Modern Applied Science; Vol. 12, No. 8; 2018
ISSN 1913-1844 E-ISSN 1913-1852
Published by Canadian Center of Science and Education

An Overview of Hadoop Scheduler Algorithms

Faten Hamad[1]

[1] School of Educational sciences, The University of Jordan, Amman, Jordan

Correspondence: Faten Hamad, School of Educational sciences, The University of Jordan, Amman, Jordan. E-mail: f.hamad@ju.edu.jo

Received: January 27, 2018 Accepted: July 6, 2018 Online Published: July 25, 2018

doi:10.5539/mas.v12n8p69 URL: https://doi.org/10.5539/mas.v12n8p69

Abstract

Hadoop is a cloud computing open source system, used in large-scale data processing. It became the basic computing platforms for many internet companies. With Hadoop platform users can develop the cloud computing application and then submit the task to the platform. Hadoop has a strong fault tolerance, and can easily increase the number of cluster nodes, using linear expansion of the cluster size, so that clusters can process larger datasets. However Hadoop has some shortcomings, especially in the actual use of the process of exposure to the MapReduce scheduler, which calls for more researches on Hadoop scheduling algorithms.

This survey provides an overview of the default Hadoop scheduler algorithms and the problem they have. It also compare between five Hadoop framework scheduling algorithms in term of the default scheduler algorithm to be enhanced, the proposed scheduler algorithm, type of cluster applied either heterogeneous or homogeneous, methodology, and clusters classification based on performance evaluation. Finally, a new algorithm based on capacity scheduling and use of perspective resource utilization to enhance Hadoop scheduling is proposed.

Keywords: Hadoop scheduling, capacity scheduler, fair scheduler, FIFO scheduler

1. Introduction

The past decade has witnessed a rapid development of cluster computing platforms, due to the increased data sizes, known as big data, which require more scalable applications. Big data, cannot be handled using traditional database and software technologies due to its size, speed, and variety. For example, Google reported to process more than 20G of data per day, and Facebook reported that it handles between 15-20 terabytes of compressed data each day. This amount of data certainly cannot be handled by a single computer. It is also not feasible nor cost effective to handle big data with a single super high performance PC. Therefore, many models are designed to efficiently handle big data in parallel with business computer sets. For instance, the open-source Apache Hadoop has emerged as a de facto platform to handle large-scale, semi-structured and unstructured data (Brahmwar et al., 2016) (Al-Sayyed et al., 2017).

Hadoop is a reliable and scalable tool for distributed computing, data storage and processing. It is an open source program for writing and implementing applications in a group. Hadoop and Hadoop are sources accessible to Mapreduce and Google's file system. Hadoop was originally implemented in Java (Hamad, & Alawamrah, 2018). Applications in Hadoop can be written in different programming languages. When programmers write applications using the map and reduce functionality, the Hadoop framework automatically performs these functions in parallel. Hadoop was first created by Doug Kuting in 2004 and was developed primarily by Yahoo. Its streaming utility allows the user to create and run functions using any executable map and reduction functions. Apache Hadoop System is a Mapreduce project that was developed in Java by the Apache Foundation. As Hadoop is published under the Apache license, the Hadoop source code is available for public download. Hadoop is deployed in Yahoo servers, where hundreds of terabytes of data are created on at least 10,000 cores. Facebook makes use of Hadoop packages to handle more than 20 terabytes of new data per day. Other web giants such as Amazon employ ad hoc groups to manage massive amounts of data on a daily basis (Guo et al, 2015) (Hudaib, et al., 2016).

The Hadoop system contains two basic components. The first component is a distributed file system called HEDFES; and the second is the Mapreduce programming framework for processing large data sets. The implementation of Hadoop MapRedus is designed for large groups, and targets distributed file system (i.e. HDFs), In most Hadoop functions, HDFs is used to store both inputs from the map and output tasks by

minimizing tasks (Zaharia et al, 2010).

Hadoop has two layers; on top, a Mapreduce engine that contains tracker function and tracker functions; and the bottom, HDFs, contains NAMnode and Datanodes. Each node can act as master or slave in relation to both Hodges and Mapreduce. The default Hadoop messages are shown in figure 1.

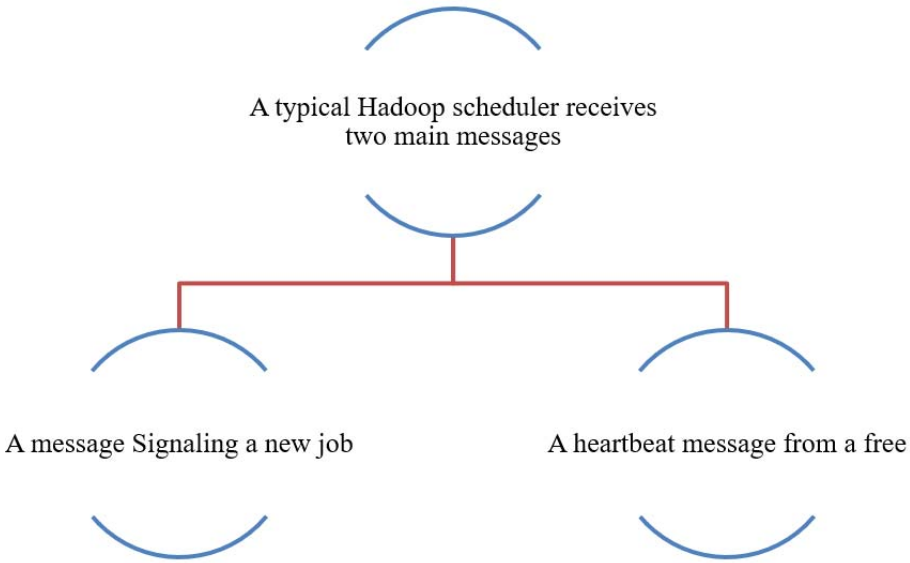


Figure 1. Hadoop typical scheduling massages

Hadoop has its own file system known as HDVs. HDF provides high productivity access to the application and is designed to store large files with the flow of the data access pattern and run on cluster devices. Monitor constantly monitors the server to ensure data availability. The HDVs are designed to predict failures in the commodity machinery groups in order to tolerate and compensate for all those failures. The main benefit of HDFs is that the software itself deals with hardware issues, freeing users of having to worry about system failure.

1.1 Characteristics of Hadoop

The main characteristics of Hadoop are:

Accessibility and Cost Effectiveness: it is the best approach for solving big data set problems isong a range of commodity machines. It has been used as a cloud computing service, for example. Amazon Web Services (OS) (Al-Sayyed et al., 2017).

Scalability: Hadoop is a highly scalable model that can be used as a classroom research tool in colleges, or as a terabyte of data storage and analysis as in Yahoo, Facebook or Amazon. If more capacity is needed, it can be added as required (Reddy, et al, 2011).

General Purpose: Hadoop is a powerful and easy tool for programming. Even new or non-programmers can write and execute parallel code in a simple way (White, 2012).

Low barrier entry: Hadoop does not need any schema or database in the front. All users are able to easily download and use raw data.

However, Hadoob has many issues and challenges associated with the scheduler research and management in cluster computing frameworks for large-scale data processing. For instance, Clauser's distinctive computing platforms, such as Haddob, were deliberately designed to improve liberal liberalism or a range of large functions. (Rasooli, & Down, 2011). Moreover, cluster computing platforms serve diverse workloads of deferred and deferred sources. These workloads usually have important considerations in the underlying performance. For example, interactive applications that require good response times while making a period or deadlines are more important for periodic batch jobs. There is no single resource management schema or scheduling applications that are optimized for all performance metrics (Bhosale & Gadekar, 2014). The original Vivo scheduling policy is designed by Hadoop Mapreduce to make a better payment. However, the response time of short functions is sacrificed when applications are introduced behind long intervals. Scheduling such as fair and resource sharing

capabilities is designed between users and applications that support equity and provide better performance for short applications. However, they are not optimal in terms of response times for work or productivity (Yong, Garegrat, & Mohan, 2009) (Hamad, & Adwan, 2018).

Dependency between tasks is another issue with Hadoop. While large functions are broken in small tasks for parallel performance, there are usually dependencies between functions in most cluster computing applications. In the Hadrid Mapredus platform, task reduction depends on the map functions of the same function since performing task reductions based on the average data produced by map tasks. The data transfer process is called in the Mapreduce mix. In the traditional definition of task dependency, when the task depends on others, the start time cannot be earlier than the completion of any of its subordinate tasks. However, in the Iron Mapreduce platform, the reduction of tasks actually begins earlier (He et al, 201). The reason is that the dribbling process is one with reducing tasks under the center, such that starting to reduce tasks earlier can help improve performance by interfering with Xu progress with the progress of the map, i.e. keeping the intermediate data produced by the map tasks done while other map tasks are still running or waiting. In other frameworks, there may be more complex dependencies between tasks in each job. Types of deferred tasks for cluster computing applications are typically resource-driven. For example, in the Mapreduce framework, each application has two main phases, the map and the minimization (DeWitt, & Stonebraker, 2008).

There can be multiple independent tasks that perform the same functions at each stage, i.e. mapping tasks and reducing tasks. These two types of tasks are often quite different resource requirements. The mapping tasks are typically intensive for the CPU while reducing intensive I/O tasks, especially when the middleware map designers are brought in. System resources can be efficiently used if maps and task reduction work concurrently on a worker's contract. To ensure the best use of resources, the first generation aims to distinguish the important tasks of the map/reduce tasks by configuring a different map/reducing slots on each node (Rao, & Reddy, 2012). The concept of time interval is to extract node capacity where each map/time interval accommodates at least one map/reduces task at any given time (Al Khattab, Aet al, 2015). By setting the number of map /reducing slots on each node, the Hadoop platform controls the synchronization of different types of tasks in the cluster to achieve better performance. The second-generation system of Hadoop-Yarn adopts the management of granular resources where each task needs to explicitly define its demands on different types of resources, i.e. CPU and memory. The resource manager therefore benefits from heterogeneous resource requirements and uses the resources of the group more accurately and efficiently (Kc and Anyanwu, 2010).

Moreover, many current resource management schemes cannot fully benefit from group resources. For example, a Twitter production group managed by Missus reported that the total CPU utilization is less than 20% and Google system reported a combined CPU usage of 25-35%. One of the main reasons for this is that existing resource management schemes always maintain a fixed amount of resources for each task as requested by their resources. However, note that the tasks of different data processing frameworks and applications can have different patterns to use resources. For example, many tasks of cluster computing applications consist of multiple internal phases and have relatively long execution times. These tasks usually have a variety of resource requirements during execution. As discussed above, minimizing tasks in the framework of MapRedus is usually less CPU usage at the shuffle stage, i.e. fetching intermediate data, when waiting for map tasks to generate outputs (Hudaib, & Fakhouri, 2016).

Another example is Spark's tasks. When deployed on the Yarn system, Spark's mission is to host multiple user-defined stages that also require different types and amounts of resources. Moreover, when Spark tasks serve an interactive function, the use of resources from these tasks can often change, for example, being completely idle during the user's thinking time, and become busy and requesting more resources. Similarly, frames that handle data flow may maintain a large number of functions alive and wait for input. Therefore, resource requirements must change over time when new incoming data arrive, which unfortunately cannot be predicted. Although short tasks dominate many cluster computing clusters, the long-term effects of long-term tasks on the use of system resources are negligible given their high resource demands and the long occupation of resources. In such cases, the allocation of resources during the lifetime of the task becomes ineffective to take full advantage of system resources (Chen et al., 2010).

The default scheduler assumes that the cluster environment is designed to be homogeneous so that all nodes in the cluster have the same computation and configuration capacity. However, the real world applications work in a heterogeneous environment. Accordingly, the overall performance of the Hadoop fails if it continues to use the default schedule policy in a heterogeneous environment. Moreover, some tasks take longer time to perform compared to other tasks on the same node. Such tasks are called stragglers. These are responsible for lengthening task execution time. To solve this issue, Hadoop Mapreduce run the backup task of the slow task on another node

that has a faster account in order to prevent slow tasks of lengthening the overall execution time of work.

2. Overview of Default Hadoop Scheduler Algorithms

2.1 FAIR Scheduler

Scheduling in Hadoop organizes functions in categories, including shared resources. Each pool is allocated a guaranteed minimum share, ensuring that certain users or applications always have adequate resources. Fair sharing also works with work priorities, which are used as weights to determine the part of the total time allocated to each post. The fair scheduling determines the resources for posts so that all posts, on average, consume an equal share of resources. For example, if queue1 gets 80% of clusters and queue2 gets 20% of clusters, then a high priority task goes to queue1 (Zaharia, 2009).

The Gallery Scheduler allows all functions to be run by a default or specified configuration file, limiting the number of jobs per user and per pool. This configuration file is very useful in two specific situations. First, the user tries to provide hundreds of jobs at once. Second, many functions simultaneously run cause high context switching overhead and a huge amount of intermediate data. Reducing the number of jobs running, of course, does not cause any later functionality provided for failure. However, you must wait for the newly arrived jobs in the Scheduler queue until some of the running functions are finished (Usha, & Jenil, 2014).

2.2 Capacity Scheduler

Capacity Scheduler is developed by Yahoo for a large collection of resource sharing. The functions are organized and placed in multiple queues, each of which is secured to reach a fraction of the mass capacity (i.e. number of task slots). All posts submitted to a particular queue guaranteed resources for this queue. If tasks of functions in queues have extra capacity it kills tasks. Free resources can be allocated to any queue beyond its capacity. When there is a demand for resources from queues operating under capacity at a later time (e.g. scheduled tasks on full resources), resources will be allocated to posts on queues operating under capacity. If inactive queues start to get work requests, their lost capacity will be rolled back.

Queues can also support priorities for posts that are disabled by default. In the queue, high priority jobs have access to queue resources before getting jobs with lower priority. However, once a function is run regardless of its priorities, task will not be pre-empted by any higher priority function. However, the new tasks of the highest priority function will be predefined in the queue. To prevent one or more users from monopolizing resources, each row assigns a percentage of the resources allocated to a user at any given time, if all users are using resources (Ghemawat, , Gobioff, & Leung, 2003). .

Whenever TaskTracker is a free, scheduling capabilities select queue with most free resources. Once the queue is selected, a scheduler selects a task in the queue according to the priority of the task. This scheduling mechanism ensures that there is enough free memory in TaskStaker to run the task in case the task has significant memory requirements. In this way, resource requirements can always be met immediately (Dean, & Ghemawat, 2010).

2.2.1 I Capacity Scheduler

Icapacity scheduling puts jobs into varied queues in accordance with the conditions, and allocates certain system capability for each queue. If a queue has serious load, it seeks unallocated resources, then makes redundant resources assigned equally to each job. It re-allocates the resources for empty queue to queues exploitation for maximizing resource. Once jobs arrive therein queue, running tasks square measure completed and resources square measure given back to main queue. It conjointly permits priority based totally programming of jobs in associate organization queue. To use icapacity dynamic scheduler, the subsequent property has to be set in yarnsite.xml like below: yarn. resourcemanager. schedular.class org.apache.hadoop.yarn.server.resourcemanager.sche dular.capacity.CapacitySchedular.

2.3 Hadoop Scheduling Algorithm Framework

Five scheduling algorithms have been selected in order compare between Hadoop scheduling algorithms framework in term of the proposed scheduler algorithm, the default scheduler algorithm that has been enhanced, the type of cluster applied; either heterogeneous or homogeneous, methodology, and clusters classification based on performance evaluation. See table 1. Five selected schedulers will be briefly discussed.

2.3.1 Tolhit – A Scheduling Algorithm for Hadoop Cluster (2016)

Tolhit uses the usage of resources and network information from cluster nodes to find the optimal node for scheduling a speculative version of a slow task. The performance of the proposed scheme has been evaluated through a series of experiments (Brahmwar, Kumar, & Sikka, 2016).

The name "Tolhit" comes from the algorithm ability to use more accurate phase weights to provide an estimation to the operation of tasks. Resources from cluster nodes such as Ram & Disk usage is also maintained and updated periodically in resource info. Parameter for Disk usage by the node is calculated by looking only at input and output requests from the running tasks on that node (Brahmwar, Kumar, & Sikka, 2016).

The proposed algorithm maintains history information of the date when each node is present in the cluster. Each record contains historical information about any map (M1 and M2) and reduces phase weights (R1, R2, and R3). Tolhit assumes the implementation of the map function Weight as M1 and then reorder the medium weight of the result as M2 of the map task. R1, R2, R3 stand for the copy, sort and reduce phase weights from task minimization. The date information stored on each node is sorted using genetic (13) to any of the clusters. If the ongoing work satisfies the extant (threshold for the task map Detection) on a data node, then the algorithm assigns a temporary task to the phase weight map (M1). The map phase weight is used as a search parameter to determine the block with the nearest map phase weight (M1) between (K) in historical information (Brahmwar, Kumar, & Sikka, 2016).

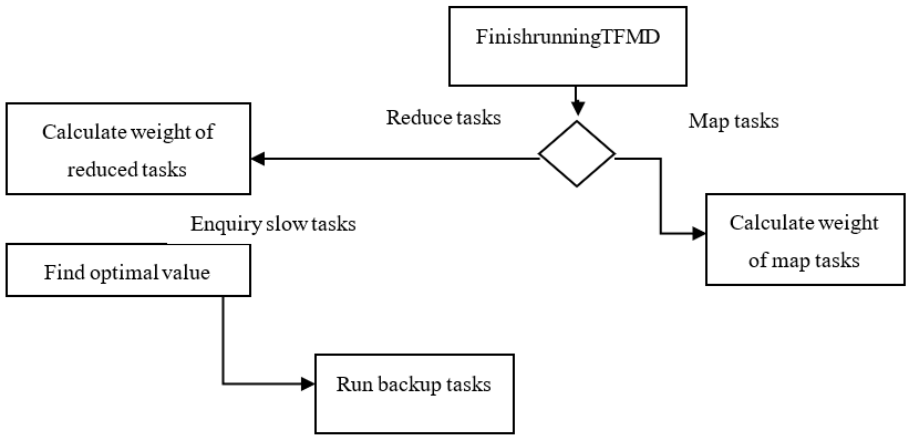


Figure 2. Tolhit algorithm flow chart

2.3.2 COSHH

The scheduling system, named COSHH, is considered heterogeneous for both application and mass levels. The goal of COSHH is to improve the average time of completion of employment. The high-level architecture of COSHH is displayed in the Figure.3 while COSHH Queuing Process are displayed in figure 4 (Rasooli, & Down, 2014).

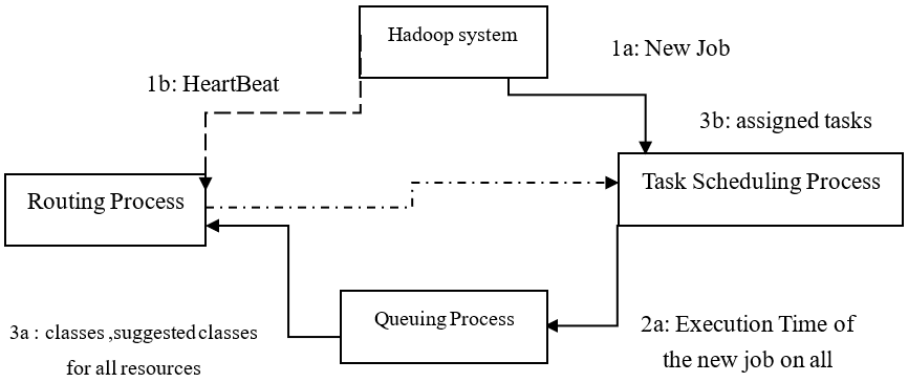


Figure 3. The high-level architecture of COSHH.

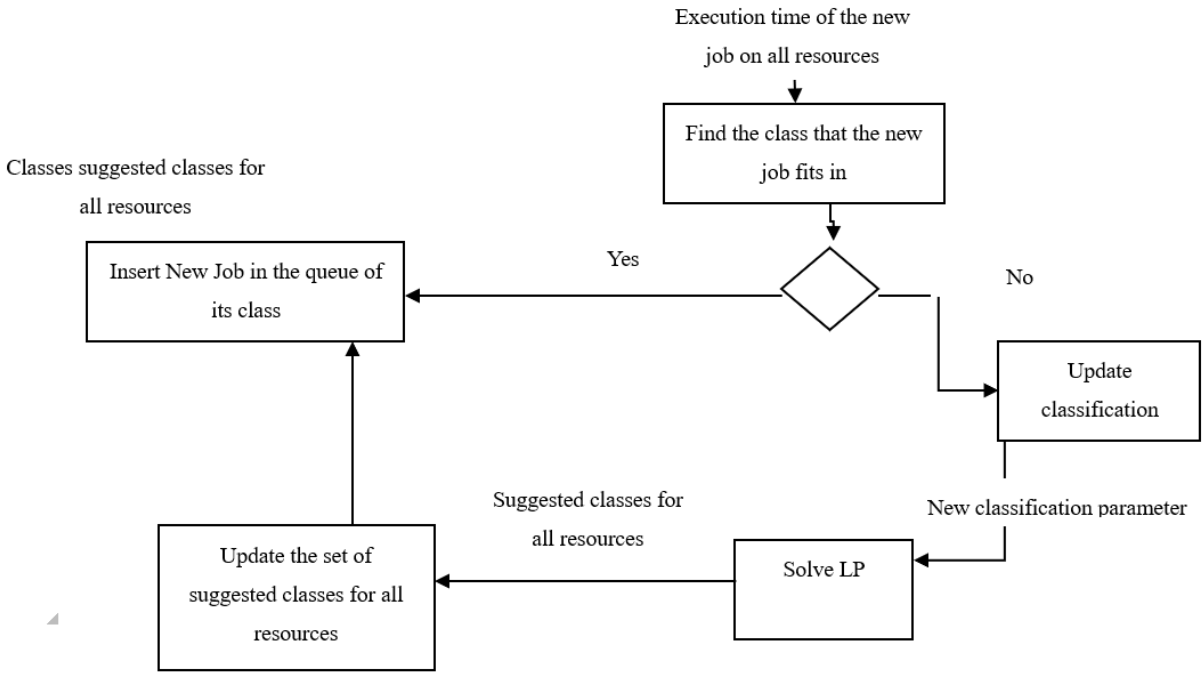


Figure 4. COSHH Queuing Process

The Hadoop Model Scheduling receives two key messages From the Hadoop system: A m`essage refers to a new job Access from user, heartbeat message from free Resources. Therefore, COSHH consists of two main supporters, Sys, where each process is run by receiving one of the messages. When you receive a new functionality, schedule lead the queue process to store the incoming function in Suitable queue. When receiving heartbeat mes-SAGA, scheduling causes the routing process to be set Function to the current free resource.

2.3.3 An Adaptive Scheduling Algorithm for Dynamic Heterogeneous Hadoop Systems

Rasooli &. Down, (2011) designed a scheduling algorithm which classifies the jobs based on their requirements and finds an appropriate matching of resources and jobs in the system. The algorithm is completely adaptable to any variation in the system parameters. The classification part detects changes and adapts the classes based on the new system parameters. Also, the mean job execution times are estimated when a new job is submitted to the system, which makes the scheduler adaptable to changes in job execution times. A high level view of Dynamic Heterogeneous Hadoop Systems is shown in Figure 5.

This scheduling algorithm uses system information such as estimated access rates and times to make scheduling decisions. The goal of this algorithm is to improve the average time of completion of the submitted jobs. It receives a typical two Hadoop schedule Key Messages: New Access Message from user and heartbeat message of free re-charging source. When the scheduler receives a new functionality from a user, the scheduler queue the process and store the incoming function in the appropriate queue.

Let:

- $JobClasses1 = \{C_1, \ldots, C_n\}$, where C_i represents a class of jobs whose minimum share > 0, and share similar arrival rates, estimated execution rates, and priorities.
- $JobClasses2 = \{C'_1, \ldots, C'_n\}$, where C'_i represents a category of all the jobs in the system with similar arrival rates, estimated execution rates, and priorities.

Upon receiving a message (either a "job arrival" from a user or a "heartbeat" from a free resource) from the Hadoop system:

- If incoming message = job (say J) arrival
  1. Get the estimated execution time of J from the Task Scheduler component.
  2. Check if J fits in the current job classification
     (a) If J fits in any class (say C_i)
        i. add J to the queue of C_i
     (b) Else
        i. update the job classification by adding a new class for J (say C_j)
        ii. add J to the queue of C_j
        iii. find a matching of classes and resources using an optimization approach, which results in two sets of suggested classes for each resource as follows:

$$SC_R = \{C_1, \ldots, C_n\}, \text{ where } C_i \in JobClasses1$$
$$SC'_R = \{C'_1, \ldots, C'_n\}, \text{ where } C'_i \in JobClasses2$$

- If incoming message = heartbeat from a free resource (say R)
  - For each free slot in the resource R (say S_R):
    1. Look for a job where

$$job.user.min\_share - job.user.currentshare > 0$$

and

$$job.class \in SC_R$$

and

$$((job.user.min\_share - job.user.currentshare) \times priority) \text{ is maximum}$$

    2. If no job is found, look for a job where

$$job.class \in SC'_R$$

and

$$(job.user.currentshare/priority) \text{ is minimum}$$

    3. If a job is found (say $J_{selected}$), send S_R and $J_{selected}$ to the Task Scheduler. The Task Scheduler chooses a task from $J_{selected}$ and assigns it to S_R.
    4. If no job is found, S_R will left unassigned until the next heartbeat message.

Figure 5. A high level view of Adaptive Scheduling Algorithm for Dynamic Heterogeneous Hadoop Systems

When a heartbeat message arrives from a resource, scheduling prompts the CES to assign a function to the free resource. The algorithm uses a classic function, so when new function up to the system, the queue process-specific class of this function, stores post in the queue of its class. Column A process that sends updated information to everyone and chapters to the routing process, where this routing process uses information for selection Function of current free resources.

2.3.4 Dynamic Capacity Scheduling in Hadoop

Thakur, Singh & Sharma, (2015), aimed in their work decrease the completion time of reduced tasks in map-reduce framework. They tend to organized yarn-site.xml file in Hadoop and additional property values for capacity scheduler. By adding properties for icapacity scheduler we tend to inform to icapacity scheduler algorithmic program, the properties of that are organized in ICapacity-scheduler.xml. The following algorithm of icapacity scheduler is used in which first we initialize the queue, second collect the running container then add to the scheduler. It kill the container, if a queue is below capacity because of lack of demand, and so demand will increase, the queue can solely come to capacity as resources area unit discharged from different queues as containers complete as shown in figure 6.

```
Initialize from Config file
Minimum allocation, maximum allocation, node locality, threshold
Initialize queues
For (Csleaf Q: queueManager)
{ Resource. add (resTopreempt)
If (Resource greater than Resource Calculator)
Preempt + resource (quemagr. Get LeafQueue)
// collect running container
for (sched: scheds)
{ If (Resource. greater than Resource Calculator)
{ for (appsched: sched.getAppSchedulable)
{ For (RMContainer: getLiveContainer)
{ Running container. add(c)
Apps.put (c, appsched.getApp ())
Queues. put (c, sched) }}}
//kill container
If (time! = null)
{ If (time + waitTime BeforeKill < clock.getTime ()) Create preemped container Status (container .getContainerId ()) CompletedContainer (containers, status, RMcontainer.KILL)
```

Figure 6. ICapacity scheduler algorithm

2.3.5 The Improved Job Scheduling Algorithm of Hadoop Platform

In the Improved Job Scheduling Algorithm of Hadoop Platform, the scheduling algorithm based on Bayes Classification where the jobs in job queue are classified into bad job and good job by Bayes Classification. When JobTracker gets task request, it will select a good job from job queue, and select tasks from good job to allocate JobTracker, then the execution result will feedback to the JobTracker. Therefore the scheduling algorithm based on Bayes Classification influence the job classification via learning the result of feedback with the JobTracker will select the most appropriate job to execute on TaskTracker every time. the feature usage are considered for job resource and the influence of TaskTracker resource on task execution, the former of which we call it job feature, for instance, the average usage rate of CPU and average usage rate of memory, the latter node feature, such as the usage rate of CPU and the size of idle physical memory, the two are called feature variables. These two types of feature variable are: 1) Job feature that mainly describes the resource usage situation of job. The value of the variable can be set when the user commits job or obtained via analysis the history information of job execution. The variable values are set from 10 to 1, and 10 is the maximum value which represents the utmost using of resources, 1 corresponding to the minimum value which represents the min usage of resources. The feature variables average CPU usage rate of job, average network usage rate, and average usage rate of IO and average memory usage rate will be adopted. 2) Node feature that represents the computation resource state and quality on a TaskTracker computing node. The variable can be divided into static feature variable of node and dynamic feature variable of node. The static feature variable refers to feature variables which stay static or are constants, while the dynamic feature variable refers to those node properties which change frequently along with time (Guo, Wu, Wu, & Wang, 2015).

With Job scheduling algorithm based on Bayes classification the administrator can adjust task allocation policy through learning the feedback result of every task allocation decision to cluster resources to affect or adjust later allocation strategy constantly. This is accomplished without knowledge of the resource using feature of MapReduce job and resource of TaskTracker on the cluster to improve the correct rate of task allocation and then provide the most system availability. At the end it can reduce administrator's burden, improve management efficiency and reduce the possibility of human error obviously, (Guo, Wu, Wu, & Wang, 2015).

Table 1. Comparison between scheduling algorithms (Tolhit, COSHH, An Adaptive Scheduling Algorithm for Dynamic Heterogeneous Hadoop Systems, Dynamic Capacity Scheduling in Hadoop, The Improved Job Scheduling Algorithm of Hadoop Platform)

	Tolhit (Brahmwar, Kumar, & Sikka, 2016).	**An Adaptive Scheduling Algorithm for Dynamic Heterogeneous Hadoop Systems. (Rasooli, & Down, 2011).**	**COSHH, (Rasooli, & Down, 2014)**	**Dynamic Capacity Scheduling in Hadoop. (Thakur, Singh & Sharma, 2015).**	**The Improved Job Scheduling Algorithm of Hadoop Platform. (Guo, Wu, Wu, & Wang, 2015).**
Proposed work	An algorithm to assist the scheduler in identifying the nodes on which stragglers can be executed so that the overall delay can be reduced.	A new scheduler algorithm to improve mean completion time of submitted jobs.	Design and implement a new Hadoop scheduling system, named COSHH.	Introduced pipeline and queue management in proposed work for improving the performance of Hadoop.	Jobs scheduling optimization algorithm based on Bayes Classification.
The default scheduler algorithm that to be enhanced	Fair Scheduler (HFS).	Fair scheduling	Fair scheduling	Improved capacity scheduler	Fair scheduling
Enhancement	Execution time	Mean completion time of submitted jobs	Improve the mean completion time of jobs.	Improve the existing scheduler issues that help the scheduler to execute the task in less time.	Improvement in execution efficiency and Stability of job scheduling.
Cluster applied to: heterogeneous or homogeneous	Heterogeneous.	Heterogeneous.	Heterogeneity at both the application and cluster levels.	Heterogeneous.	Heterogeneous.
Objective	Aid the scheduler In identifying the nodes on which stragglers can be executed and to solve the stragglers problem.	Improve mean completion time of submitted jobs.	Improve the mean completion time of jobs.	Improve the existing scheduler issues to minimize the scheduler execution time.	Redesign scheduling algorithm based on Bayes Classification and review the shortcomings of the used algorithms.
Methodology	Maintains historical information for every node present in the cluster. Each record in the historical information holds 5 values i.e. map (M1 & M2) and reduce stage weights (R1, R2 & R3). Tolhit assumes “Map function execution weight “as M1 and “Reordering intermediate results weight “as M2 of a map task. R1, R2, R3 stand for the copy, sort and reduce stage weights of reduce task. The history information stored on every node is classified using Genetic algorithm based clustering scheme into k no of clusters.	Uses system information such as estimated job arrival rates and mean job execution times to make scheduling decisions.	COSHH consists of two main processes, where each process is triggered by receiving one of These messages. Upon receiving a new job, the scheduler performs the queuing process to store the incoming job in an appropriate queue. Upon receiving a heartbeat message, the scheduler triggers the routing process to assign a job to the current free resource.	initialize the queue, second collect the running container then add to the scheduler .It kill the container, if a queue is below capacity because of lack of demand, and so demand will increase, the queue can solely come to capacity as resources area unit discharged from different queues as containers complete.	the scheduling algorithm based on Bayes Classification influence the job classification via learning the result of feedback with the JobTracker will select the most appropriate job to execute on TaskTracker every time
performance evaluation	The simulations were done on a 5 node heterogeneous cluster. The performance evaluation of the proposed scheme has been done by series of experiments.	Using simulation, demonstrate that the algorithm is a very promising candidate for deployment in real systems.	However, as it is concerned with other key Hadoop Performance metrics, our proposed scheduler also achieves competitive	Experimental result show that the proposed strategies can result in about 29 to 50 % decrease in average response time.	Not mentioned

	From the performance analysis 27% improvement in terms of the overall execution time	performance under minimum share satisfaction, fairness and locality metrics with respect to other well-known Hadoop schedulers.

3. Proposed hadoob Scheduler Algorithm

At present, the common scheduling algorithm mainly has the default scheduling algorithm FIFO, which adopts a single advanced First out of the queue, regardless of the size or priority of the job, the efficiency is low. The inefficiency and the right Heterogeneous system demand new scheduling algorithms

For this we propose a new scheduling algorithm, from the perspective of resource utilization, that enhance the capacity scheduling algorithm based on the best job priority, resource requirements, node distance to calculate the weight of the job, while observing the real-time and feedback execution status of the jobs , and this will adaptively adjust the node workload, to achieve the task scheduling process load balancing, so as to improve the efficiency of the task scheduling of the cluster.

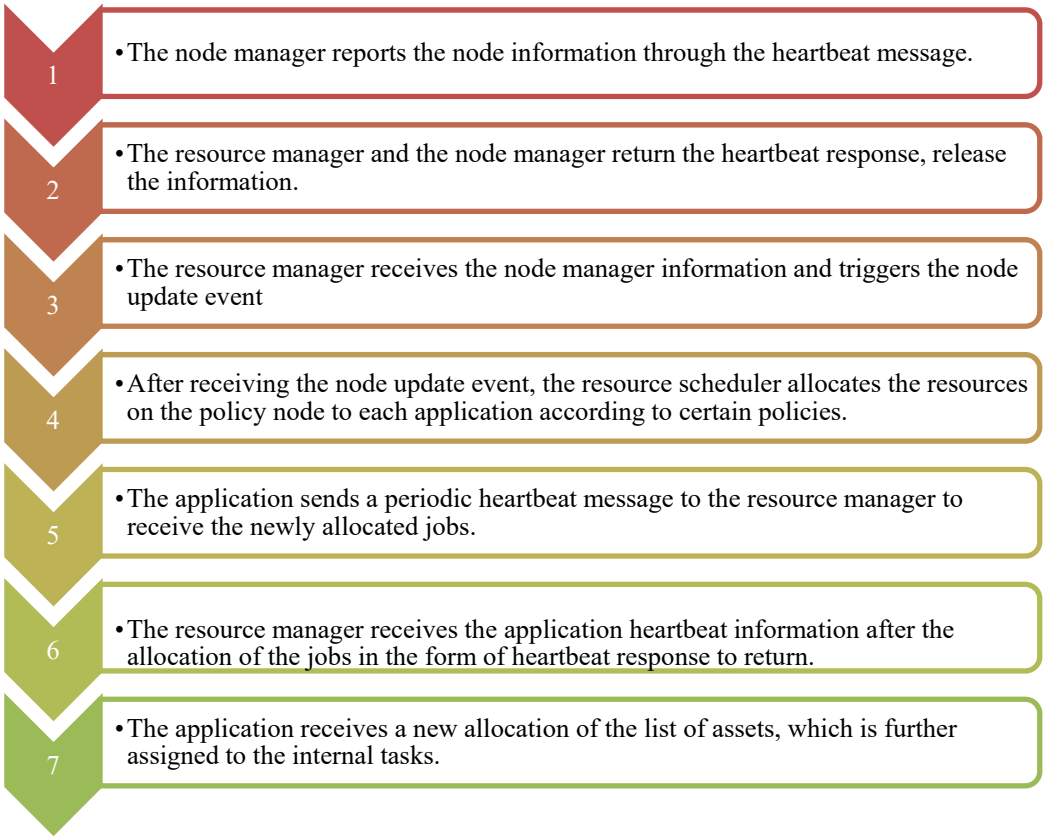
3.1 The Proposed and Improved Algorithm: Weighted Capacity Scheduling Algorithm

In the original algorithm, the job queue is sorted by job submission time and job priority, and then the queue header job is selected. The proposed algorithm first sorts the jobs for each queue according to the job weights, and then allocates the idle slots to the first queue for a job. Therefore, the choice of job weight in the proposed algorithm is an important reference for job scheduling.

In order to avoid long-term waiting for the scheduling task, the proposed algorithm optimizes the order of the jobs in the allocation step, that is, according to the actual status of the job execution, in order to dynamically adjust the weight of the jobs. And this is done by analyzing the weight order, by computing the average of the requested resource size.

To discuss the proposed algorithm in more details we will first describe the resource scheduling model:

The organization and distribution management of cluster resources is one of the most basic functional modules in the Hadoop system. The process can be summarized as shown in figure 7:



4. Summary and Findings

Scheduler of jobs in hadoop to better evaluation plays a major role in enhancing the performance hadoob in general, in this paper we have studied hadoop scedular algorithms, we provided an over view of hadoop and hadoop scheduler algorithms then we have introduced and compare five of the state of art scheduling algorithms to improve that enhance the speed of Hadoop framework , we compare these algorithms in term of The proposed scheduler algorithm, The default scheduler algorithm that is enhanced, Cluster applied to : heterogeneous or homogeneous, methodology, clusters re classification according and Performance evaluation. The compared algorithms focused on reducing the time needed to execute the job with taking into consideration the fairness between the jobs, the algorithms have enhanced the default hadoop scheduler algorithms (fair scheduler, capacity scheduler), we have seen that the field of scheduling in hadoop is a promising research area that need more researches to enhance the hadoop scheduling algorithms.

References

Al Khattab, A., Al-Shalabi, H., Al-Rawad, M., Al-Khattab, K., & Hamad, F. (2015). The Effect of Trust and Risk Perception on Citizen's Intention to Adopt and Use E-Government Services in Jordan. *Journal of service science and management, 8*(03), 279.

Al-Sayyed, R. M., Fakhouri, H. N., Murad, S. F., & Fakhouri, S. N. (2017). CACS: Cloud Environment Autonomic Computing System. *Journal of Software Engineering and Applications, 10*(3), 273.

Al-Sayyed, R. M., Fakhouri, H. N., Rodan, A., & Pattinson, C. (2017). Polar Particle Swarm Algorithm for Solving Cloud Data Migration Optimization Problem. *Modern Applied Science, 11*(8), 98.

Bhosale, H. S., & Gadekar, D. P. (2014). Big data processing using hadoop: Survey on scheduling. *International Journal of Science and Research (IJSR), 3*(10), 272-277.

Brahmwar, M., Kumar, M., & Sikka, G. (2016). Tolhit–A Scheduling Algorithm for Hadoop Cluster. *Procedia Computer Science, 89*, 203-208.

Chen, Q., Zhang, D., Guo, M., Deng, Q., & Guo, S. (2010). SAMR: A Self Adaptive MapReduce Scheduling Algorithm in Heterogeneous Environment, In 10th IEEE International Conference on Computer and Information Technology, CIT'10, (Washington, DC, USA), IEEE Computer Society, pp. 2736-2743.

Dean, J., & Ghemawat, S. (2010). MapReduce: a flexible data processing tool. *Communications of the ACM, 53*(1), 72-77.

DeWitt, D., & Stonebraker, M. (2008). MapReduce: A major step backwards. *The Database Column, 1*, 23.

DongjinYoo, Kwang, & Mong, S. (2011). A comparative review of job scheduling for mapreduce. Multi-Agent and Cloud Computing Systems Laboratory, Proceedings of IEEE CCIS2011.

Ghemawat, S., Gobioff, H., & Leung, S. T. (2003). *The Google File System, 37*(5), 29-43. ACM.

Guo, Y., Wu, L., Yu, W., Wu, B., & Wang, X. (2015). The Improved Job Scheduling Algorithm of Hadoop Platform. arXiv preprint arXiv:1506.03004.

Hamad, F., & Adwan, O. (2018). Policy Based Approach for Information Transfer over Mobile ad hoc Network using Messages Privacy Control. *Modern Applied Science, 12*(5), 22.

Hamad, F., & Alawamrah, A. (2018). Measuring the Performance of Parallel Information Processing in Solving Linear Equation Using Multiprocessor Supercomputer. *Modern Applied Science, 12*(3), 74.

He, C., Lu, Y., & Swanson, D. (2011, November). Matchmaking: A new mapreduce scheduling technique. In Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on (pp. 40-47). IEEE.

Hudaib, A. A., & Fakhouri, H. N. (2016). An Automated Approach for Software Fault Detection and Recovery.

Hudaib, A. A., Fakhouri, H. N., Al Adwan, F. E., & Fakhouri, S. N. (2016). A Survey about Self-Healing Systems (Desktop and Web Application). *Communications and Network, 9*(01), 71.

Jagmohan, Chauhan (n.d.). Dwight Makaroff and Winfried Grassmann, "The Impact of Capacity Scheduler Configuration Settings on MapReduce Jobs".

Kaabneh, K., Abu-Hammad, E., & Hamd, F. (2007, November). Enhanced Skin Detection Technique Using Block Matching. In Signal Processing and Communications, 2007. ICSPC 2007. IEEE International Conference on (pp. 21-24). IEEE.

Kc, K. & Anyanwu, K. (2010). Scheduling Hadoop Jobs to Meet Deadlines. In Proc. CloudCom, pp.388-392.

Mark, Y., Nitin, G., & Shiwali, M. (2009). Towards a Resource Aware Scheduler in Hadoop. In Proc. ICWS, 2009, 102-109.

Radheshyam, N., Niteshaheshwari, R. R., & Vasudeva, V. (). Job Aware Scheduling Algorithm for MapReduce Framework. 3rd IEEE International Conference on Cloud Computing Technology and Science Athens, Greece.

Rao, B. T., & Reddy, L. S. S. (2012). Survey on improved scheduling in Hadoop MapReduce in cloud environments. arXiv preprint arXiv:1207.0780.

Rasooli, A., & Down, D. G. (2011). An adaptive scheduling algorithm for dynamic heterogeneous Hadoop systems. In Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research (pp. 30-44). IBM Corp.

Rasooli, A., & Down, D. G. (2014). COSHH: A classification and optimization based scheduler for heterogeneous Hadoop systems. *Future Generation Computer Systems, 36*, 1-15.

Reddy, V. K., Rao, B. T., & Reddy, L. S. S. (2011). Research issues in cloud computing. *Global Journal of Computer Science and Technology.*

Sandholm, T., & Lai, K. (2010, April). Dynamic proportional share scheduling in hadoop. In Workshop on Job Scheduling Strategies for Parallel Processing (pp. 110-131). Springer, Berlin, Heidelberg.

Sun, X., He, C., & Lu, Y. (2012). ESAMR: An Enhanced Self-Adaptive MapReduce Scheduling Algorithm, IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS), pp. 148-155.

Usha, D., & Jenil, A. A. (2014). A survey of Big Data processing in perspective of Hadoop and mapreduce. *International Journal of Current Engineering and Technology, 4*(2), 602-606.

White, T. (2012). Hadoop: The definitive guide. "O'Reilly Media, Inc.".

Yong, M., Garegrat, N., & Mohan, S. (2009, December). Towards a resource aware scheduler in hadoop. In Proc. ICWS (pp. 102-109).

Zaharia, M. (2009). Job scheduling with the fair and capacity schedulers. *Hadoop Summit, 9.*

Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., & Stoica, I. (2010, April). Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In Proceedings of the 5th European conference on Computer systems (pp. 265-278). ACM.

Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, L., Shenker, S., & Stoica, I. (2010). Delay scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling, In 5th European Conference on Computer systems (EuroSys).