International Journal of
Science and Engineering Investigations vol. 4, issue 43, August 2015
ISSN: 2251-8843

Scheduling Algorithms for MapReduce Framework

Hadi Yazdanpanah
Department of Computer, Islamic Azad University, Bushehr Branch, Bushehr, Iran
(Hadiyazdanpanah@outlook.com)

Abstract- Google proposed MapReduce as a simple and flexible parallel programming model, for large-scale distributed data processing. MapReduce framework allows users to quickly develop big-data applications and process big-data effectively. However, unexpected malfunction may be found in cloud environment because a distributed system consists of several hardware, and this malfunction often causes delay of overall processing. In MapReduce framework, the underlying runtime system automatically parallelizes the computation through large-scale nodes of machines, handles machine failures, and schedules inter-machine communication to make use of the network and disks efficiently. Scheduling is one of the important factors in MapRduce. In order to achieve good performance a MapReduce scheduler must avoid unnecessary data transmission. Hence different scheduling algorithms for MapReduce are necessary to provide good performance. how to schedule the service resources to achieve the lowest cost becomes more and more important. In this paper, we describe the overview of fifteen different scheduling algorithms for MapReduce in Hadoop and their scheduling issues and problems. At the end, Advantages and disadvantages of these algorithms are identified.

Keywords- *Scheduling algorithm, MapReduce, Hadoop*

I. INTRODUCTION

As a popular programming model in cloud-based data processing environment, MapReduce and Hadoop [1] is Apache's open source implementation of the MapReduce framework, are widely used both in industry and academic researches. MapReduce [2] is proposed by Google in 2004 and has become a popular parallel computing framework for large-scale data processing since then. It is best suited for embarrassingly parallel and data-intensive tasks. It is designed to read large amount of data stored in a distributed file system such as Google File System (GFS) [3], process the data in parallel, aggregate and store the results back to the distributed file system. In a typical MapReduce job, the master divides the input files into multiple map tasks, and then schedules both map tasks and reduce tasks to worker nodes in a cluster to achieve parallel processing [4]. The two major performance metrics in MapReduce are job execution time and cluster throughput.

Nowadays, requirements for huge data processing are increasing, such as machine learning [5], scientific analysis [6], astrophysics [7] and etc. There are several programming model for processing massive data, such as Microsoft Dryad is another parallel computing framework [8], Scatter-Gather-Merge [9], and MapReduce [2]. MapReduce is a distributed programming model for expressing distributed computation on massive amounts of data and an execution framework for large-scale data processing on clusters of commodity servers, it has been implemented in multi environments, such as Mar [10], Phoenix [11], and Hadoop [12].

One of the most important advantages of MapReduce is its convenience, such that, programmers can process massive data without knowing the details of distributed implementation, and users can process large scale of data by only providing the Map and Reduce interface. MapReduce programming framework enables controlling huge amount of data fast and efficiently by cooperation of many nodes [13].

The process of scheduling parallel tasks determines the order of task execution and the processor to which each task is assigned. Typically, an optimal schedule is achieved by minimizing the completion time of the last task. Finding the optimal schedule has long been known as an NP-complete problem in both homogeneous and heterogeneous environments [14]. Besides completion time, fairness is another important criterion for scheduling tasks if there are multiple jobs consisting of tasks to schedule. Fairness among jobs should be considered to keep any jobs from starving or being over penalized.

The initial MapReduce model was designed for off-line data processing. However, it is now popularly applied in heterogeneous, sharing and multi-user environments. The research of the MapReduce scheduling algorithm mainly in five areas: (1) the data locality of the MapReduce tasks. It is the effect of the data distribution to task scheduling; (2) fault-tolerant scheduling and expectation execution time in a heterogeneous environment; (3) resource sharing: for the hadoop cluster, how to share the computing resources through scheduling the user groups; (4) resource aware scheduling algorithm. It is based on the status of the cluster resources , such as memory, disk IO, network, and other factors; (5) real-time scheduling. It is the study for the MapReduce real-time scheduling model. Nowadays, some MapReduce scheduling algorithms exist. Basic features such as data locality, user priority, fault tolerant and fairness are all considered by these algorithms.

The rest of the paper is organized as follows: Section 2 provides a background on Hadoop and MapReduce Mechanisms. In Section 3, we introduce the MapReduce Scheduling Algorithms. In Section 4 we analyzes and consider advantage and disadvantage in the form of table. In Section 5 conclude the paper.

II. BACKGROUND

In this section, we briefly describe how a Hadoop and MapReduce Computing model work.

A. *Hadoop*

Hadoop is an implementation of MapReduce programming framework, which is an open source [15]. Hadoop runs over a distributed file system called Hadoop Distributed File System (HDFS) which has the same architecture as Google File System [16]. HDFS has a master/slave architecture. HDFS consists of one the master server, called Namenode and there are a number of slaves, called Datanodes. Namenode which controls several Datanodes, and the Datanodes store actual data. Namenode supervises metadata such as information of directories, access log from users, detail of data location, and system logs. Datanode keeps data in Blocks. A Block is a basic unit for data storing in HDFS. Figure 1 briefly describes the Hadoop Architecture.

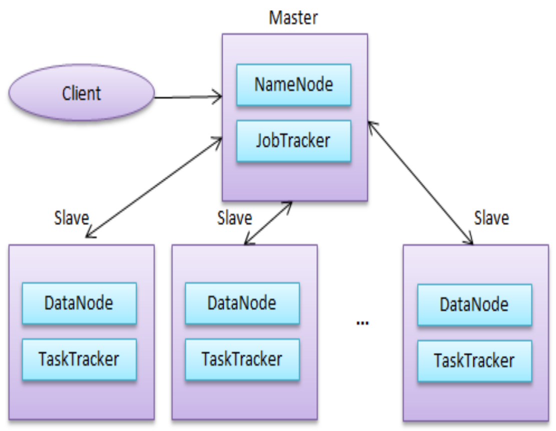

Figure 1. Hadoop Architecture.

B. *MapReduce Computing Model*

MapReduce [2] is a software framework introduced by Google to support distributed processing on large data sets on a cluster of computers. The input data is stored in a distributed file system, divided into a number of splits. Each split is loaded and processed by a number of map tasks, which in turn generate corresponding intermediate data that is grouped by keys. Then the intermediate data is shuffled (i.e., sent to corresponding reduce tasks according to the key) and processed by reduce tasks. Each of the reduce tasks is responsible for a range of key space and produces the final output, which is stored back to the distributed file system. Basically a MapReduce job goes through the following three phases.

Map phase: The map phase consists of a number of map tasks. Each task is responsible for reading a block (split) of the input file, applying a user-defined map function, and producing map intermediate data. Typically the number of map tasks is much larger than the number of workers.

Shuffle phase: The shuffle phase consists of a number of shuffle tasks. Each shuffle task is responsible for reading and sorting a subset of map intermediate data hashed on map output keys to that shuffle task and producing shuffle intermediate data. The map and shuffle phases can overlap because shuffle tasks can start reading map intermediate data as soon as any map task finishes.

Reduce phase: The reduce phase consists of a number of reduce tasks. Each reduce task has a corresponding shuffle task. A reduce task reads the corresponding shuffle intermediate data, applies user-defined reduce function and produces the final results. Typically the number of shuffle/reduce tasks is set to be a little lower than the number of workers, but sometimes it is larger.

MapReduce can be divided into Map function and Reduce function. In Map function, data is split into a set of small key/value pairs. A Map task processes one set of the key/value pairs. A JobTracker designs schedules of Map tasks by analyzing and subdividing jobs. The JobTracker assigns scheduled Map tasks to each TaskTracker and TaskTrackers process Map tasks. When a Map task is finished, the TaskTracker reports to the JobTracker about the status and stores the intermediate results of key/value pairs to its local file system. If there are unporecessed Map tasks, the JobTracker assigns a new Map task to the TaskTracker.

In Reduce function, intermediate results of key/value pairs created by Map tasks are sorted and merged. Also, a Reduce task is a unit of process to perform users' request like a Map task. At first, a JobTracker assigns a Reduce task to a TaskTracker. The assigned TaskTracker copies intermediate results of key/value pairs from other TaskTrackers which processed Map tasks to its local file system. Second, the TaskTracker sorts intermediate results of key/value pairs. Third, the TaskTracker merges the results according to users' request. Finally, the JobTracker returns merged results to users. The output of MapReduce is reduced and simplified key/value pairs of the input, because they are sorted and merged according to user programs. The entire process is conducted in parallel. Figure 2 briefly describes the MapReduce workflow.

Basically, one JobTracker is assigned to Namenode of HDFS, whereas several TaskTrackers are located in each of Datanodes of HDFS. JobTracker controls all TaskTrackers and supervises status of all TaskTrackers in the system. TaskTrackers update their status by sending HeartBeat in regular basis. HeartBeat message contains status of its sending TaskTracker, information of its executing task, and request for a new task. The status of TaskTracker includes information of its CPU, Memory, ID, Health, Maximum number of Map tasks and Maximum number of Reduce tasks. The information of executing task includes status of task, TaskID, start time of the task, finish time of the task, and progress rate of the task.

Briefly In the fact, MapReduce Scheduling system takes on in six steps [2]: first, User program divides the MapReduce job. Second, master node distributes Map Tasks and Reduce Tasks to different workers. Third, Map Tasks reads in the data splits, and runs map function on the data which is read in. fourth, Map Tasks write intermediate results into local disk. Then, Reduce Tasks read the intermediate results remotely, and run reduce function on the intermediate results which are read in. finally, These Reduce Tasks write the final results into the output files.

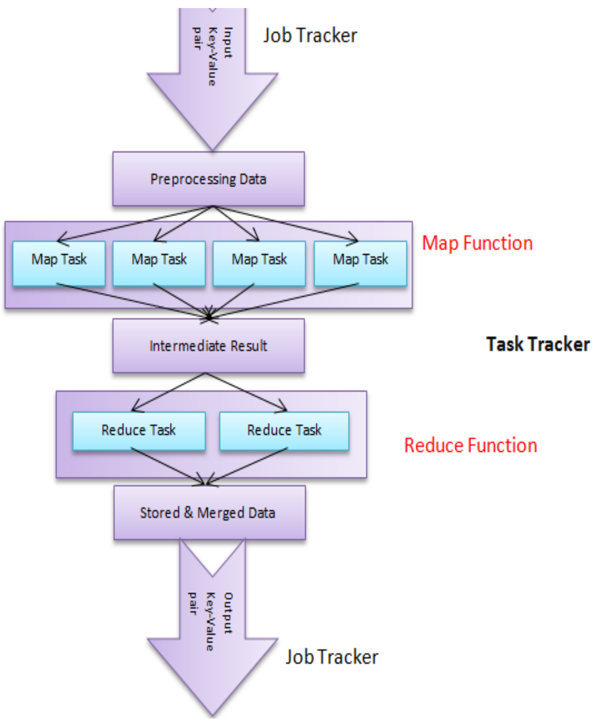

Figure 2. MapReduce Workflow.

III. MAPREDUCE SCHEDULING ALGORITHMS

The Scheduling is one of the most critical aspects of MapReduce. There are many algorithms to address these issues with different techniques and approaches.

A. *FIFO Scheduling Algorithm*

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 [17]. Hadoop's scheduler exploits FIFO policy. FIFO scheduler have many limitations such as The drawback of FIFO scheduling is poor response times for short jobs in the presence of large jobs and Low performance when run multiple types of jobs and it give good result only for single type of job. To address these problems scheduling algorithms such as Fair and Capacity was introducing.

B. *Fair Scheduling Algorithm*

Fair scheduling is a method of assigning resources to jobs such that all jobs get, on average, an equal share of resources over time [18]. When there is a single job running, that job uses the entire cluster. When other jobs are submitted, tasks slots that free up are assigned to the new jobs, so that each job gets roughly the same amount of CPU time. Unlike the default Hadoop scheduler, which forms a queue of jobs, this lets short jobs finish in reasonable time while not starving long jobs [19]. It is also an easy way to share a cluster between multiple of users. Fair sharing can also work with job priorities, the priorities are used as weights to determine the fraction of total compute time that each job gets. An important characteristic for a scheduler that manages tasks on this kind of cluster is fair sharing, meaning that each job needs to be allocated the same amount of resources [27]. For example, if two jobs are running, they should each receive half of the resources. To achieve fair sharing, a scheduler has to reallocate resources between jobs every time a new job starts. This reallocation can be done in two ways:

- By killing running tasks to make room for a new job. Using this method, the resources needed for the new job are reallocated instantly, but the downside is that work is lost by killing jobs.
- By waiting until running tasks finish. This method does not waste any work, but it can negatively impact fairness if tasks take a long time to finish.

Facebook built the Fair Scheduler, which allocates resources evenly between multiple jobs and also supports capacity guarantees for production jobs. The objective of Fair scheduling algorithm is to do equal distribution of compute resources among the users/jobs in the system [20]. The scheduler actually organizes jobs by resource pool, and shares resources fairly between these pools. By default, there is a separate pool for each user. The Fair Scheduler can limit the number of concurrent running jobs per user and per pool. Also, it can limit the number of concurrent running tasks per pool. The traditional algorithms have high data transfer and the execution time of jobs. Tao et al. [21] introduced an improved FAIR scheduling algorithm, which takes into account job characteristics and data locality, which decreases both data transfer and the execution time of jobs. Thus, Fair scheduling can covers some limitation of FIFO such as: it can works well in both small and large clusters and less complex. Fair scheduling algorithm does not consider the job weight of each node, which this is an important disadvantage of it.

C. *Capacity Scheduler*

The Capacity Scheduler from Yahoo offers similar functionality to the Fair Scheduler but takes a somewhat different philosophy. In the Capacity Scheduler, you define a number of named queues. Each queue has a configurable number of map and reduce slots. The scheduler gives each queue its capacity when it contains jobs, and shares any unused capacity between the queues. However, within each queue, FIFO scheduling with priorities is used, except for one aspect you can place a limit on percent of running tasks per user, so that users share a cluster equally [17],[22]. In other words, the capacity scheduler tries to simulate a separate FIFO/priority cluster for each user and each organization, rather than performing fair sharing between all jobs. The Capacity Scheduler also supports configuring a wait time on each queue after which it is allowed to preempt other queues' tasks if it is below its fair share.

D. Hybrid Scheduler Based on Dynamic Priority

In [25], the authors propose a case study in scheduling multiple workflows to the Cloud Computing paradigm by verifying that only the task that has a dependency value within the range specified. Nguyen et al. [23] propose a Hybrid Scheduler algorithm based on dynamic priority (HybS) in order to reduce the delay for variable length concurrent jobs, and relax the order of jobs to maintain data locality. The dynamic priorities can accommodate multiple task lengths, job sizes, and job waiting times by applying a greedy fractional knapsack algorithm for job task processor assignment. Also its provides a user-defined service level value for QoS. This algorithm is designed for data intensive workloads and tries to maintain data locality during job execution [24]. Their believes, average response time for the workloads approximately 2.1x faster over the Hadoop Fairs with a standard deviation of 1.4x. it achieves this improved response time by means of relaxing the strict proportional fairness with a simple exponential policy model. This algorithm is a fast and flexible scheduler that improves response time for multi-user Hadoop environments.

E. Longest Approximate Time to End (LATE)

The LATE scheduler was proposed by several engineers from the Berkley University of California in the article named "Improving MapReduce Performance in Heterogeneous Environments". LATE [26] algorithm improves the execution in Hadoop by finding real slow tasks. Although the LATE scheduler is proposed to resolve some problems that occur in heterogeneous environments, but this scheduler is designed to minimize the response time of first job in the job queue, and will prolong the response time of the other jobs in the job queue. LATE scheduler uses the past information to estimate the time to finish of tasks and is not suitable for environments with dynamic loading. For this it computes the remaining time of all the tasks and selects a set of tasks with longer remaining time when compared to all the nodes and considers them as real slow tasks. In this algorithm it does not depend on the data locality property for launching a speculative map task.

In this algorithm the task which is speculatively executed is one which will finish farthest into the future because this task provides the greatest opportunity for a speculative copy to overtake the original and reduce the job's response time [19].If nodes run at consistent speeds and if there is no cost to launch a speculative task on an otherwise idle node, this policy is optimal.

Usually speculative tasks are launched only on fast nodes and not on stragglers in order to beat the original task with the speculative task. For this a heuristic is followed, i.e., don't launch speculative tasks on nodes that are below some threshold (SlowNode Threshold) of total work performed. Another method is to allow more than one speculative copy of each task, but it is just the wastage of resources. In order to handle the fact that speculative tasks cost resources, the algorithm is implemented with two heuristics:

- A SpeculativeCap which is a limit on the number of speculative tasks that can be running at once.
- A SlowTask Threshold that a task's progress rate is compared with, to determine whether it is "slow enough" to be speculated upon. This is done to prevent needless speculation when only fast tasks are running.

Briefly the LATE algorithm works as follows:

If a node asks for a new task and there are less speculative tasks running than the SpeculativeCap:

- When the node's total progress is below SlowNode Threshold defined, the request is ignored.
- Running tasks that are not speculated by its time to finish are ranked currently.
- A copy of the highest-ranked task with progress rate below SlowTask Threshold is launched.

F. Self-Adaptive MapReduce (SAMR)

Quan Chen et al. [13] proposed SAMR scheduling algorithm, which calculates progress of tasks dynamically and adapts to the continuously varying environment automatically. SAMR is inspired by LATE [28] scheduling algorithm. This scheduling algorithm holds historical information on each node. The TaskTracker reads the historical information and sets the parameters using these informations. Quan Chen et al. proposed several equations for finding slow tasks and slow TaskTrackers. The scheduler can launch backup tasks for slow tasks according to these equations. SAMR launches backup tasks for map tasks on the rapid nodes or on the slow reduce nodes, and launches the backup tasks for reduce tasks on the rapid nodes or on the slow map nodes. When a task or several tasks execute on TaskTracker, execution informations feedbacks to TaskTracker, and historical informations update using them. SAMR computes progress score of tasks more accurate than LATE, thus this scheduler launches backup tasks for really slow tasks that prolong job execution time. Self-Adaptive MapReduce scheduling algorithm (SAMR) uses historical information to adjust stage weights of map and reduce tasks when estimating task execution times. However, SAMR does not consider the fact that for different types of jobs their map and reduce stage weights may be different. Even for the same type of jobs, different datasets may lead to different weights.

G. An Enhanced Self-Adaptive MapReduce Scheduling Algorithm (ESAMR)

Xiaoyn Sun et al. [29] proposed the ESAMR algorithm to overcome these problems. ESAMR classifies the historical information stored on every node into k clusters using a machine learning technique. If a running job has completed some map tasks on a node, ESAMR records the job's temporary map phase weight (i.e., M1) on the node according to the job's map tasks completed on the node. The temporary M1 weight is used to find the cluster whose M1 weight is the closest. ESAMR then uses the cluster's stage weights to estimate the job's map tasks' TimeToEnd on the node and identify slow tasks that need to be re-executed. If a running job has not completed any map task on a node, the average of all k clusters' stage weights are used for the job. In the reduce stage, ESAMR carries out a similar procedure. After a job has finished, ESAMR calculates the job's stage weights on every node and saves these new weighs as a part of the historical information. Finally, ESAMR applies k-means, a machine learning algorithm, to re-classify the historical information stored on every worker node into k clusters and saves the updated average stage weights for each of the k clusters. By utilizing more accurate stage weights to estimate the TimeToEnd of running tasks, ESAMR can identify slow tasks more accurately than SAMR, LATE, and Hadoop default scheduling algorithms.

H. Delay Scheduling

Zaharia et al. [16] proposed in the article "Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling" a simple algorithm which named delay scheduling To address the conflict between locality and fairness. In delay scheduling when a node requests a task, if the head of line job cannot launch a local task, we skip it and look at subsequent jobs. However, if a job has been skipped long enough, we start allowing it to launch non- local tasks, to avoid starvation. Delay scheduling is a solution that temporarily relaxes fairness to improve locality by asking jobs to wait for a scheduling opportunity on a node with local data. When a node requests a task, if the head of line job cannot launch a local task, it is skipped and looked at subsequent jobs. However, if a job has been skipped long enough, non-local tasks are allowed to launch to avoid starvation [30]. Also, with delay scheduling added to the fair scheduler, the overall performance improved.

I. Context-Aware Scheduler

Kumar et al. [31] propose a context-aware scheduler (CASH). The proposed algorithm uses the existing heterogeneity of most clusters and the workload mix, proposing optimizations for jobs using the same dataset. This scheduler increases the performance in heterogeneous Hadoop clusters. Although still in a simulation stage, this approach seeks performance gains by using the best of each node on the cluster. The design is based on two key insights. First, a large percentage of MapReduce jobs are run periodically and roughly have the same characteristics regarding CPU, network, and disk requirements. Second, the nodes in a Hadoop cluster become heterogeneous over time due to failures, when newer nodes replace old ones. The proposed scheduler is designed to tackle this, taking into account job characteristics and the available resources within cluster nodes. The scheduler uses then three steps to accomplish its objective: classify jobs as CPU or I/O bound; This scheduler classifies the nodes as Computational or I/O good; and map the tasks of a job with different demands to the nodes that can fulfill the demands. Thus, by implementing CASH the performance of the heterogeneous cluster and the aggregate execution times of the jobs can be improved.

J. Deadline Constraint Scheduler

This scheduling solution was proposed in the article "Scheduling Hadoop Jobs to Meet Deadlines" [32]. The main problem that this scheduling algorithm tries to address is scheduling jobs based on deadline constraints specified by the user. To do this, it first proposes a job execution cost model that accounts for the different parameters that affect the job completion time of Hadoop and after that it presents the design of a constraint based Hadoop scheduler that takes a deadline as input data and determines if a job can be scheduled using the model that was proposed. Jobs are only scheduled if specified deadlines can be met. To verify if a deadline can be met, a schedulability test must be run. The condition for a job to be scheduable is that the minimum number of tasks required for a job's schedulability independent of task assignment approach for both map and reduce is less than or equal to the available slots.

The main goals for designing the Deadline Constraint Scheduler were:

- To be able to give feedback to the users whether the job can be completed in the given deadline. If the job can meet the deadline then it will be executed, else the job is rejected and in this case can try to resubmit it with a modified deadline requirement.
- To maximize the number of jobs that can be run in the cluster while satisfying the deadlines for all the jobs.

In [33], the authors propose an extensional MapReduce Task Scheduling algorithm for Deadline constraints in Hadoop platform: MTSD. It allows user specify a jobs deadline and tries to make the job be finished before the deadline.

K. Matchmaking Scheduling Algorithm

In [34], the authors develop a new MapReduce scheduling technique to enhance map task's data locality. They had integrated this technique into Hadoop default FIFO scheduler and Hadoop fair scheduler. The main idea behind this algorithm is to give all nodes a fair chance to grab a local task before any non-local tasks are assigned to any nodes. The Matchmaking algorithm also relaxed the job order for task assignments meaning, that if a local task cannot be found for a specific node, then the scheduler moves on to the next job to try and find a local task. If no local tasks can be found for a node, then the node will not receive any task for this heartbeat interval. During one heartbeat interval all the other nodes that have free map slots will likely have sent their own heartbeats to the master node, and possibly received local task if it's the case. If the second time in a row a local task can't be found for a node, only then will a non-local task be assigned, so that we don't waste computing resources, but, during a heartbeat interval, the algorithm allows a node to take at most one non-local task. To enforce this rule, the nodes use a locality marker that marks their status. If a local task cannot be assigned to a node, then, depending on the value of the node's locality marker, the node will either wait for a heartbeat interval or receive a non-local task. In the case that a new job is added to the queue, all the locality markers are reset, because the new job might have tasks that are local to the nodes. The matchmaking algorithm is applicable to any scheduling policy that defines the order in which jobs are given resources.

L. SLO-Driven

A key challenge in MapReduce environments is the ability to efficiently control resource allocation and task scheduling for achieving Service Level Objectives (SLOs) of MapReduce jobs. However, there are few effective task scheduling methods to guarantee MapReduce jobs' SLOs. Therefore, In [35], the authors address this challenge by proposing a SLO-driven task scheduling mechanism. Based on the MapReduce performance model they build, this mechanism dynamically adjusts resource allocation and task scheduling in order to guarantee the SLOs of jobs and improve global job utility.

However, there is currently a lack of efficient task schedulers in MapReduce environments to meet jobs' SLOs. In this algorithm, the authors propose a SLO-driven task scheduling mechanism to address the problem of how to effectively perform task scheduling in order to guarantee the SLO requirements and improve job utility. First, they present a MapReduce job performance model to build the relationship between job performance and resources the job holds. Second, based on the job performance model, they design a SLO-based resource estimator which estimates minimum resources required for a job to meet its SLO and a utility estimator which estimates job utility under different resource allocation strategies. Finally, they propose a SLO-driven task scheduler which implements task scheduling in order to guarantee jobs' SLOs and enhance utility.

In MapReduce environments, meeting SLOs of jobs is an important requirement. However, there are few efficient task schedulers to guarantee SLOs in MapReduce environments. In this algorithm, the authors addressed this problem and proposed a MapReduce job performance model and a SLO-driven task scheduling mechanism. they job performance model maps the slot allocation for a job to its expected completion time using historical profile of the job. The SLO-based resource estimator derives minimum slots for performing task scheduling with SLOs and the utility estimator estimate marginal utility for each slot allocation under consideration. This mechanism proved to be effective in achieving SLO, as well as improve the global job utility.

M. Throughput Driven

In [36], the authors proposes a novel technique for job-intensive scheduling to improve the system throughput. The authors propose a novel technique, called Throughput Driven task (TD) scheduler, for the job-intensive MapReduce environment. The scheduler mostly focuses on the map phase, which dominates the computational cost of the most MapReduce applications.

In the job-intensive environment, Briefly following 4 major factors which can impact the system throughput. the authors design the TD scheduler to satisfy the requirements of the 4 factors.

- High ratio of the local processing. Nonlocal tasks would lead more network transmission and more execution delay. In the distributed environment, it is necessary to maintain high ratio of the local processing.
- Choosing a befitting nonlocal task. For the high parallelism, nonlocal execution cannot be avoided, and choosing a better nonlocal task would be beneficial for the throughput.
- Avoiding hotspots. While processing in parallel, data stored on each node may be unbalanced in a certain period of time, there will be some nodes from which many nonlocal tasks read data, and calls the nodes as hotspots. The hotspot can distinctly impact system thoughput because a number of nodes read data from it so that the transmission speeds become lower, and the performance of hotspot itself decreases.
- Full use of system resources. for energy saving.

N. Real-Time Scheduling

Many MapReduce applications require real-time data processing, scheduling real-time applications in MapReduce environments has become a significant problem. Polo et al. [37] developed a soft real-time scheduler that allows performance-driven management of MapReduce jobs. Dong et al. [38] extended the work by Polo et al. where a two-level MapReduce scheduler was developed to schedule mixed soft real-time and non-real-time jobs according to their respective performance demands. In [39], the authors create a novel real-time scheduler for MapReduce, which overcomes the deficiencies of an existing scheduler. It avoids accepting jobs that will lead to deadline misses and improves the cluster utilization and ensuring the real-time property for all admitted MapReduce jobs. RTMR scheduler not only provides deadline guarantees to accepted jobs but also well utilizes system resources. RTMR scheduler includes:

- The input data is available in (HDFS) before a job starts.
- No preemption is allowed. The proposed scheduler orders the job queue according to job deadlines.
- A MapReduce job contains two stages: map and reduce stages. A MapReduce job contains two stages: map and reduce stages.

RTMR scheduler is composed of three components. The first and most important one is the admission controller, which makes decisions on whether to accept or reject a job. The admission controller makes decisions based on information maintained in job records. The second component is the job dispatcher, which assigns tasks to execute on worker nodes. The last component is the feedback controller. Since a job may finish at a different time than estimated, a feedback controller is designed to keep the admission controller up-to-date.

In a heterogeneous environment, worker nodes have different data retrieving and processing power. In order to avoid deadline miss, they follow the same mechanism as adopted by the Deadline Constraint scheduler [32] where the longest time of running a map/reduce task is used in the execution time estimation.

O. Locality-Aware Reduce Task Scheduler

In [40], the authors proposed in the article "Locality-Aware Reduce Task Scheduling for MapReduce" another approach discussing the data locality problem. It deals specifically with Reduce tasks, they propose the Locality-aware reduce task scheduler (LARTS) which is supposed to increase Hadoop performance by taking into account locality when scheduling reduce tasks too so that the network traffic generated by moving data to the node that needs it decreases. The scheduler, named Locality-Aware Reduce Task Scheduler (LARTS), uses a practical strategy that leverages network locations and sizes of partitions to exploit data locality. LARTS tries to obtain data locality for reduce tasks. It does this by scheduling a reduce task on the maximum-node of maximum-rack. Each reduce task has several partitions of data that it needs for the reducing process. All these partitions are spread on different nodes situated on different racks throughout the cluster. LARTS attempts to schedule Reducers as close as possible to their maximum amount of input data and conservatively switches to a relaxation strategy seeking a balance among scheduling delay, scheduling skew, system utilization, and parallelism.

IV. ANALYSIS ADVANTAGE AND DISADVANTAGE

Advantages and disadvantages of MapReduce scheduling methods are expressed in Tables I. In a heterogeneous environment where each node has different computing power the heuristic method is not well suited. To improve the response time of Hadoop in heterogeneous environments Longest Approximate Time to End (LATE) scheduling is devised. In the disadvantage column, some of these algorithms have null value. Because, they can achieved to their proposed and due to result of many articles we believe they don't have any disadvantage that able to reduce their abilities and performances. All of these algorithms proposed to have some advantages and disadvantages.

TABLE I. COMPARISON OF DIFFERENT ALGORITHMS

Algorithm	Advantages	Disadvantages
FIFO	• Cost of entire cluster scheduling process is less. • Implementation is easy.	• poor response times for short jobs in the presence of large jobs. • designed only for single type of job. • Low performance when run multiple types of jobs.
Fair	• Is not complex. • It can provide fast response times for small jobs mixed with larger jobs.	• Never consider the job weight of each node.
Capacity	• Ensure guaranteed access with the potential to reuse unused capacity and prioritize jobs within queues over large cluster.	• The most complex than FIFO and Fair schedulers.
Hybrid scheduler based on dynamic priority	• Good performance because fast and flexible scheduler. • Improves response time for multi-user Hadoop environments.	-
LATE	• Robust to node heterogeneity. • Takes into account node heterogeneity when deciding where to run speculative tasks. • Speculatively executes only tasks that will improve job response time, rather than any slow tasks.	• Only takes action on appropriate slow tasks. • However it does not compute the remaining time for tasks correctly and cannot find real slow tasks in the end. • Poor performance due to the static manner in computing the progress of the tasks.
SAMR	• Reduce runtime • Save system resources • Scalability	• It does not consider that the dataset sizes and the job types can also affect the stage weights of map and reduce tasks. • Don't consider the data locality management for launching backup tasks.
ESAMR	• Reduce runtime • Save system resources • Scalability	• Little overhead due to K-means algorithm. • Allows only one speculative copy of a task to run on a node at a time. • Ignore data locality for launching backup tasks.
Delay scheduling	• Simplicity of scheduling	-
Context-aware	• Optimizations for jobs using the same dataset. • Performance of the heterogeneous cluster and the aggregate execution times of the jobs can be improved.	-
Deadline Constraint Scheduler	• Maximize the number of jobs that can be run in the cluster.	• Doesn't try to achieve maximum performance. • Low data locality rate.
Matchmaking Scheduling	• Increase data locality for map tasks. • Near-optimal average response times.	-
SLO-Driven	• Enhances job utility guarantee the SLOs of jobs.	-
Throughput Driven	• Improve the throughput of a MapReduce cluster system. • Excellent performance if the job queue contains a high percentage of small jobs.	-
Real-Time Scheduling	• Achieves good cluster utilization. • Better performance than Deadline Constraint Scheduler.	-
Locality-Aware Reduce Task Scheduler	• Improve scheduling delay, scheduling skew, system utilization, and parallelism. • Reduce network traffic. • Increase of performance.	• Static sweet spot determination (Sweet spot of a program is the spot at which early shuffle is triggered and provides the best performance for the program).

V. CONCLUSION

In this paper we attempted to explain and analyzed fifteen different MapReduce scheduling algorithms. Besides completion time, fairness is another important criterion for scheduling tasks. Longest Approximate Time to End (LATE) scheduling can improved response time of Hadoop in heterogeneous environments. So, to improve the overall MapReduce performance in the heterogeneous environments we can use of SAMR algorithms. ESAMR can identify slow tasks more accurately than SAMR. Also, to improvement data Locality LARTS is the best case. If you need a fast and flexible scheduler, Hybrid scheduler based on dynamic priority is better than anyone. For achieves good cluster utilization we can use of Real-Time Algorithm.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph , "A View of Cloud Computing ", Comm. Of the ACM, Vol. 53, No. 4, pp. 50-58, April 2010.

[2] J. Dean and S. Ghemawat, “MapReduce: Simplied Data Processing on Large Clusters”, In Proc. of 5th Symposium on Operating Systems Design and Implementation, 2008, pp. 137-150.

[3] S. Ghemawat, H. Gobioff, and S. T. Leung, "The Google File System", In ACM Symposium on Operating Systems Principles (SOSP), 2003.

[4] W. Jiang, T. Ravi and G. Agrawal, "Comparing MapReduce and Freeride For Data-Intensive Applications", In Proc. Of Cluster Computing and Workshops, 2009, pp. 1-10.

[5] C. T. Chu, S. K. Kim, Y. A. Lin, Y. Y. Yu, G. Bradski, A. Y. Ng, K. Olukotun, Map-reduce for machine learning on multicore, http://www.cs.standford.edu/peop;e/ang//papers/nips06-mapreducemulticoure.pdf(2006), Accessed 1 March 2012.

[6] J. Ekanayake, S. Pallickara, G. Fox, "MapReduce for data intensive scientific analyses", Proceedings of the IEEE Fourth International Conference on eScience, 2008.

[7] G. Mackey, S. Sehrish, J. Bent, J. Lopez, S. Habib, J. Wang, "Introducing map-reduce to high end computing", Proceedings of the 3rd Patascale Data Storage Workshop, 2008.

[8] M. Isard, M. Budiu, Y. Yu, A. Birrell, D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks", Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems, 2007.

[9] H. Han, H. Jung, H. Eom, H. Y. Yeom, "Scatter-Gather-Merge:an efficient star-join query processing algorithm for data-parallel frameworks", Cluster Computing, 14(2), 2010.

[10] B. He, Q. Luo, N. K. Govindaraju, "Mars: accelerating MapReduce with graphics processors", IEEE Trans. Parallel Distribute System, 22(4), 2011.

[11] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, C. Kozyrakis, "Evaluating mapreduce for multi-core and multiprocessor systems", IEEE 13th International Symposium on High Performance Computer Architecture, 2007.

[12] The Apache Software Foundation: Hadoop (2012).http://hadoop.apache.org. Accessed 1 March 2012.

[13] Q. Chen, D. Zhang, M. Guo, Q. Deng, and S. Guo, “ SAMR: a self-adaptive MapReduce scheduling algorithm in heterogeneous environment ”, In Proc. of the 10th IEEE International Conference on Computer and information Technology, 2010, pp. 2736-2743.

[14] O. H. Ibarra and C. E. Kim. "Heuristic algorithms for scheduling independent tasks on nonidentical processors", Journal of the ACM, 24(2), 1977, pp. 280–289.

[15] C. Tian, H. Zhou, Y. He and L. Zha, “A Dynamic MapReduce Scheduler for Heterogeneous Workloads”, In Proc. of the Eighth International Conference on Grid and Cooperative Computing, 2009, pp. 218-224.

[16] M. Zaharia, D. Borthakur, J.S. Sarma, K. Elmeleegy, S. Shenker and I. Stoica, “ Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling”, In: Proceedings of the fifth European conference on computer systems, New York, NY, USA: ACM, 2010, pp. 265–278.

[17] Hadoop, “Hadoop home page.” http://hadoop.apache.org/.

[18] Hadoop’s Fair Scheduler. https://hadoop.apache.org/docs/r1.2.1/fair_scheduler.

[19] B. P. Andrews and A. Binu, “ Survey on Job Schedulers in Hadoop Cluster ”, IOSR Journal of Computer Engineering, Vol.15, NO. 1, Sep - Oct. 2013, pp. 46-50.

[20] The Apache Hadoop Project. http://www.hadoop.org.

[21] Y. Tao Y, Q. Zhang, L. Shi and P. Chen, “ Job scheduling optimization for multi-user MapReduce clusters ”, In: The fourth international symposium on parallel architectures algorithms and programming, IEEE, 2011, pp. 213–217.

[22] J. Chen, D. Wang and W. Zhao, “ A Task Scheduling Algorithm for Hadoop Platform ”, JOURNAL OF COMPUTERS, VOL. 8, NO. 4, APRIL 2013, pp. 929-936.

[23] P. Nguyen, T. Simon, M. Halem, D. Chapman and Q. Le, “A hybrid scheduling algorithm for data intensive workloads in aMapReduce environment”, In: Proceedings of the 2012 IEEE/ ACM fifth international conference on utility and cloud computing. Washington, DC, USA: IEEE computer society; UCC'12, 2012, pp. 161-168.

[24] I. Polato “A comprehensive view of Hadoop research—A systematic literature review ”, Journal of Network and Computer Applications, 2014, http://dx.doi.org/10.1016/j.jnca.2014.07.022.

[25] B. A. Kumar and T. Ravichandran, "Instance and value (IVH) algorithm and dodging dependency for scheduling multiple instances in hybrid cloud computing", Pattern Recognition, Informatics and Mobile Engineering (PRIME), International Conference, 2013.

[26] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz and I. Stoica, “Improving MapReduce performance in heterogeneous environments ”, In: OSDI 2008: 8th USENIX Symposium on Operating Systems Design and Implementation, 2008.

[27] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, I. Stoica, "Job scheduling for multi-user MapReduce clusters", http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-55.pdf.

[28] A. Konwinski, "Improving mapreduce performance in heterogeneous environments", Technical Report No. UCB/EECS-2009-183, University of California, Berkeley, 2009.

[29] X. Sun, C. He and Y. Lu, “ESAMR: An Enhanced Self-Adaptive MapReduce Scheduling Algorithm”, IEEE 18th International Conference on Parallel and Distributed Systems, 2012.

[30] A. P. Kulkarni and M. Khandewal, “ Survey on Hadoop and Introduction to YARN ”, International Journal of Emerging Technology and Advanced Engineering, Vol.4, NO. 5, May 2014, pp. 82-87.

[31] K. A. Kumar, V. K. Konishetty, K. Voruganti and G. Rao, “ CASH: context aware scheduler for Hadoop”, In: Proceedings of the international conference on advances in computing, communications and informatics, New York, NY, USA: ACM, 2012, pp. 52–61.

[32] K. Kc and K. Anyanwu, “Scheduling hadoop jobs to meet deadlines", In 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom), 2010, pp. 388 –392.

[33] Z. Tang. J. Zhou, K. Li and R. Li, "A MapReduce task scheduling algorithm for deadline constraints", Cluster Computing, Vol. 16, 2013.

[34] C. He, Y. Lu, D. Swanson, "Matchmaking: A New MapReduce Scheduling Technique".

[35] J. Wang, Q. Li, Y. Shi, "SLO-Driven Task Scheduling in MapReduce Environments ", 10th Web Information System and Application Conference, 2013, pp. 308-313.

[36] X. Wang, D. Shen, G. Yu, T. Nie, Y. Kou, "A Throughput Driven Task Scheduler for Improving MapReduce Performance in Job-intensive Environments ", IEEE International Congress on Big Data, 2013, pp. 211-218.

[37] J. Polo, D. Carrera, Y. Becerra, M. Steinder, and I. Whalley, "Performance-driven task co-scheduling for mapreduce environments", In Network Operations and Management Symposium (NOMS), IEEE, 2010, pp. 373 –380, 19-23.

[38] X. Dong, Y. Wang, H. Liao, "Scheduling Mixed Realtime and Non-real-time Applications in MapReduce Environment", In the proceeding of 17th International Conference on Parallel and Distributed Systems, 2011, pp. 9 – 16.

[39] C. He, Y. Lu and D. Swanson, " Real-Time Scheduling in MapReduce Clusters ", IEEE International Conference on High Performance Computing and Communications & IEEE International Conference on Embedded and Ubiquitous Computing, 2013, pp.1536-1544.

[40] M. Hammoud and M. Sakr, " Locality-aware reduce task scheduling for MapReduce", In: The third international conference on cloud computing technology and science, IEEE, 2011, pp. 570–576.