jobs from other users have quite different sizes. (2) Various performance considerations. As discussed before, cluster computing platforms serve diverse workloads of different properties and from different sources. These workloads usually have different primary performance considerations. For example, interactive ad hoc applications requiring good response times while makespan or deadlines are more important for periodical batch jobs. There is no single resource management scheme or application scheduler that is optimal for all performance metrics. The original FIFO scheduling policy of Hadoop MapReduce is designed for better batch execution

III. MAPREDUCE PROGRAMMING PARADIGM

There are two major phases in a typical MapReduce job, i.e., map phase and reduce phase. In the map phase, each mapper task processes one split/block of the input data (data is usually chuncked into small blocks and stored in a distributed file system, such as HDFS and S3), and produces intermediate data in the format of key/value pairs. Each intermediate data record is then sent to reduce task based on the value of its key. All data records with the same key are sent to the same reduce task. Therefore, each reduce task receives an exclusive sub set of the total intermediate data. The data transmitting process is called shuffle. A reduce task starts processing intermediate data and producing final results after receiving all associated key/value pairs through shuffle process. Within each phase, there are multiple distributed tasks, either map pers or reducers, running the same function independently to process their input data sets. Therefore, data processing in each stage can be performed in parallel in a cluster for performance improvement. If some tasks of a job fail or straggle, then only these tasks, instead of the entire job, will be re-executed. In the MapReduce framework, programmers only need to design appropriate map and reduce functions for their applications, without taking care of data flow, data distribution, failure recovery, and other implementation details.

Hadoop MapReduce has been widely adopted as the prime framework for large-scale data processing jobs in recent years. Although initially designed for batch job processing, Hadoop MapReduce platforms usually serve much more complex workloads that comes from multiple tenants in real world deployments. For example, Facebook [2], one of Hadoop's biggest champions, keeps more than 100 petabytes of Hadoop data on-line, and allows multiple applications and users to submit their ad-hoc queries to the shared Hive-Hadoop clusters. For those ad-hoc jobs, the average job response time becomes a prime performance consideration in the shared Hadoop MapReduce cluster. At the same time, the Hadoop cluster in Facebook also serves periodical batch jobs where the total completion length of jobs is of greater importance. In this section, we propose two different schemes for Hadoop MapReduce platform that aim to improve the system performance under different primary performance considerations. Scheduling policy plays an important role for improving job response times in Hadoop when multiple users compete for available resources in cluster. However, we found that the existing policies

supported by Hadoop MapReduce platform do not perform well in terms of job response times under heavy and diverse workloads. The default FIFO policy, which is originally designed for better total job completion length (i.e., makespan) for batch jobs, performs poorly in terms of average job response time. Since short jobs may stuck behind long jobs and have extremely long waiting time. Fair and Capacity policies mitigate the problem of FIFO by sharing total system resources among jobs from different queues. Such that short jobs can process immediately after submission without waiting for long jobs if they are assigned to a different queue from the long jobs. However, we found that Fair policy could also perform poorly in terms of average job response times under certain workload patterns.

REFERENCES

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.

[2] Apache Hadoop. [Online]. Available: http://hadoop.apache.org/

[3] M. Zaharia, D. Borthakur, J. S. Sarma et al., "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in EuroSys'10, 2010.

[4] A. Verma, L. Cherkasova, and R. H. Campbell, "Two sides of a coin: Optimizing the schedule of mapreduce jobs to minimize their makespan and improve cluster performance," in MASCOTS' 12, Aug 2012.

[5] M. Isard, Vijayan Prabhakaran, J. Currey et al., "Quincy: fair scheduling for distributed computing clusters," in SOSP'09, 2009, pp. 261–276.

[6] A. Verma, Ludmila Cherkasova, and R. H. Campbell, "Aria: Automatic resource inference and allocation for mapreduce environments," in ICAC'11, 2011, pp. 235–244.

[7] J. Polo, D. Carrera, Y. Becerra et al., "Performance-driven task coscheduling for mapreduce environments," in NOMS'10, 2010.

[8] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth et al., "Apache hadoop yarn: Yet another resource negotiator," in Proceedings of the 4th annual Symposium on Cloud Computing. ACM, 2013, p. 5.

[9] X. W. Wang, J. Zhang, H. M. Liao, and L. Zha, "Dynamic split model of resource utilization in mapreduce," in DataCloud-SC '11, 2011.

[10] J. Polo, C. Castillo, D. Carrera et al., "Resource-aware adaptive scheduling for mapreduce clusters," in Proceedings of the 12th ACM/IFIP/USENIX international conference on Middleware, 2011.