Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 5.258

IJCSMC, Vol. 5, Issue. 4, April 2016, pg.344 – 355

Scheduler for Efficient Task Assignments in Hadoop

Saranay. K[1], NaveenDurai.K[2], Surendar Reddy.N[3]
[1]PG Scholar, Department Of CSE, SECE, Coimbatore & Anna University, India
[2]Assistant Professor, Department Of CSE, SECE, Coimbatore & Anna University, India
[3]Senior Solution Architect Tech Mahindra Chennai, India
[1] saranikannan@gmail.com; [2] naveendurai.k@sece.ac.in; [3] SN00356230@gmail.com

Abstract— The MapReduce paradigm and its open source implementation Hadoop are emerging as an important standard for large scale data-intensive processing in both industry and academia. A MapReduce cluster is typically shared among multiple users with different types of workloads. When a flock of jobs are concurrently submitted to a MapReduce cluster, they compete for the shared resources and the overall system performance in terms of job response times, might be seriously degraded. Therefore, one challenging issue is the ability of efficient scheduling in such a shared MapReduce environment. However, we find that conventional scheduling algorithms supported by Hadoop cannot always guarantee good average response times under different workloads. The proposed approach uses performance models to match hadoop tasks to the servers that will benefit them the most, and deadline-aware scheduling to effectively order incoming jobs. It use admission control to meet deadlines even when resources are overloaded. The combination of these schedulers allows data center administrators to safely mix resource intensive hadoop jobs with latency sensitive applications, and still achieve predictable performance for both. The proposed system implement system using hadoop, and our evaluation shows that our schedulers allow a mixed cluster to reduce response times by more than tenfold compared to the existing job size scheduler, while meeting more hadoop deadlines and lowering total task execution times by 6.5%.

Keywords— Big data, Hadoop, HDFS, MapReduce, Schedulers

I. INTRODUCTION

"Big Data consists of **extensive** large datasets – primarily in the characteristics of volume, variety, velocity, and/or variability – that requires a scalable architecture for efficient storage, analysis, and manipulation. MapReduce has become an important paradigm for parallel data-intensive luster programming due to its simplicity and flexibility. Essentially, it is a software framework that allows a cluster of computers to process a large set of structured or unstructured data in parallel. MapReduce's users are quickly growing. Apache Hadoop is an open source implementation of MapReduce that has been widely adopted in the industry area. With the rise of cloud computing, it becomes more convenient for IT business to set a cluster of servers in the cloud and launch a batch of MapReduce jobs. Therefore, there are a large variety of data-intensive applications using the MapReduce framework. In a classic Hadoop system, each MapReduce job is partitioned into small tasks which are distributed and executed across multiple machines. There are two kinds of tasks, i.e., map tasks and reduce tasks. Each map task applies the same map function to process a block of the input data and produces

intermediate results in the format of key-value pairs. The intermediate data will be partitioned by hash functions and fetched by the corresponding reduce tasks as their inputs. Once all the intermediate data has been fetched, a reduce task starts to execute and produce the final results. The Hadoop implementation closely resembles the MapReduce framework.

A single master node is adopted to manage distributed slave nodes. The master node communicates with slave nodes with heartbeat messages which consist of status information of slaves. Job scheduling is performed by a centralized jobtracker routine in the master node. The scheduler assigns tasks to slave nodes which have free resources and response to the heartbeats as well. The resources in each slave node are represented as map/reduce slots. Each slave node has a fixed number of slots, and each map (resp. reduce) slot only processes one map (resp. reduce) task at one moment.

1.1 BIG DATA

A. Characteristics

Bigdata involves Five main characteristics are,

- Volume
- Variety
- Velocity
- Veracity
- Value

Volume - It covers the size of the data needed for management. There is more data than ever before, its size keep on growing exponentially: 90% of all the data available today were created in the last two years. A short time ago, we were talking about gigabytes, we are talking now relatively about terabytes, petabytes, exabytes and even zettabytes.

Variety - The category to which the data belongs is defined by variety. This helps the people who are closely analyzing the data. Heterogeneous, complex, and variable data, which are generated in formats as different as e-mail, social media, video, images, blogs and sensor data.

Figure 1: 5V's in Bigdata

Velocity - It specifies the speed of generation of data or how fast the data is generated. We are focused on getting knowledge from the data arriving as streams in real time. More we focus on real time; more we are in big data problem. Gradually, the immediate treatment of data would be the key element of a model big data.

Variability - This is the most crucial point. Confidence in the accuracy of data collected and presented is an issue more and more real as the number of sources generating data increases. It reflects the emphasis of the need for quality data in a Big Data system. It is also called as Veracity.

Value – It represents the value that can be drawn from these data and the uses they produce. Sorting data is then essential. It is essential to properly select the data to be analyzed, based on its activity and especially its objectives.

II. HADOOP OVERVIEW

When data sets go beyond a single storage capacity, it is necessary to distribute them to multiple independent computers. Trans-computer network storage file management system is called distributed file system. A typical Hadoop distributed file system contains thousands of servers, each server stores partial data of file system. HDFS cluster configuration is simple. It just needs more servers and some simple configuration to improve the Hadoop cluster computing power, storage capacity and IO bandwidth. In addition HDFS achieves reliable data replication, fast fault detection and automatic recovery, etc.Hadoop is an open source framework, from the Apache foundation, capable of processing large amounts of heterogeneous data sets in a distributed fashion across clusters of commodity computers and hardware using a simplified programming model. Hadoop provides a reliable shared storage and analysis system.

2.1 Architecture of Hadoop

Below is a high-level architecture of multi-node Hadoop Cluster.

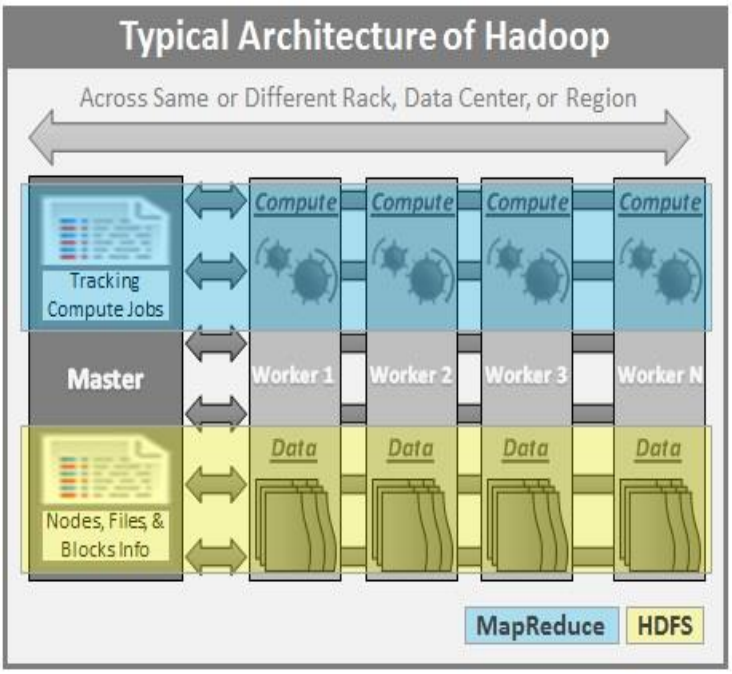

Figure 2.1 Architecture of Hadoop

Here are few highlights of the Hadoop Architecture:

- Hadoop works in a master-worker / master-slave fashion.
- Hadoop has two core components: HDFS and MapReduce.
- **HDFS (Hadoop Distributed File System)** offers a highly reliable and distributed storage, and ensures reliability, even on a commodity hardware, by replicating the data across multiple nodes. Unlike a regular file system, when data is pushed to HDFS, it will automatically split into multiple blocks (configurable parameter) and stores/replicates the data across various datanodes. This ensures high availability and fault tolerance.
- **MapReduce** offers an analysis system which can perform complex computations on large datasets. This component is responsible for performing all the computations and works by breaking down a large complex computation into multiple tasks and assigns those to individual worker/slave nodes and takes care of coordination and consolidation of results.
- The master contains the Namenode and Job Tracker components.
    - **Namenode** holds the information about all the other nodes in the Hadoop Cluster, files present in the cluster, constituent blocks of files and their locations in the cluster, and other information useful for the operation of the Hadoop Cluster.
    - **Job Tracker** keeps track of the individual tasks/jobs assigned to each of the nodes and coordinates the exchange of information and results.
- Each Worker / Slave contains the Task Tracker and a Datanode components.
    - **Task Tracker** is responsible for running the task / computation assigned to it.

- **Datanode** is responsible for holding the data.

➢ The computers present in the cluster can be present in any location and there is no dependency on the location of the physical server.

2.2 MAPREDUCE OVERVIEW

In distributed data storage, when improved processing the data, we need to consider much, such as synchronization, concurrency, load balancing and other details of the underlying system. It makes the simple calculation become very complex. MapReduce programming model was proposed in 2004 by the Google, which is used in processing and generating large data sets implementation. This framework solves many problems, such as data distribution, job scheduling, fault tolerance, machine to machine communication, etc.

They form the core of task.

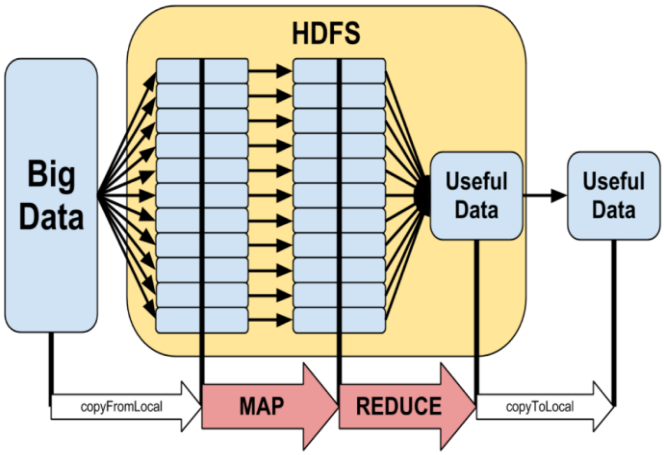

Figure 2.2 MAP/Reduce Jobs

A. Mapper

Map function requires the user to handle the input of a pair of key value and produces a group of intermediate key and value pairs. <key,value> consists of two parts, value stands for the data related to the task, key stands for the "group number " of the value . MapReduce combine the intermediate values with same key and then send them to reduce function. Map algorithm process is described as follows:

Step1. Hadoop and MapReduce framework produce a map task for each InputSplit, and each InputSplit is generaed by the InputFormat of job. Each <Key,Value> corresponds to a map task.

Step2. Execute Map task, process the input <key,value> to form a new <key,value>. This process is called "divide into groups". That is, make the correlated values correspond to the same key words. Output key value pairs that do not required the same type of the input key value pairs. A given input value pair can be mapped into 0 or more output pairs.

Step3. Mapper's output is sorted to be allocated to each Reducer. The total number of blocks and the number of job reduce tasks is the same. Users can implement Partitioner interface to control which key is assigned to which Reducer.

B. Reducer

Reduce function is also provided by the user, which handles the intermediate key pairs and the value set relevant to the intermediate key value. Reduce function mergers these values, to get a small set of values. the process is called "merge ". But this is not simple accumulation. There are complex operations in the process. Reducer makes a group of intermediate values set that associated with the same key smaller.

In MapReduce framework, the programmer does not need to care about the details of data communication, so <key,value> is the communication interface for the programmer in MapReduce model. <key,value> can be seen as a "letter", key is the letter's posting address, value is the letter's content. With the same address letters will be delivered to the same place. Programmers only need to set up correctly <key,value>, MapReduce framework can automatically and accurately cluster the values with the same key together. Reducer algorithm process is described as follows:

Step1. Shuffle. Input of Reducer is the output of sorted Mapper. In this stage, MapReduce will assign related block for each Reducer.

Step2. Sort. In this stage, the input of reducer is grouped according to the key (because the output of different mapper may have the same key). The two stages of Shuffle and Sort are synchronized;

Step3. Secondary Sort. If the key grouping rule in the intermediate process is different from its rule before reduce. We can define a Comparator. The comparator is used to group intermediate keys for the second time.

Map tasks and Reduce task is a whole, cannot be separated. They should be used together in the program. We call a MapReduce the process as an MR process. In an MR process, Map tasks run in improved, Reduce tasks run in improved, Map and Reduce tasks run serially. An MR process and the next MR process run in serial, synchronization between these operations is guaranteed by the MR system, without programmer's involvement.

2.3 HDFS FILE SYSTEM

HDFS is an interesting technology in that it provides data distribution, replication, and automatic recovery in a user-space filesystem that is relatively easy to configure and, conceptually, easy to understand. However, its true utility comes to light when map/reduce jobs are executed on data stored in HDFS.

As the name implies, map/reduce jobs are principally comprised of two steps: the map step and the reduce step. The overall workflow generally looks something like this:

The idea underpinning map/reduce--bringing compute to the data instead of the opposite--should sound like a very simple solution to the I/O bottleneck inherent in traditional parallelism. However, the devil is in the details, and implementing a framework where a single large file is transparently diced up and distributed across multiple physical computing elements (all while appearing to remain a single file to the user) is not trivial.

HDFS exists as a filesystem into which you can copy files to and from in a manner not unlike any other filesystem. Many of the typical commands for manipulating files (ls, mkdir, rm, mv, cp, cat, tail, and chmod, to name a few) behave as you might expect in any other standard filesystem (e.g., Linux's ext4).

The magical part of HDFS is what is going on just underneath the surface. Although it appears to be a filesystem that contains files like any other, in reality those files are distributed across multiple physical compute nodes:

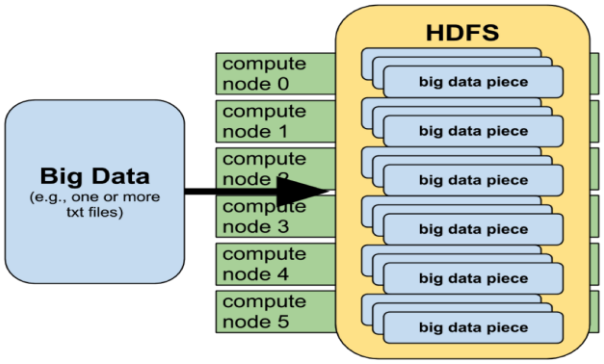

Figure 1.3 Overview HDFS

When you copy a file into HDFS as depicted above, that file is transparently sliced into 64 MB "chunks" and replicated three times for reliability. Each of these chunks are distributed to various compute nodes in the Hadoop cluster so that a given 64 MB chunk exists on three independent nodes. Although physically chunked up and distributed in triplicate, all of your interactions with the file on HDFS still make it appear as the same single file you copied into HDFS initially. Thus, HDFS handles all of the burden of slicing, distributing, and recombining your data for you.

III. MOTIVATIONS

3. 1 HADOOP JOB SCHEDULE

Hadoop is a general-purpose system that enables high-performance processing of data over a set of distributed nodes. But within this definition is the fact that Hadoop is a multi-tasking system that can process multiple data sets for multiple jobs for multiple users at the same time. This capability of multi-processing means that Hadoop has the opportunity to more optimally map jobs to resources in a way that optimizes their use.

A. FIFO scheduler

The original scheduling algorithm that was integrated within the JobTracker was called FIFO. In FIFO scheduling, a JobTracker pulled jobs from a work queue, oldest job first. This schedule had no concept of the priority or size of the job, but the approach was simple to implement and efficient.

B. Fair scheduler

The core idea behind the fair share scheduler was to assign resources to jobs such that on average over time, each job gets an equal share of the available resources. The result is that jobs that require less time are able to access the CPU and finish intermixed with the execution of jobs that require more time to execute. This behavior allows for some interactivity among Hadoop jobs and permits greater responsiveness of the Hadoop cluster to the variety of job types submitted. The fair scheduler was developed by Facebook.

You configure fair share in the mapred-site.xml file. This file defines the properties that collectively govern the behavior of the fair share scheduler. An XML file—referred to with the property mapred.fairscheduler.allocation.file defines the allocation of shares to each pool. To optimize for job size, you can set the mapread.fairscheduler.sizebasedweight to assign shares to jobs as a function of their size. A similar property allows smaller jobs to finish faster by adjusting the weight of the job after 5 minutes (mapred.fairscheduler.weightadjuster). Numerous other properties exist that you can use to tune loads over the nodes (such as the number of maps and reduces that a given TaskTracker can manage) and define whether preemption should be performed.

C. Capacity scheduler

The capacity scheduler shares some of the principles of the fair scheduler but has distinct differences, too. First, capacity scheduling was defined for large clusters, which may have multiple, independent consumers and target applications. For this reason, capacity scheduling provides greater control as well as the ability to provide a minimum capacity guarantee and share excess capacity among users. The capacity scheduler was developed by Yahoo!.

You configure the capacity scheduler within multiple Hadoop configuration files. The queues are defined within hadoop-site.xml, and the queue configurations are set in capacity-scheduler.xml. You can configure ACLs within mapred-queue-acls.xml. Individual queue properties include capacity percentage (where the capacity of all queues in the cluster is less than or equal to 100), the maximum capacity (limit for a queue's use of excess capacity), and whether the queue supports priorities. Most importantly, these queue properties can be manipulated at run time, allowing them to change and avoid disruptions in cluster use.

IV. RELATED WORK

4.1 MULTIPLE USERS

Typically, multiple users compete for the available slots in a MapReduce cluster when these users concurrently submit jobs to the cluster. As a result, the average job response times1, an important performance concern in MapReduce systems, might be seriously degraded. Recent studies found that MapReduce workloads often exhibit the heavy-tailed (or long-tailed) characteristics, where a MapReduce workflow consists of few but extremely large jobs and many small jobs. In such a MapReduce system, an efficient scheduling policy is a critical factor for improving system performance in terms of average job response times. However, we found that the existing policies supported by Hadoop do not perform well under heavy-tailed and diverse workloads. By default, Hadoop uses a FIFO (First-In-First-Out) scheduler which is originally designed to optimize the completion length (i.e., makespan) of a batch of jobs. However, it is not efficient in clusters which serve diverse workloads since small jobs will experience extremely long waiting time when submitted after a large job. Fair scheduler was proposed to improve the average job response times in shared Hadoop clusters by assigning to all jobs, on average, an equal share of resources over time. Specifically, we first develop a lightweight information collector that tracks statistic information of recently finished jobs from each user. A self-tuning scheduling policy is then designed to scheduler Hadoop jobs at two levels: the resource shares across multiple users are

tuned based on the estimated job size of each user; and the job scheduling for each individual user is further adjusted to accommodate to that user's job size distribution.

4.2 DEADLINE AWARE SCHEDULING ALGORITHM

Virtualization promised to dramatically increase server utilization levels, yet many data centers are still only lightly loaded. In some ways, big data applications are an ideal fit for using this residual capacity to perform meaningful work, but the high level of interference between interactive and batch processing workloads currently prevents this from being a practical solution in virtualized environments. Further, the variable nature of spare capacity may make it difficult to meet big data application deadlines.
The proposed system implement system using hadoop, and our evaluation shows that our schedulers allow a mixed cluster to reduce response times by more than tenfold compared to the existing job size scheduler, while meeting more hadoop deadlines and lowering total task execution times by 6.5%.

Deadline Aware Scheduling algorithm (DAS)

- Virtualization layer designed to minimize interference on high priority interactive services, and one in the Hadoop framework that helps batch processing jobs meet their own performance deadlines.
- Deadline-aware scheduling to effectively order incoming jobs.
- Minimize the cost of workflow execution while meeting a userdefined deadline.

4.3 MULTI NODE INSTALLATION

The explaining the Hadoop cluster environment using three systems (one master and two slaves); given below are their IP addresses.

- Hadoop Master: 192.168.1.1 (hadoop-master)
- Hadoop Slave: 192.168.1.2 (hadoop-slave)

Follow the steps given below to have Hadoop Multi-Node cluster setup. Once the single-node Hadoop clusters are up and running, we need to make configuration changes to make one Ubuntu box as a " *master*" (this box will also act as the slave) and the other Ubuntu box as the "*slave*".

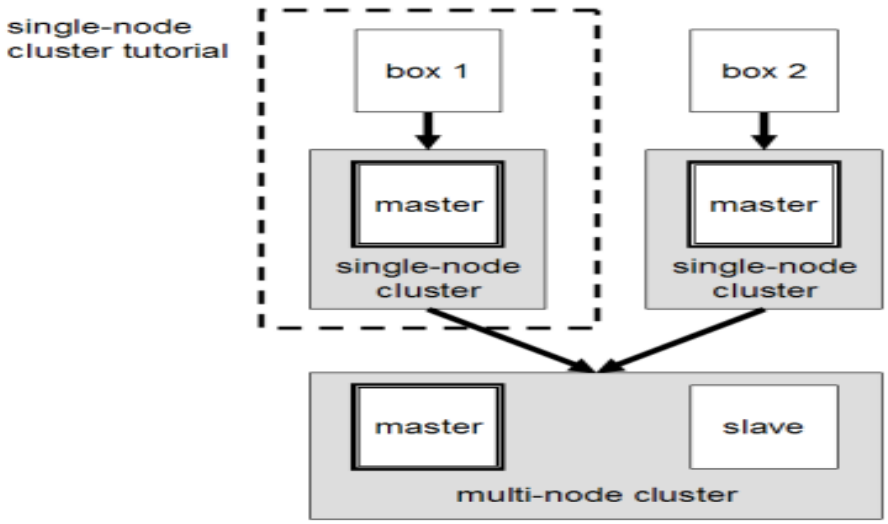

Figure 4.3 Multinode

Hadoop MapReduce is a software framework for easily writing applications which process big amounts of data in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. The term MapReduce actually refers to the following two different tasks that Hadoop programs perform:

- The Map Task: This is the first task, which takes input data and converts it into a set of data, where individual elements are broken down into tuples (key/value pairs).
- The Reduce Task: This task takes the output from a map task as input and combines those data tuples into a smaller set of tuples. The reduce task is always performed after the map task.

Typically both the input and the output are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks. The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for resource management, tracking resource consumption/availability and scheduling the jobs component tasks on the slaves, monitoring them and re-executing the failed tasks.

4.4 SLOT ALLOCATION BASED ON DEADLINE

- Add the code that recieve the deadline that the user defines
- The jobQueue is sorted by the deadline
- Add some statistical information eg, map task exec time, reduce task exec time and so on
- After finishing the job, whether the job meets the deadline
- A program running on each tasktracker node connetcts to the hadoop scheduler and sends resource statistics Using xentop periodically to get the available resources of tasktracker
- JobTracker has a server socket that receives these stats and stores them into a resource object For each tasktracker, server side will create a receiveThread to receive the stats
- For each job, predict task exection time based on the reources available
- Use canMeetDeadline to decide whether a job can meet the deadline based on the current slots
- When have free slots, firstly select missed deadline job from the jobQueue. If all jobs can meet the deadline, select the job that has the most progress on this free slot .

4.5 JOB EXECUTION

Word Count example reads text files and counts how often words occur. The input is text files and the output is text files, each line of which contains a word and the count of how often it occured, separated by a tab. Each mapper takes a line as input and breaks it into words. It then emits a key/value pair of the word and 1. Each reducer sums the counts for each word and emits a single key/value with the word and sum. As an optimization, the reducer is also used as a combiner on the map outputs. This reduces the amount of data sent across the network by combining each word into a single record.

(1) a job execution cost model that considers various parameters like map and reduce runtimes, input data sizes, data distribution, etc.,
(2) a Constraint-Based Hadoop Scheduler that takes user
deadlines as part of its input.

Estimation model determines the available slot based a set of assumptions:

Schedulability of a job is determined based on the proposed job execution cost model independent of the number of jobs running in the cluster. Jobs are only scheduled if specified deadlines can be met. After a job is submitted, schedulability test is performed to determine whether the job can be finished within the specified deadline or not. Free slots availability is computed at the given time or in the future irrespective of all the jobs running in the system. The job is enlisted for scheduling after it is determined that the job can be completed within the given deadline.

V. PERFORMANCE ANALYSIS

The proposed approach uses performance models to Multi node hadoop tasks to the servers that will benefit them the most, and deadline-aware scheduling to effectively order incoming jobs. It uses admission control to meet deadlines even when resources are overloaded. The combination of these schedulers allows data center administrators to safely mix resource intensive hadoop jobs with latency sensitive applications, and still achieve predictable performance for both. The proposed system implement system using hadoop, and our evaluation shows that our schedulers allow a mixed cluster to reduce response times by more than tenfold compared to the existing job size scheduler, while meeting more hadoop deadlines and lowering total task execution times by 6.5%.

Hadoop Technology	**Slot Size(MB)**			
	4	8	12	18
LsPS	20	40	63	68
Multi node Deadline Aware	18	34	49	54

Table 5.1 Compare the existing algorithm with proposed

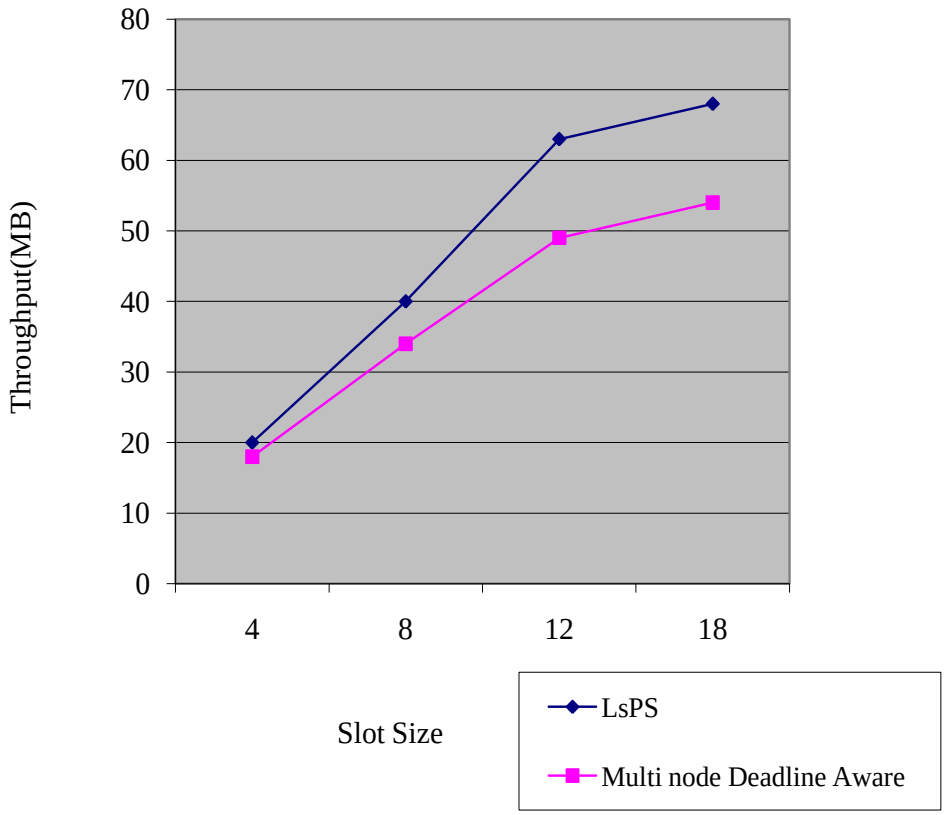

Figure 5.1 Comparison of slot allocation

Hadoop Technology	**No of Reduce Task**			
	1	2	3	4
LsPS	54.31	48.56	45.38	38.98
Multi node Deadline Aware	27.57	17.39	12.23	8.42

Table 5.2 MapReduce execution times of the LsPS basic approach and Deadline ware for different types of jobs, including WordCount

Large-scale MapReduce clusters that routinely process petabytes of unstructured and semi-structured data represent a new entity in the changing landscape of clouds. A key challenge is to increase the utilization of these MapReduce clusters. In this work, we consider a subset of the production workload that consists of MapReduce jobs with no dependencies. We observe that the order in which these jobs are executed can have a significant impact on their overall completion time and the cluster resource utilization. Our goal is to automate the design of a job schedule that minimizes the completion time (makespan) of such a set of MapReduce jobs. We offer a novel abstraction framework and a heuristic, called BalancedPools, that efficiently utilizes performance properties of MapReduce jobs in a given workload for constructing an optimized job schedule. Simulations performed over a realistic workload demonstrate that 15%-38% makespan improvements are achievable by simply processing the jobs in the right order.

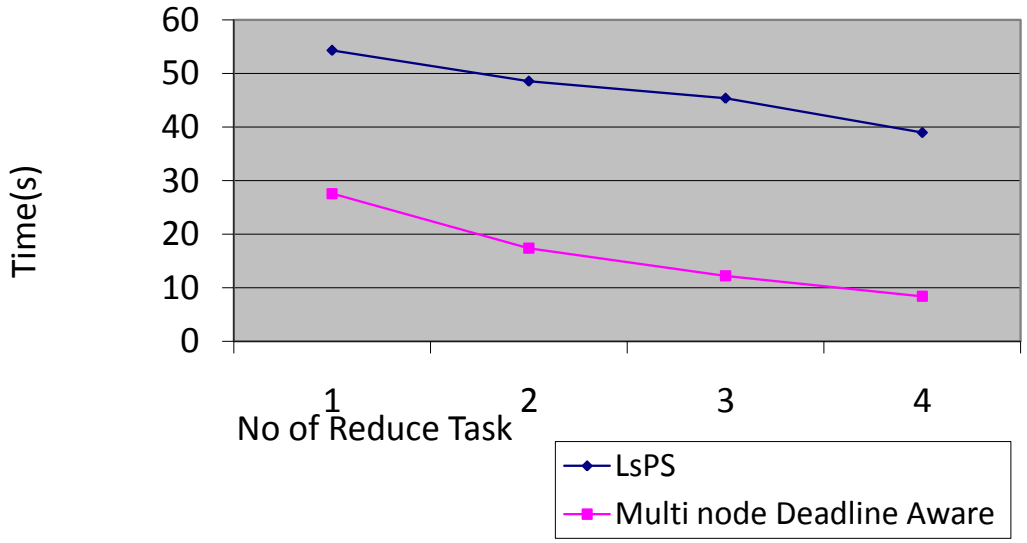

Figure 5.2 Comparison Reduce Task

Hadoop Technology	**No of User**			
	1	2	3	4
FIFO	251.36	280.06	235.33	330.20
Fair	121.18	149.95	118.36	248.00
LsPS	67.50	74.79	75.18	209.00
Deadline ware	43.76	64.98	68.21	96.43

Table 5.3 Average response times WordCount

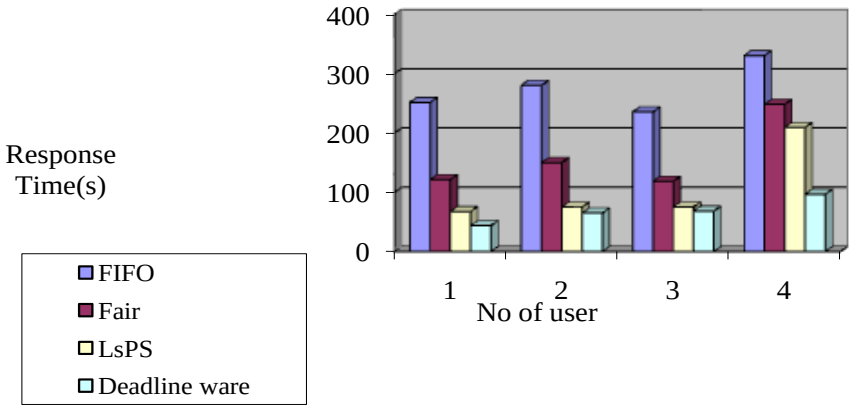

Figure 5.3 Comparison Response time

VI. CONCLUSION

The efficient scheduling framework for deadline-aware Map-Reduce workflows. In data centers, complex backend data analysis often utilizes a workflow that contains tens or even hundreds of interdependent Map-Reduce jobs. Meeting deadlines of these workflows is usually of crucial importance to businesses (for example, workflows tightly linked to time-sensitive advertisement placement optimizations can directly affect revenue).. In order to simplify the process of submitting workflows, solutions like emerge, which take a workflow configuration file as input and automatically submit its Hadoop jobs at the right time. To address this problem and at the same time honor the efficiency of Hadoop master node, DAS allows client nodes to locally generate scheduling plans which are later used as resource allocation hints by the master node. Under this framework design, we propose a novel scheduling algorithm that improves deadline satisfaction ratio by dynamically assigning priorities among workflows based on their progresses. We implement DAS by extending Hadoop-Our experiments over an 80-server cluster show that DAS manages to increase the deadline satisfaction ratio by 10% compared to state-of-the-art solutions, and scales up to tens of thousands of concurrently running workflows.

Acknowledgement

I am thankful to Mr.K.Naveendurai for his guidance, support and supervision. And also for providing the information and ready to help any time in completion of this Paper.

REFERENCES

[1] C. Tian, H. Zhou, Y. He, and L. Zha, "A dynamic mapreduce scheduler for heterogeneous workloads," in GCC'09. Eighth International Conference on. IEEE, 2009, pp. 218–224.

[2] M. Zaharia, D. Borthakur, J. S. Sarma et al., "Job scheduling for multi-user mapreduce clusters," University of California, Berkeley, Tech. Rep., Apr. 2009.

[3] Y. Chen, A. S. Ganapathi, R. Griffith, and R. H. Katz, "A methodology for understanding mapreduce performance under diverse workloads," University of California, Berkeley, Tech. Rep., 2010.

[4] K. Avrachenkov, U. Ayesta, P. Brown et al., "Discriminatory processor sharing revisited," in INFOCOM, Mar. 2005, pp. 784– 795.

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

[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] 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.
[8] M. Isard, Vijayan Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters," in SOSP'09, 2009, pp. 261–276.
[9] J. Wolf, D. Rajan, K. Hildrum, R. Khandekar, V. Kumar, S. Parekh, K.-L. Wu, and A. Balmin, "Flex: A slot allocation scheduling optimizer for mapreduce workloads," in Middleware 2010. Springer, 2010, pp. 1–20.
[10] 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. IEEE, 2012, pp. 11–18.

