DESIGN AND DEVELOPMENT OF HADOOP DYNAMIC SLOT ALLOCATION TECHNIQUE USING MAPREDUCE IN HEALTH CARE

Abstract

The present study uses two different approaches to schedule tasks: the self-adaptive cluster groups, which form data analysis and the Hadoop Dynamic Slot Scheduler. These two methods reduce the total queue time by several experiments. The previous model uses several distribution formulas for the planning task and the latter uses a meta-heuristic scheduler to improve cloud planning efficiency.

The first part concentrates the research on the algorithm programming through subtask resources that improve the performance of data analysis to support the customer's task requirements. The focus is to plan the work using a cloud-based computing model, total calculation task, speed and probability of distribution by using a MapReduce slot that provides the associated resources.

In this work the selection of idle VMs to reduce the overall runtime and resource consumption via the Hadoop dynamic slot allocation. It considers that the whole task is independent. Tasks are carried out automatically during the task planning process, which reduces the planning load and runtime.

These two techniques effectively reduce the time for each task, increase network performance and improve the scalability of the cloud computing model. The results of the simulation show that the proposed allocation and scheduling of the Hadoop Dynamic Slot is effective in choosing VM than other existing techniques.

Keywords: Health Care; Hadoop; MapReduce; BigData.

Authors

Dr. Srikanth Reddy E
Associate Professor
Department of CSE
Vaageswari College of Engineering
karimnagar, Telangana, India
srikanth574@gmail.com

Dr. D Srinivas Reddy
Associate Professor
Department of CSE
Vaageswari College of Engineering
Karimnagar, Telangana, India
srinivasreddydhava@gmail.com

I. INTRODUCTION

Big Data in healthcare is being utilized to foresee plagues, fix infection, improve personal satisfaction and evade preventable passings. With the total populace expanding and everybody living longer, models of treatment conveyance are quickly changing, and huge numbers of the choices behind those progressions are being driven by data. In customary hadoop framework, the ace allot equivalent assignment to all hub. This strategy gets come up short in heterogeneous condition, where execution of every single hub considers in an unexpected way. To maintain a strategic distance from this situation we will consider advance Hadoop big data system. The data blast for example creating huge measure of data. Also, it is hard to oversee, Retrieve and preparing by utilizing customary base framework. This healthcare association has made by keeping record, and administrative necessity. This potential will assist with improving personal satisfaction. Hadoop comprise of essentially two Factors;

1. Map Reduce
2. HDFS (hadoop distributed file system)

The Hadoop stage which is in circulated way and conveyed in clustering design. Furthermore, cluster ought to be homogeneous. This immense size of investigation will require enormous calculation which should be possible with assistance of appropriated handling, Hadoop. MapReduce, a mainstream registering worldview for huge scope data preparing in distributed computing. Sickness and their potential indications are bunch together and send it as contribution to framework which create aggregate data.

After investigation done, on the off chance that we give side effects, at that point framework will produce name of illness. Calculation will make away from of yield in graphical configuration. Age, Gender, Disease, Region, Survival Status, Insurance are some gathering classes dependent on which investigation and gathering should be possible.

This will be accomplished with the assistance of Hadoop Framework with the assistance of which we can do an exceptionally quick examination for big data. It will be an awesome effect if the framework utilized by Govt. of India. This system comprises of two capacities to be specific map () and reduce (), each having various boundaries. Map work contain two boundaries for example key and worth. Of course, this structure appoints esteem 1 to allkeys. Hadoop utilizes a specific scheduling instrument for dispensing undertaking to each hub. Scheduling is a significant part of Hadoop which guarantees reasonable errand allocation and burden adjusting. In heterogeneous clusters the exhibition of each hub contrasts from every single other hub. To augment the presentation of such clusters and for better asset usage, the assignment scheduling ought to be versatile. In hadoop data won't store on single cluster however it will save money on number of clusters. So data will be continuing in equal way to accomplish execution. Hadoop is attempting to keep reinforcement of data. Quantities of times data will get evaporated, to dodge this gathering of clusters will be produced.

II. MAP REDUCE

MapReduce is an advanced data processing paradigm in clusters and data centers, which in recent years has been very popular. MapReduce is known as Hadoop as an open-source implementation. Many studies have been carried out to optimize MapReduce or Hadoop, but there are still some important challenges to the use and performance improvement of a Hadoop cluster. The conceptual structure for a broad data analytics project in the health sector is similar to conventional information management or analytical projects in the health sector. However, how these procedures are done is completely different. A business intelligence platform can be used on an independent basis in a routine health research project. Big data is, as described, just a large process spread across the multiple nodes. For several years this idea has been around. In the study of incredibly broad data sets, it is relatively new to get an insight into better-informed patient decisions as health care providers begin to use their large-scale data repositories. In addition, open-source platforms like Hadoop / MapReduce, which are available in the cloud, promoted Big Data Analytics applications in healthcare.

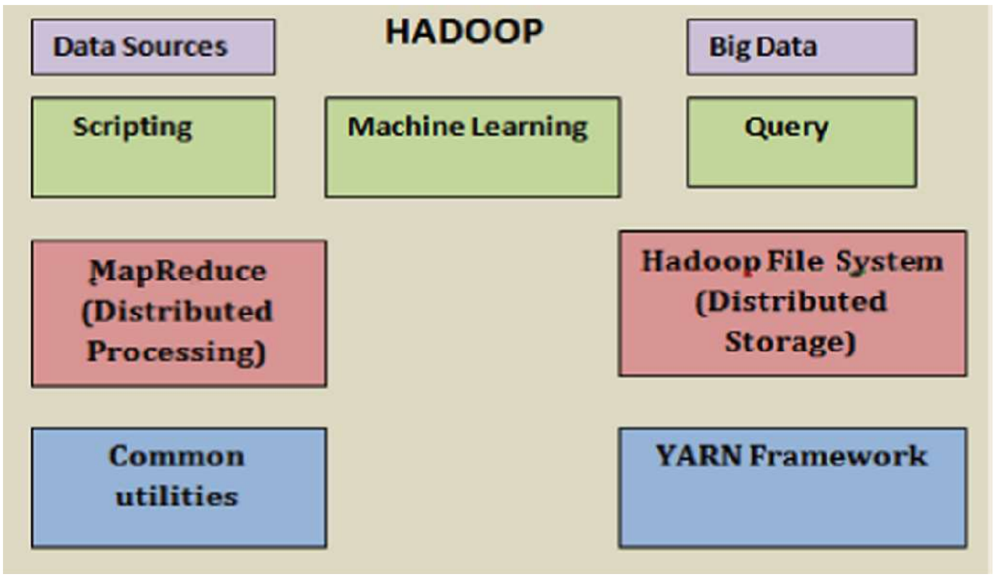

Figure 1: Hadoop Architecture

To enhance the performance of a MapReduce cluster in two ways by optimizing slot use. Initially, to divide the slots into two kinds: busy (i.e. work) and idle (i.e. no work running). With the overall map number and decreased slots, the optimizing approach is to improve slots utilization through the optimization of the number of busy Slots and the reduction of idle slots (i.e., macro level optimization). Secondly, not every active slot can be effectively used. It should be noted. Our optimization approach (i.e. the optimization of micro levels) aims to improve the efficiency of operating slots after macro-level optimization. In particular, two main affective factors are identified: (1). Tasks of speculation; (2). Location of data. In this way, we are proposing to improve the performance of a shared Hadoop cluster under fair scheduling of the Dynamic MapReduce user.

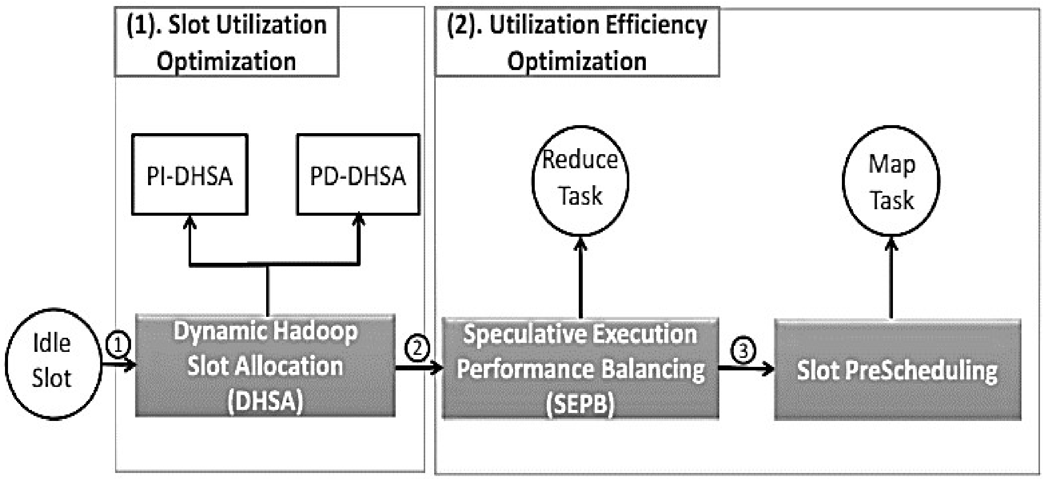
(1). Slot Utilization Optimization
(2). Utilization Efficiency Optimization
PI-DHSA
PD-DHSA
Reduce Task
Map Task
Idle Slot
①
Dynamic Hadoop Slot Allocation (DHSA)
②
Speculative Execution Performance Balancing (SEPB)
③
Slot PreScheduling

Figure 2: Dynamic Framework

Figure 2 gives a Dynamic Mapreduce overview. It consists of three slot assignment methods, i.e. Dynamic Hadoop Slot Assignment. The improvement of performance from different aspects is taken into account in each method. When equity is maintained when tasks are not complete, DHSA seeks to maximize the use of slot. SEPB recognizes the slot resource inefficiency problem caused by speculative tasks for a Hadoop cluster. It works for the Hadoop speculative planner to balance the performance of a single job and various jobs. Slot Planning improves slot efficiency and performance by improving map functions data location while maintaining fairness. It prescribes tasks when cartography tasks are pending, but no idle map slots are permitted.

Dynamic Mapreduce can significantly optimize the use and performance of a Hadoop cluster with these three techniques:

1. Dynamic Mapreduce works to enhance slot use with DHSA whenever an idle slot is available. Subject to a number of constraints, like fairness, load balance he decides whether or not.
2. When the assignment is true, Dynamic Maperduce further improves performance by enhancing SEPB slot efficiency. Because speculation can increase the performance of a single job and at the expense of the efficiency of the cluster, the SEPB acts as a balance between efficiency and a batch of job opportunities. It works on whether the idle slot is assigned to the pending task or to the speculative task, together with the speculative programmer Hadoop.
3. Dynamic Mapreduce can continue to increase the slot use efficiency by designing Slot from the data optimization aspect when assigning the idle slots to pending / speculative map tasks.

III. RELATED WORK

Dynamic Hadoop Slot Allocation (DHSA): The current map reduction design is underused, as the amount of maps and tasks varies over time and therefore less than the number of map / decrease slots allocated. The foundation for our dynamic slot distribution policy is that idle map slots (or reductions) can exist at various times while the work from the map stage decreases. For those overloaded, unused map slots can be used to reduce the workload and improve MapReduce performance. For example, at the start of MapReduce workload computation, only map computing tasks happen, and computing won't cut work, that means that all computing workloads will be on the map side. In this case, for map tasks, we can use idle slots. This means that the implicit assumption that the map tasks can only work on map slots and reduce tasks on slots is distorted by the current MapReduce Framework. Instead, the following is modified: both maps can be executed and tasks reduced on either the map or the slots reduced.

However, there are two challenges to be considered:

- A significant measure of fairness is the Hadoop Fair Scheduler (HFS). We say that it is fair to allocate all the pools with equal resources. In HFS, task slots are distributed primarily across pools and jobs are allocated within the pool. Moreover, the job calculation of MapReduce comprises two components: the work calculation map phase and the calculation of task reduction. One question concerns how to define and ensure fairness in the dynamic slot assignment policy.

- The demands for resources are generally different between the slot map and the slot reduction. The reason is that the map task and the reduction of tasks often have completely different execution patterns. Tasks reduction usually requires much more resources such as network memory and bandwidth. Each map slot needs to be configured to take more resources in order to simply allow tasks that can be restricted to using map slots, reducing the effective number of slots at each node, which makes them less resourceful during operation. Therefore, it is important and necessary for this difference to be accurately designed a dynamic allocation policy. In relation to (C1), we propose a Dynamic Hadoop Slot Assignment (DHSA). There are two alternatives: PI-DHSA and PD-DHSA, which take into account the fairness from different aspects.

1. **GWO Algorithm:** Task scheduling is a technique in cloud computing to ensure a dedicated resource is assigned work and that assigned work is completed. The resources include virtual computer elements or hardware. The programming activity in turn is performed by a programmer. The scheduler assigns multiple users to occupy a certain part of a resource for a good QoS. Scheduling is a basic calculation process and an execution model or an integral part of a cloud computing model. The scheduler allows the computer system to try multi-tasking for each CPU.

   The scheduler prioritises each task according to the user's needs and therefore the multi-tasks in parallel distributed applications set the schedule of work over idle VMs to complete the process soon. The main problem with task execution lies in the increase of parallelism, as it depends on another task to perform a task in cloud computing. Figure 3.1 shows the architecture of the proposed method.

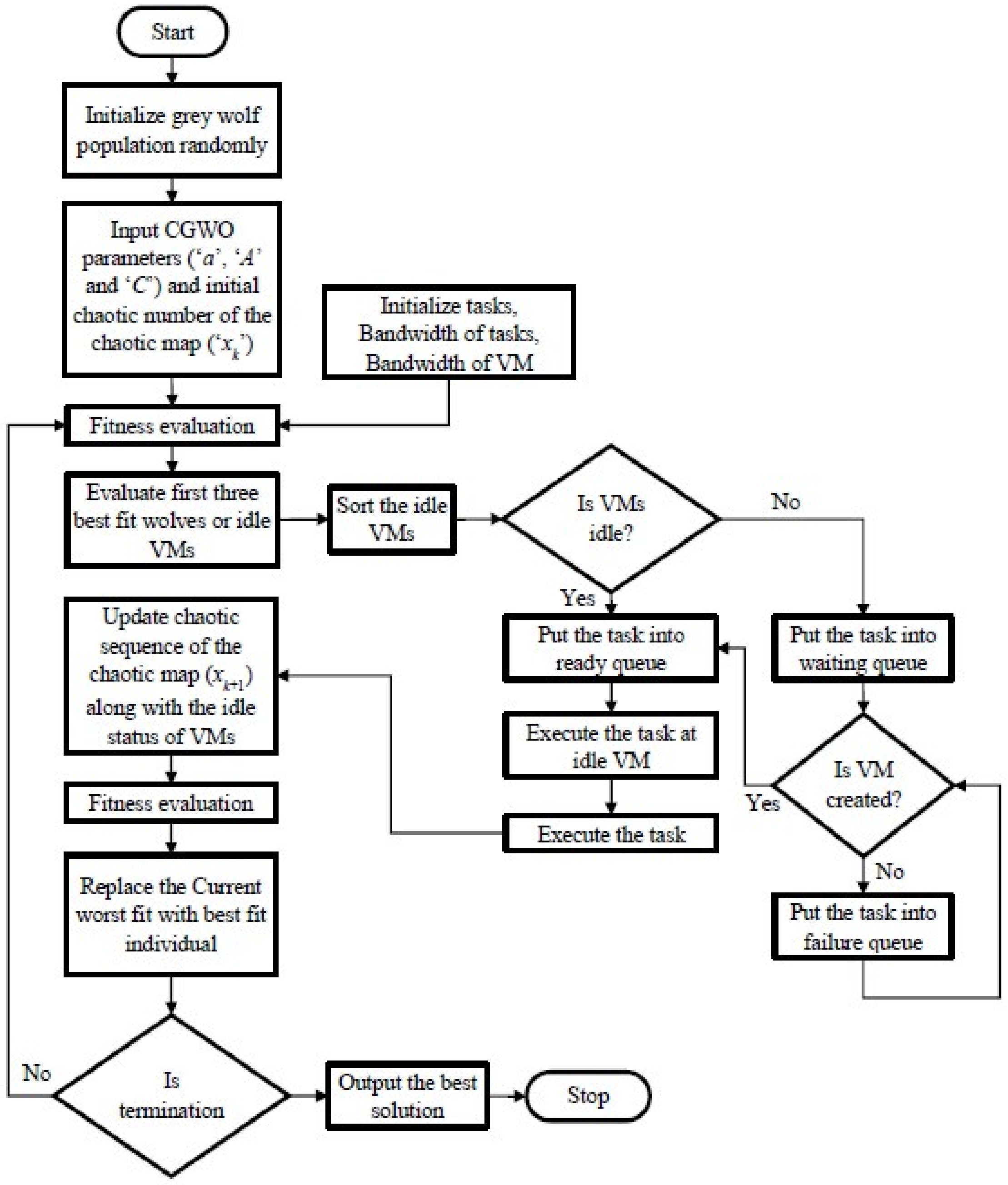
Start
Initialize grey wolf population randomly
Input CGWO parameters ('a', 'A' and 'C') and initial chaotic number of the chaotic map ('x_k')
Initialize tasks, Bandwidth of tasks, Bandwidth of VM
Fitness evaluation
Evaluate first three best fit wolves or idle VMs
Sort the idle VMs
Is VMs idle?
No
Yes
Put the task into ready queue
Put the task into waiting queue
Update chaotic sequence of the chaotic map (x_{k+1}) along with the idle status of VMs
Execute the task at idle VM
Is VM created?
Yes
Fitness evaluation
Execute the task
No
Replace the Current worst fit with best fit individual
Put the task into failure queue
No
Is termination
Output the best solution
Stop

Figure 3: Slot Scheduling Architecture

2. **Grey Wolf Optimization Algorithm:** Golf Wolf Optimization is a technique of swarm intelligence. The social intelligence of grey wolves, namely chasing and leadership, is the inspiration behind the GWO algorithm. There is a common social hierarchy in every pack of grey wolves that dictate domination and power. Figure 3.2 shows grey wolves' social hierarchy.

   The mighty wolf is Alpha, who hunts the whole pack and eats it. The strongest wolf is the alphan, which chases, migrates and feeds the whole pack. If the alpha wolf isn't, sick or dead, the strongest wolf from beta wolves takes the lead.

Delta and Omega are less powerful and dominant than alpha and beta. As Figure 3.1 shows. The GWO algorithm is based primarily on social intelligence. The hunt for grey wolves is another inspiration. Gray Wolves take a range of successful chasing, encircling, harassing, attacking and doing so. This enables you to pursue big beasts.

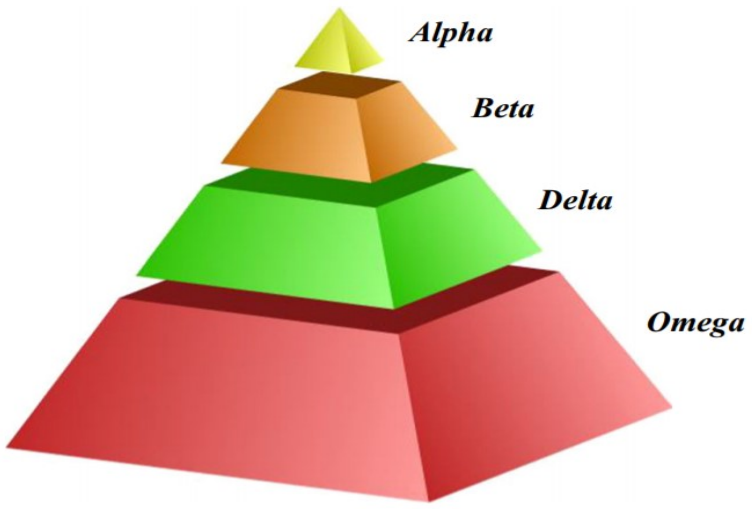

Figure 4: Social Hierarchy of Wolves

Inspiration: The Candidate family is the Gray Wolf. Grey wolves are considered the highest food chain predators. The grey wolves prefer a package mostly. The group size is average between 5 and 12. The social hierarchy is very strict, particularly interesting, as shown in Figure 3.2. The leaders are an alphas-named man and woman. It is mainly the Alpha who decides to hunt, sleep, wake, etc. The alpha's decisions are determined by the pack. There is, however, a certain democratic position where an alpha follows the other wolves.

The whole pack identifies the alpha by holding its tails down during meetings. The Alpha Wolf is also known as the dominant wolf because the pack should follow guidance. The pack can only matte the alpha wolves. Of course, alpha is not necessarily the most important part of the packaging, but the most important part of packaging management. This demonstrates the importance of the organization, discipline and strength of the package.

The second stage of the grey wolves' hierarchy is beta. Betas are wolves that support alpha when making decisions or other packaging. The beta wolf can be male or female. If one wolf is missing or is very older, is probably the best alpha candidate. The beta wolf must comply with the alpha, but also the other bottom wolves should control. It is an alpha adviser and discipliner in the packaging. Beta strengthens the Alpha commands and gives the entire package Alpha feedback.

The lowest omega is the grey wolf. The Omega is the wolves' role. All other dominant wolves must always be present. They're the final eating wolves. The omega is not an important person but when the omega is lost the entire package faces inner difficulties and challenges. It's because every wolf has omega(s) of frustration and violence. The whole package is satisfied and the dominant structure is maintained. In some cases, Omega is also the kids in the pack. It's called a subordinate, if a wolf is not alpha, beta or omega. Delta Wolves shall be presented by Alphas and Betas, but omega is dominated. There are Scouts, Sentinels, Elders, Hunters and Guards in this category. Scouts shall monitor the territorial borders and warn the pack in case of any danger. Sentinels protect and ensure packaging safety. Elderly people are former alpha or beta wolves. Hunting for animals and pocket feeding helps huntsmen to obtain alphas and betas. Finally, it is the responsibility of the custodians to take care of the weak, diseased and wounded wolves. Apparently, hunting groups have an interesting social behaviour by grey wolves in addition to their social hierarchy. The main phases of hunting grey wolves are as follows:

- Track, chase and approach the prey.
- To pursue, encircle and harass the prey until it stops moving.
- Attack on the prey.

Figure 5: illustrates the steps. In order to develop and optimise GWOs, this work models hunting and the social hierarchy of grey wolves.

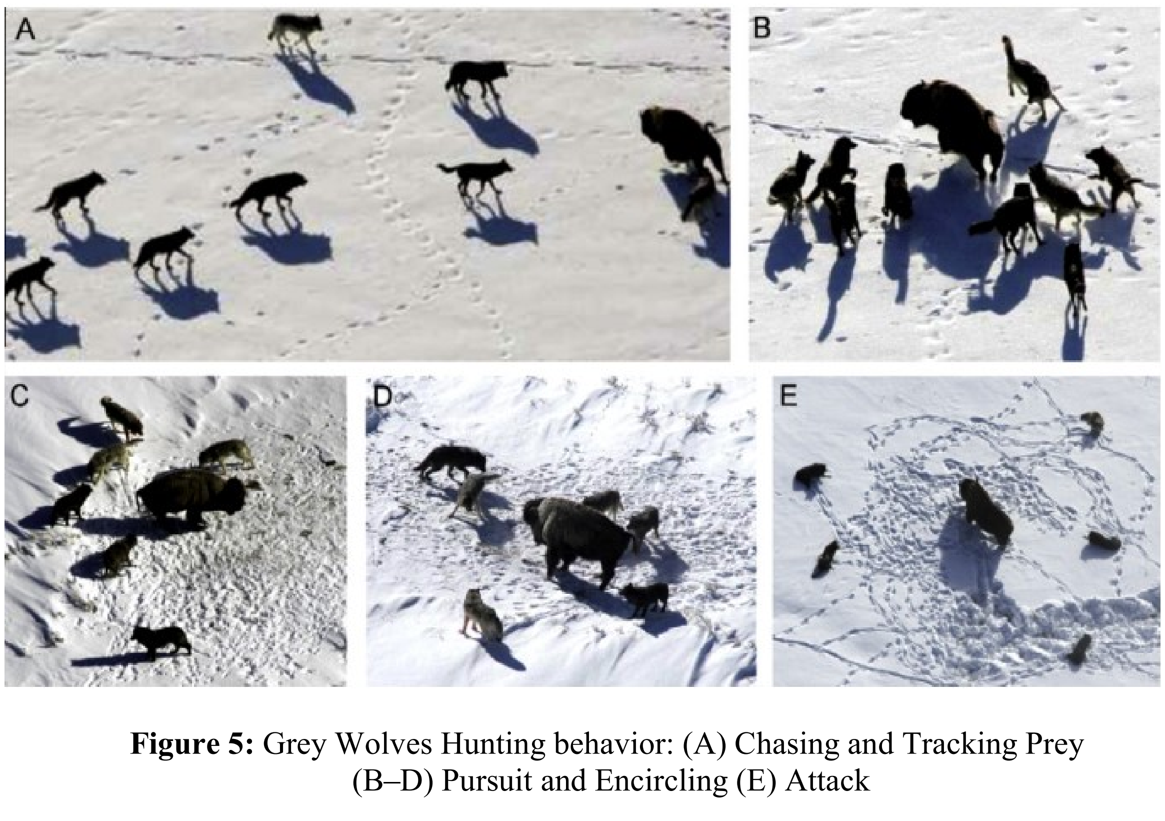

Figure 5: Grey Wolves Hunting behavior: (A) Chasing and Tracking Prey (B–D) Pursuit and Encircling (E) Attack

The Pseudocode of GWO is given below:

```
For i = 1 : n
zi ←ith row vector elements of F;
Generate initial search agents Gi (i=1, 2,….,n)
```

$$G_i^j = \begin{bmatrix} G_1^1 & G_2^1 & G_3^1 & G_4^1 & \cdots & G_{Gd-1}^1 & G_{Gd}^1 \\ G_1^2 & G_2^2 & G_3^2 & G_4^2 & \cdots & G_{Gd-1}^2 & G_{Gd}^2 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ G_1^{Gs} & G_2^{Gs} & G_3^{Gs} & G_4^{Gs} & \cdots & G_{Gd-1}^{Gs} & G_{Gd}^{Gs} \end{bmatrix} \quad (5.1)$$

```
Initialize the vector’s a, A and C
$A = 2a \cdot rand - a$ (5.2)
$C = 2 \cdot rand$ (5.3)
where the vector a reduces linearly from 2 to 0 as the number of iterations increases.
Estimate the fitness value of each hunt agent
$D = \left| C \cdot G_p(t) - G(t) \right|$ (5.3)
$G(t+1) = G_p(t) - A \cdot D$ (5.4)
where,
$G_\alpha$ is the first best hunt agent,
$G_\beta$ is the second best hunt agent and
$G_\delta$ is the third best hunt agent.
I=1 (Iteration start)
Repeat
For i=1: Gs (pack size of grey wolf)
Update the current hunt agent location using following expressions,
$G(t+1) = \frac{(G_1 + G_2 + G_3)}{3}$ (5.5)
$G_1 = G_\alpha - A_1 \cdot D_\alpha$,(5.6)
$G_2 = G_\beta - A_1 \cdot D_\beta$ and(5.7)
$G_3 = G_\delta - A_1 \cdot D_\delta$ (5.8)
$D_\alpha = |C_1 \cdot G_\alpha - G|$,(5.9)
$D_\beta = |C_2 \cdot G_\beta - G|$ and(5.10)
$D_\delta = |C_3 \cdot G_\delta - G|$ (5.11)
End for
Calculate the fitness value of the entire obtained hunt agents
Update the values of the vectors a, A and C
Update the value of first three best hunt agent $G_\alpha$, $G_\beta$, $G_\delta$.
I=I+1
Check if I ≥ Imax (maximum iterations i.e. the Stopping criteria)
Output largest k eigenvectors G
End For
Z ← {zi | i = 1, 2,…, n}.
```

HFS is fair to assign and reduce the phase of slots to pools with minimal guarantees at map stage. DHSA (PI-DHSA) Pool-independent expends the HFS by assigning a cluster slot, irrespective of the pools. The number of typed slots for each type-pool at each stage should be taken into account when equal. The PI-DHSA slot assignment flow as shown in Figure 2. It is a dynamic phase-based assignment policy.

Intra-phase slot dynamic assignment. There are two sub-pools for each pool, i.e. a plan and a reduction phase pool. At every stage, every pool receives its share of slots. Unused slots may be dynamically purchased from other pools in the same phase from an overloaded pool whose demand is above its share. For example, if Pool 2 or Pool 3 are under-used and vice versa, Pool 1 may use the map slots for Pool 2, or Pool 3 based on a fair max minute policy.

Inter-phase dynamic slot assignment. The intraphase dynamical slot allocation enables us dynamically to allocate the slot across typed phases after both the map and reduction phases. This means that if some unused slots are in the decrease phase, and there are not enough map slots in the map phase for map tasks, some idle slots are borrowed to maximize the use of the cluster.In general, there are four potential scenarios. Allows NM and NR to be the total map tasks while SM and SR are the total map number and reduce the user slots.

In other words, if first of all we quantify the overall demand for map slots, if it comes from a computer node and the existing workload for MapReduce is reduced. Next, in the four above scenarios, we will decrease tasks based on the map requirements and map slots. The number of map slots not used and the number of map slots unused.

Practically speaking, at every stage, we may wish to reserve some unused slots to minimize potential hunger for MapReduce jobs. One question is how the number of reserved slots can be dynamically controlled.

We give 2% of the map slots taken and reduce slots defined as the percentage of the map unused. We are reducing the number of slots to be borrowed. In this way the amount of unused map can be limited, slots can be reduced and tasks reduced in each of the task trackers heartbeat.

Users can use these two parameters to balance the compromise between optimizing performance and minimizing hunger. If, as described in Appendix F of the supplementary material, users can configure parameters with large values if they can improve their performance.

Furthermore, Challenge (C2) reminds us that maps and slot reduction cannot be treated and unused slots purchased and task reduced. Instead, we should know various resource map sizes and reduce slots. It therefore proposes a slot-based approach to the problem. Based on the weights, we can determine how many maps and how many tasks should be drawn during operation. Consider, for example, an 8/4 map tasktracker / slow down slot setup. Suppose that the map weights and the reduction of slots are 1 and 2 depending on different needs for resources.

3. **Pool-Dependent DHSA (PD-DHSA):** Unlike PIDHSA, which is fair in its dynamic allotment of slots regardless of pools, pool dependent DHSA (PDDHSA) considers the dynamic allotment of slots in pools equally to be fair, as Figure 3.4 shows. It is supposed to be egotistic for every pond, composed of two parts: a map and a reduction pond. In other words, it always tries to fulfil a common map and reduce slots as much as possible before it is loaned to other pools, for its own requirements at the map and reduction phase. It considers fair that the total number of maps and the number of slots assigned is the same.

   There is also a special circumstance that should be taken into account. Note: four slot allocation attempts above that fail due to the position of the map data for all pools. The Hadoop cluster, for example, may add a new node. It may be vacant and does not contain any details. Therefore, data position cannot be completed for all map tasks and does not include all map tasks. The deficiencies mean that reduction tasks are still not available for all pools. However, any pending map tasks may occur. Thus, certain map tasks should be carried out in order to maximize system use by disregarding the position of the data in a new computer node. To achieve that, we mark the map tasks visited for Map for each job. The data location is considered without a scanned task when visited for Map. Otherwise, map tasks will relax the restriction on data location.

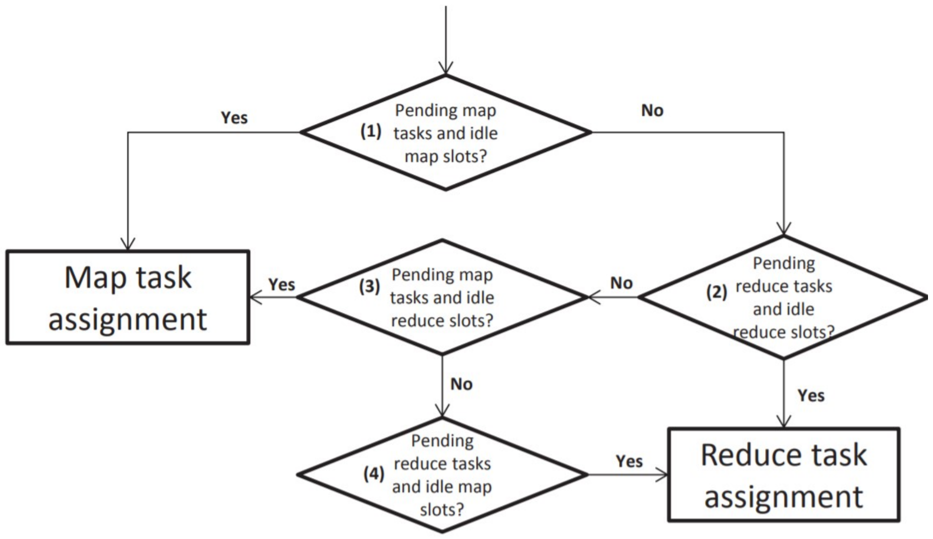
(1) Pending map tasks and idle map slots?
Yes
No
Map task assignment
Yes
(3) Pending map tasks and idle reduce slots?
No
(2) Pending reduce tasks and idle reduce slots?
No
Yes
(4) Pending reduce tasks and idle map slots?
Yes
Reduce task assignment

Figure 6: The Slot Allocation for PD-DHSA

4. **Execution Performance Balancing:** The time required to carry out the MapReduce task is sensitive to slow work. There are several reasons, such as hardware and software defectives. Hard Straggler and Soft Straggler are divided into two types.

   - **Hard Straggler:** a task blocked by the endless resource waiting. It can't end without killing it.
   - **Soft Straggler:** a task which can be calculated successfully, but lasts much longer than common tasks.

The hard traffic jammer must be killed and we must do another job or call backup after identification. However, between the soft tracker and the backup role there are two choices:

- The backup task is completed first or the same by Soft Straggler. A backup task is not required in this case.
- Soft straggler completes the backup mission at a later stage. We have to disable it and conduct a backup procedure immediately.

Hadoop is used to address the problem of traffickers in speculative execution. Instead of diagnosing and fixing the draggling task that helps to dynamically detect with heuristic algorithms such as LATE. However, once the straggler is detected, the fact is that he cannot kill him immediately.

Instead it creates a backup task that allows the straggler to operate simultaneously, i.e. between the straggler and the backup job a computational overlay exists. When both tasks are completed, the task kill operation takes place. While speculative implementation may reduce the execution time for a single job, it does not cost cluster efficiency. Speculative tasks aren't free, i.e. compete for certain sources with other work tasks, including slot map slots, which can adversely affect the performance of a lot of jobs. Speculative tasks are thus created with a view to alleviating the negative impact of a variety of jobs.

An intuitive solution is to meet the pending tasks first in order to maximise the performance of many jobs before considering speculative tasks, given the available task areas. If a node has an idle map slot, we need to select excellent map work before we are looking for speculative map work for a great number of tasks. Also, keep in mind that our dynamic programmer can reduce this task by no longer limiting the map slot to the map work. In this respect, the following should be considered: pending map job work; pending job cutbacks; speculative map working; and reducing the speculative tasks to maximise additional performance. Slots are also being reduced.

Hadoop selects a job from a job that is based on the following priority: firstly, any failed task has the highest priority. Second, consideration is given to the pending tasks. The map is selected first for tasks involving local data to a computer node. Hadoop is finally looking for a slower task to perform speculatively. We specify a variable percentage of jobs controlled for pending work with a value field of 0.0 to 1.0, user-configurable for maximum jobs that are monitored to pending jobs, i.e. the maximum number of tasks checked on the pending map.

With the performance of specific tasks, users can balance the compromise between performance in a set of tasks and response times for a given task. Better performance is achieved for the whole job if the percentage of jobs tested for pending tasks is large or if the response time is improved for one job. The information about our mechanism is that, when there is a silent map slot, we check jobs for map tasks first. The total number of remaining maps for each Ji job is calculated, and tasks are reduced through all the work from Ji to Jj. Next, for every Ji job we check the following requirements:

However it will create another problem if the planning of the speculative job is delayed. If the harsh trailer or soft trailer (S2) is not solved as quickly as possible, the resources allotted to the slot are not used effectively, thereby reducing the cluster efficiency. In order to address this problem we now use a simple heuristic algorithm:

To estimate how long each task will be completed. If the average workflow is twice as long, we kill the slot directly. Since failed / performed tasks are of the utmost priority in Hadoop, a backup job is set up to replace it quickly, improve work performance and reduce the negative effects on cluster efficiency. Finally, work is done similarly with the tasks of the speculative reduction. The detailed implementation of SEPB is given in algorithm 3 of the additional material.

The advantage of SEPB over LATE lies in its slot distribution policy. For LATE a backup task is created when a job is removed and a slot is assigned to run directly from a job point of view where the total number of speculation tasks is below the speculative threshold a parameter to determine the number of speculative tasks that are performed. SEPB, on the other hand, is responsible globally for assigning resources to speculative tasks taking into consideration several jobs. When tasks for several jobs are pending, it will postpone slot allocation to speculation tasks.

For J1 all idle slots with LATE are four speculative tasks. However, with SEPB, the 4 idling slots will have the remained J3 J4 tâches assigned in place of J1 's speculative tasks to improve efficiency of slot use. The relationship between SEPB and LATE is that SEPB works above LATE and improves LATE in the planning of speculative tasks. LATE first examines the number of speculative tasks running when it recognizes a dumb task and an idle slot. LATE shall inform SEPB immediately, rather than immediately create speculative tasks. The SEPB will then decide whether or not to create a speculative task for the overall re-calculation of the data by examining several jobs. If SEPB finds outstanding tasks, it assigns the idle slot a pending task. If not, a new speculative task will be created for the idle slot.

5. **Slot Prescheduling:** A strong and important tool to maximize use of slots and results is the calculation activity of the local data node (i.e. data placement). Delay Scheduler was proposed to boost the data position of MapReduce. He delays his planning for a little while when he learns that there are no local map tasks in his data on a node. It's at the expense of equity, though. Between the data position and fairness optimization with delay planner there is a compromise. The delay schedule in HFS is not sufficient and the data locality can still be optimised. A difficult question will be: can solutions continue to improve the location of the data without affecting fairness.

   To propose a pre-programming technique for this question, which could improve the data location without damaging the fairness of the MapReduce jobs? It is achieved by slave nodes at the expense of the load balance in contrast with the delay scheduler. Because of the load balance restriction during runtime, we can pre-assign those node slots for work to maximize the location of the data because of idle plots often not assigned.

A Hadoop Task Schedule Load Manager is designed to balance loads across slave nodes and to put resources ratios between slave nodes closer together. Before we present slot planning, we start with two concepts:

- The workload balance shows the number of map slots available in an ideally balanced situation, as shown in Figure 6. The permissible idle map slots in the white region under the working load balance are illustrated.

- It affects the location of the data and the load balance. If a work has data from a nearby node, it happens. It happens. It happens. The slave node has some idle slots because of the load balancing issue, but the load manager can't. By improving the data location, this scenario compromises the load balance. The realization of the load balance adversely affects the placement of data.

6. **Optimization:** In practice, workloads between task trackers (i.e. machines) of running maps (or reducing) for a MapReduce cluster are generally different because of these facts.

- Many MapReduce clusters (that is, various computing power among machines) are heterogeneous in the real world.
- Maps frequently use different computing loads (i.e. execution time) and reduce tasks from different jobs due to various sizes and applications of input data.
- Map driving time or reduction tasks may still not be the same even for one job under a homogenous environment.

Step 1: Calculate load factor for all tasks divided by a cluster map slot capability as the total of upcoming map tasks and map features.
Step 2: Multiply the number of map slots on a tasks tracker to calculate the present maximum number of mapslots.
Step 3: Finally, the current permissible idle map slots for the task tracker can be calculated (or reduced), by removing the current map slots number from (or reduced) the current map slots maximum amount.

Let us suppose that J1 J2 J3 J4 is the order of priority assignment. Under the current load distribution, we see that no idle map slots are acceptable to every tracker with local J1 block data. This means that based on the timeframe, J1 is delayed within a timeframe to the extent that fairness does not affect the tasktracker with the heartbeat tracker. We can however see that the map is idle in every tasktracker. When Tasktracker 2 or 4 connects with the Jobtracker, we can proactively allow additional idle maps to satisfy both the data location and the fairness needs by relaxing the strict load balancing limit. On this basis, a scheduler named Slot Scheduling proposes that will proactively assign slots to those working with local mapping tasks so as to maximise and achieve fairness in the location of data.

For Slot Scheduling, two cases are available. The first case is a task tracker with additional idle map slots but no idle map slots. In case of assignments following fair share priorities, we can proactively assign additional map slots instead of skipping it via the default Hadoop scheduler with local map functions with block data for the task tracker.

The other case concerns DHSA. Where no idle map slots are available but some idle map slots are reduced, we may proactively borrow idle slots to restore the task for a heartbeat connected tasktracker to pending local map tasks and maximise the data location.

IV. PERFORMANCE EVALUATION

1. **Simulation Environment:** The proposed method is experimented in an AMD quad core CPU of 2.4 GHz, 8 GB RAM in windows operation system. The proposed framework is implemented on CloudSim, JDK7.0 and Eclipse. The scheduling process in cloud is carried out using CloudSim simulation platform.

2. **Dataset:** The size of task dataset in the experiment is set as 1000 and the experimental dataset is divided into 10 different groups with a data size of 100. The results are estimated by calculating the average values of each groups.

Table 1: Makespan

Task Type	Makespan			
	FRESH	SLO	MapReduce scheduling	Dynamic Slot scheduling
1	35.21	27.21	26.88	12.45
2	76.24	61.21	56.37	19.37
3	85.69	69.33	65.52	26.32
4	91.42	89.33	86.03	33.66

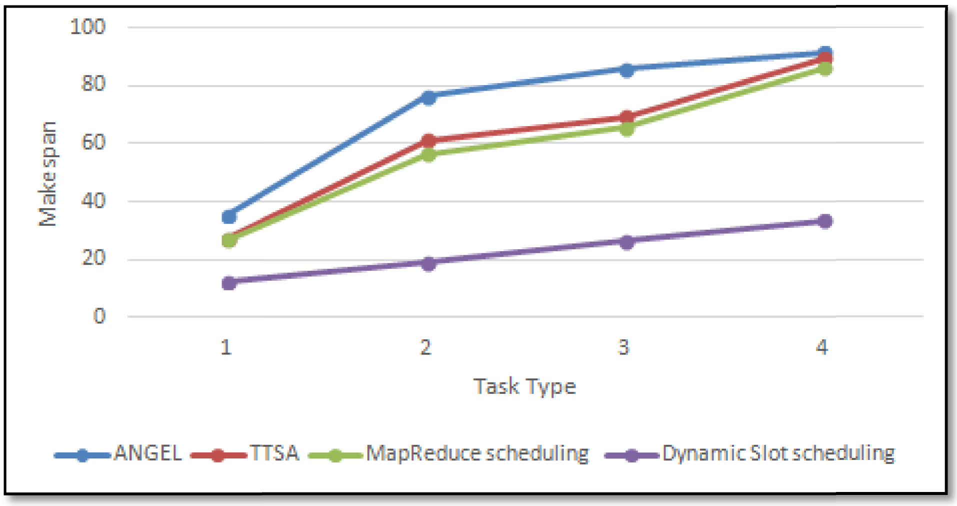

Figure 7: Make Span for four Different Tasks

The Figure 7 or Table 1 show the results of makespan metric required to evaluate the efficacy of proposed scheduling with existing scheduling techniques. The experiment uses four different task types. The result shows that proposed system exhibit a makespan of CGWO scheduling for Task 1 is 12.45, which is lesser than the makespan of MapReduce Scheduling and SLO and FRESH. The makespan of CGWO scheduling for

Task 2 is 19.37, which is lesser than the makespan of MapReduce Scheduling and SLO and FRESH. The makespan of CGWO scheduling for Task 3 is 26.32, which is lesser than the makespan of MapReduce Scheduling and existing FRESH and SLO. The makespan of CGWO scheduling for Task 4 is 33.66, which is lesser than the makespan of MapReduce Scheduling and existing FRESH and SLO.

The overall result shows that proposed system exhibit an average makespan of 22.95ms, which is lesser than the makespan of MapReduce Scheduling and SLO and FRESH. The existing FRESH and SLO method has increased makespan rate for each tasks than MapReduce scheduling and CGWO scheduling. As the task load increases from 1 to 4, the makespan rate increases in all the three methods. However, the makespan of CGWO scheduling for all task type is lesser than the MapReduce scheduling, and SLO and FRESH. The increased makespan is due to the fact that complexity of task increases from task 1 to task 4. The average makespan for the proposed system is 22.95, which is lesser than MapReduce scheduling, and existing FRESH and SLO.

Table 2: Average waiting time

Task Type	Average waiting time			
	FRESH	SLO	MapReduce scheduling	Dynamic Slot scheduling
1	50.33	15.34	13.28	11.26
2	62.35	23.25	20.13	19.62
3	79.14	40.23	35.21	23.64
4	85.21	62.31	59.65	32.61

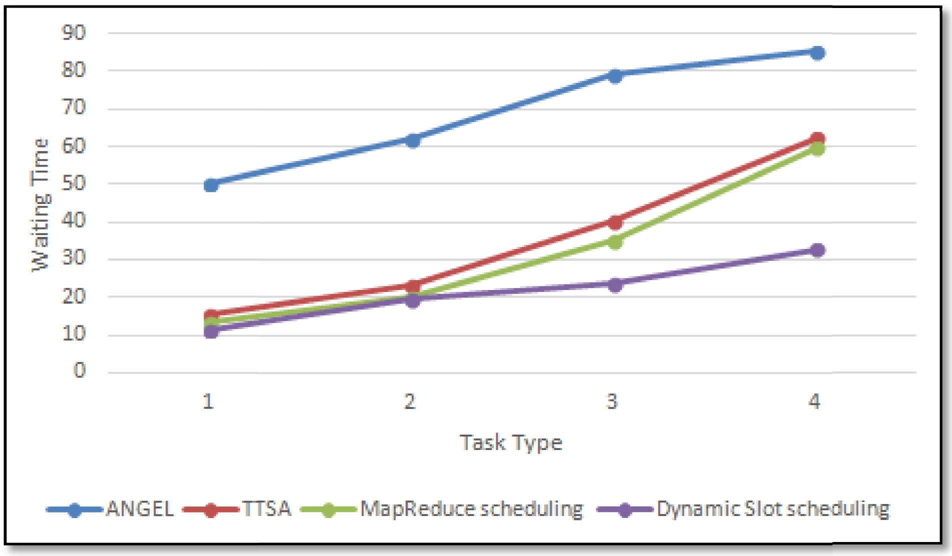

Figure 8: Average Waiting Time for Four Different Tasks

Similarly, the Figure 8 and Table 2 shows the results of average waiting time between the proposed and existing methods. The result shows that proposed system exhibit an average waiting time of CGWO scheduling for Task 1 is 11.26, which is lesser than the utilization rate of MapReduce scheduling and SLO and FRESH. The result shows that proposed system exhibit an average waiting time of CGWO scheduling for Task 2 is 19.62, which is lesser than the utilization rate of MapReduce scheduling and SLO and SLO. The result shows that proposed system exhibit an average waiting time of CGWO scheduling for Task 3 is 23.64, which is lesser than the utilization rate of MapReduce scheduling and SLO and FRESH. The result shows that proposed system exhibit an average waiting time of CGWO scheduling for Task 4 is 32.61, which is lesser than the utilization rate of MapReduce scheduling and SLO and SLO.

The Task 1 and 2 has reduced task complexity, hence there is a drop in the reduction time between the proposed and exiting methods than the Task 3 and 4. This infers that as the complexity of task increases, the average waiting time increases. The result shows that proposed method has reduced average waiting time for all the four task against MapReduce scheduling and SLO and FRESH scheduling methods. The result shows that proposed system exhibit an overall average waiting time for all these for task is 21.7825, which is lesser than the overall average waiting time of MapReduce scheduling and existing methods.

Table 3: Task Failure Rate

Task Type	Failure Rate			
	FRESH	SLO	MapReduce scheduling	Dynamic Slot scheduling
1	0.062	0.019	0.01	0
2	0.074	0.032	0.025	0
3	0.079	0.045	0.032	0.001
4	0.085	0.066	0.054	0.021

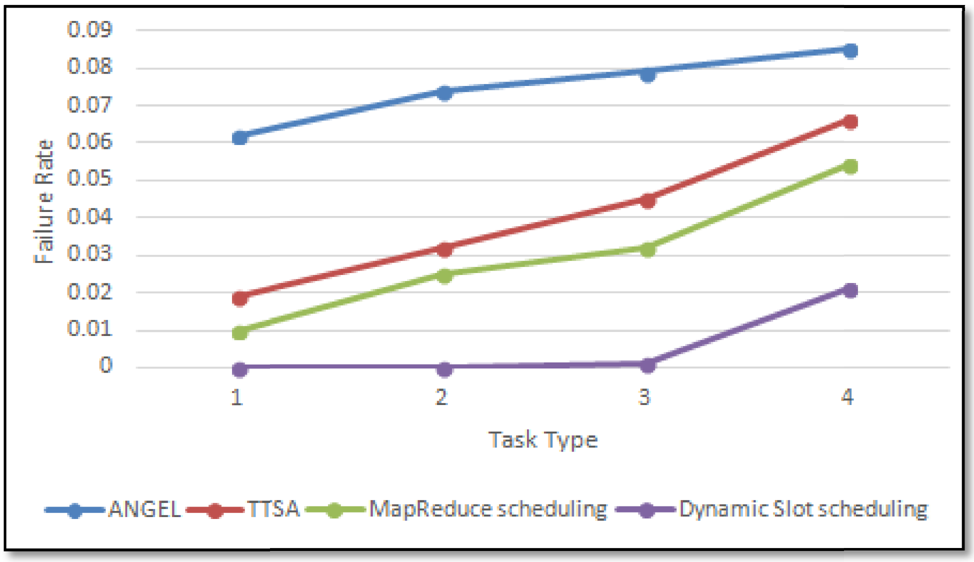

Figure 9: Failure Rate for Four Different Tasks

The failure rate of different VM task type is given in Table.3 or Figure 9. The result shows that proposed system exhibit a failure rate of CGWO scheduling for Task 1 is 0, which is lesser than the utilization rate of MapReduce scheduling and SLO and FRESH. The result shows that proposed system exhibit a failure rate of CGWO scheduling for Task 2 is 0, which is lesser than the utilization rate of MapReduce scheduling and SLO and FRESH. The result shows that proposed system exhibit a failure rate of CGWO scheduling for Task 3 is 0.001, which is lesser than the utilization rate of MapReduce scheduling and SLO and FRESH. The result shows that proposed system exhibit a failure rate of CGWO scheduling for Task 4 is 0.021, which is lesser than the utilization rate of MapReduce scheduling and SLO and FRESH. As the complexity of task increases from 1 to 4, the failure rates increase. However, the difference between them is significantly small.

The average failure rate for the proposed system is 0.0055, which is lesser than MapReduce scheduling and SLO and FRESH. It could be inferred that increasing failure rate is due to increase in complexity of task and vice versa, where most failure occurs, when the VMs work of Task type 4 and this is incredibly smaller in Task type 1.

Table 4: Utilization Rate

Task Type	Utilization Rate			
	FRESH	SLO	MapReduce scheduling	Dynamic Slot scheduling
1	24	6	5	4
2	26	19	17	15
3	32	27	25	19
4	41	38	35	25

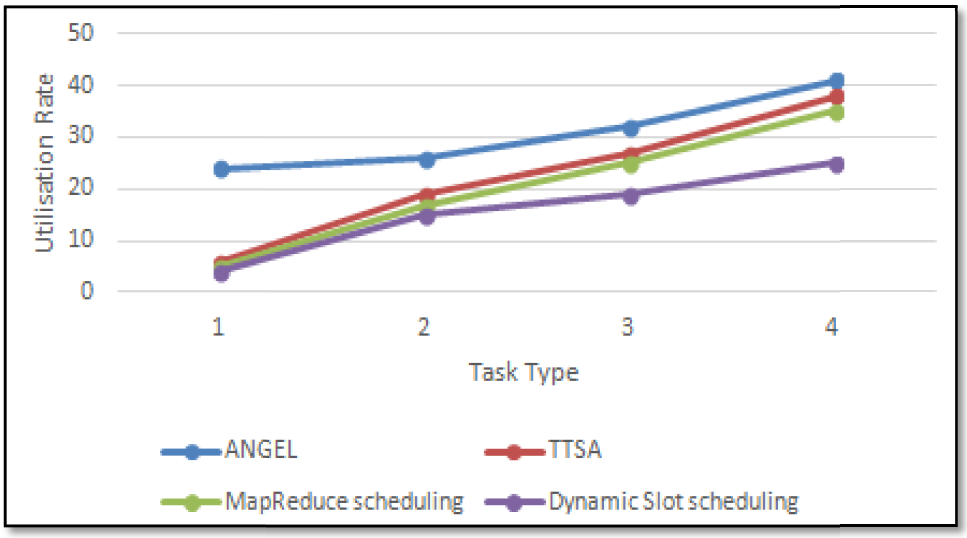

Figure 10: Utilization Rate for four Different Tasks

The Table 4 or Figure 10 shows the utilization rate between the proposed and existing methods for four different tasks. The result shows that proposed system exhibit a utilization rate of CGWO scheduling for Task 1 is 4, which is lesser than the utilization rate of MapReduce scheduling and SLO and FRESH. The result shows that proposed system exhibit a utilization rate of CGWO scheduling for Task 2 is 15, which is lesser than the utilization rate of MapReduce scheduling and SLO and FRESH. The result shows that proposed system exhibit a utilization rate of CGWO scheduling for Task 3 is 19, which is lesser than the utilization rate of MapReduce scheduling and SLO and FRESH. The result shows that proposed system exhibit a utilization rate of CGWO scheduling for Task 4 is 25, which is lesser than the utilization rate of MapReduce scheduling and SLO and FRESH.

It is inferred from the results that as the complexity of task increases, the rate of utilization increases for all the three methods. Further, it is seen that proposed method achieves lesser utilization rate than other methods.

Table 5: Communication Delay

Task Type	Communication Delay (ms)			
	FRESH	SLO	MapReduce scheduling	Dynamic Slot scheduling
1	2.79	2.16	2.13	0.99
2	6.05	4.86	4.47	1.54
3	6.80	5.50	5.20	2.09
4	7.26	7.09	6.83	2.67

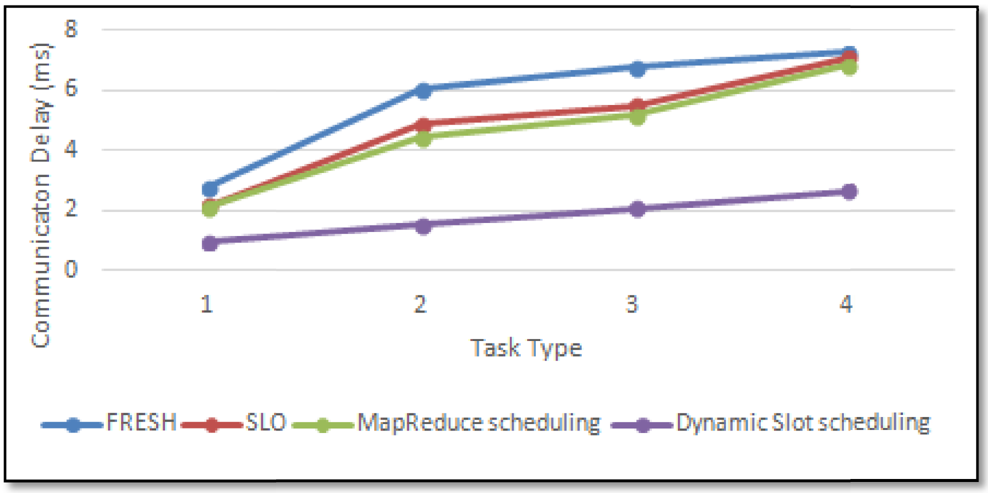

Figure 11: Communication Delay for Four Different Tasks

The Figure 11 or Table 5 show the results of Communication Delay metric required to evaluate the efficacy of proposed scheduling with existing scheduling techniques. The experiment uses four different task types. The result shows that proposed system exhibit a Communication Delay of DSS scheduling for Task 1 is 0.99, which is lesser than the Communication Delay of MapReduce Scheduling and SLO and FRESH. The Communication Delay of DSS scheduling for Task 2 is 1.54, which is lesser than the Communication Delay of MapReduce Scheduling and SLO and FRESH. The Communication Delay of DSS scheduling for Task 3 is 2.09, which is lesser than the Communication Delay of MapReduce Scheduling and existing FRESH and SLO. The Communication Delay of DSS scheduling for Task 4 is 2.67, which is lesser than the Communication Delay of MapReduce Scheduling and existing FRESH and SLO.

The existing FRESH and SLO method has increased Communication Delay rate for each tasks than MapReduce scheduling and DSS scheduling. As the task load increases from 1 to 4, the Communication Delay rate increases in all the three methods. However, the Communication Delay of DSS scheduling for all task type is lesser than the MapReduce scheduling, and SLO and FRESH. The increased Communication Delay is due to the fact that complexity of task increases from task 1 to task 4.

Table 6: Computational Delay

Task Type	Computational Delay			
	FRESH	SLO	MapReduce scheduling	Dynamic Slot scheduling
1	3.99	1.22	1.05	0.89
2	4.95	1.85	1.60	1.56
3	6.28	3.19	2.79	1.88
4	6.76	4.95	4.73	2.59

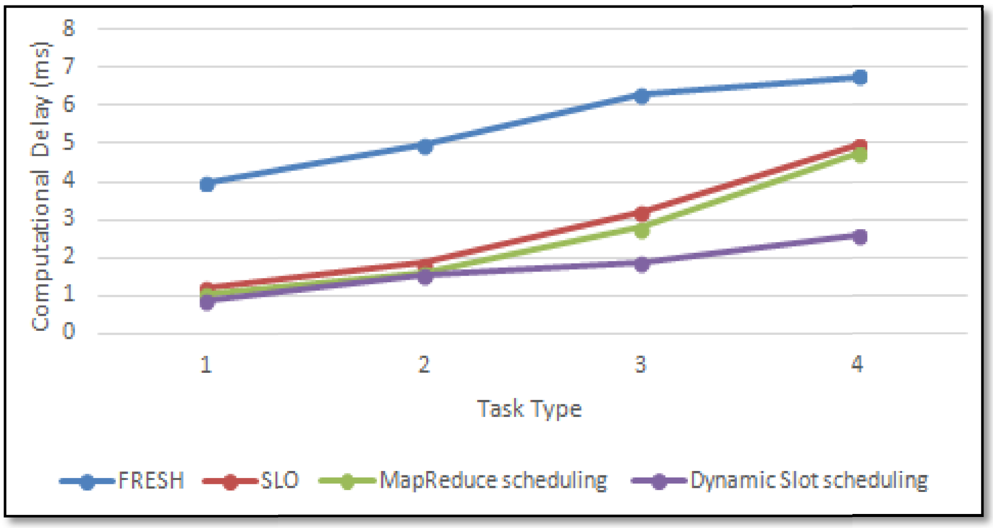

Figure 12: Computational Delay for four different tasks

Similarly, the Figure 12 and Table 6 show the results of Computational Delay between the proposed and existing methods. The result shows that proposed system exhibit an Computational Delay of DSS scheduling for Task 1 is 0.89, which is lesser than the System Throughput of MapReduce scheduling and SLO and FRESH. The result shows that proposed system exhibit an Computational Delay of DSS scheduling for Task 2 is 1.56, which is lesser than the System Throughput of MapReduce scheduling and SLO and FRESH. The result shows that proposed system exhibit an Computational Delay of DSS scheduling for Task 3 is 1.88, which is lesser than the System Throughput of MapReduce scheduling and SLO and FRESH. The result shows that proposed system exhibit an Computational Delay of DSS scheduling for Task 4 is 2.59, which is lesser than the System Throughput of MapReduce scheduling and SLO and SLO.

The Task 1 and 2 has reduced task complexity, hence there is a drop in the reduction time between the proposed and exiting methods than the Task 3 and 4. This infers that as the complexity of task increases, the Computational Delay increases. The result shows that proposed method has reduced Computational Delay for the entire four tasks against MapReduce scheduling and SLO and FRESH scheduling methods. The result shows that proposed system exhibit an overall Computational Delay, which is lesser than the overall Computational Delay of MapReduce scheduling and existing methods.

Table 7: AEED (AEED)

Task Type	AEED			
	FRESH	SLO	MapReduce scheduling	Dynamic Slot scheduling
1	0.78	0.24	0.13	0.02
2	0.93	0.40	0.32	0.03
3	1.00	0.57	0.40	0.08
4	1.07	0.83	0.68	0.26

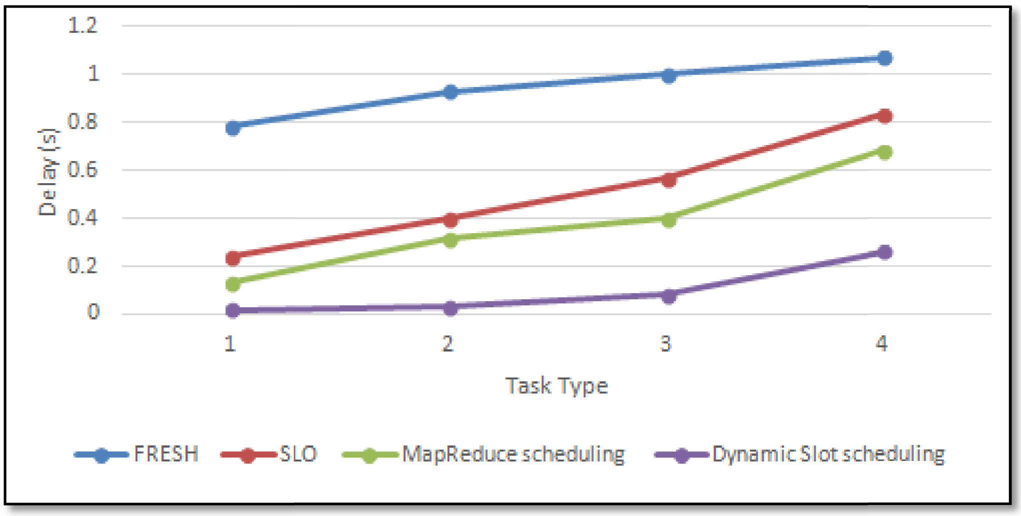

Figure 13: AEED for four different tasks

The AEED of different VM task type is given in Table 7 or Figure 13, The result shows that proposed system exhibit an AEED of DSS scheduling for Task 1 is 0.02, which is lesser than the System Throughput of MapReduce scheduling and SLO and FRESH. The result shows that proposed system exhibit an AEED of DSS scheduling for Task 2 is 0.03, which is lesser than the System Throughput of MapReduce scheduling and SLO and FRESH. The result shows that proposed system exhibit an AEED of DSS scheduling for Task 3 is 0.08, which is lesser than the System Throughput of MapReduce scheduling and SLO and FRESH. The result shows that proposed system exhibit an AEED of DSS scheduling for Task 4 is 0.26, which is lesser than the System Throughput of MapReduce scheduling and SLO and FRESH. As the complexity of task increases from 1 to 4, the AEEDs increases. However, the difference between them is significantly small.

The AEED is lesser than MapReduce scheduling and SLO and FRESH. It could be inferred that increasing AEED is due to increase in complexity of task and vice versa, where most failure occurs, when the VMs work of Task type 4 and this is incredibly smaller in Task type 1.

Table 8: System Throughput

Task Type	System Throughput (kbps)			
	FRESH	SLO	MapReduce scheduling	Dynamic Slot scheduling
1	5.04	6.30	7.56	30.24
2	18.90	21.42	23.94	32.76
3	23.94	31.50	34.02	40.32
4	31.50	44.10	47.88	51.66

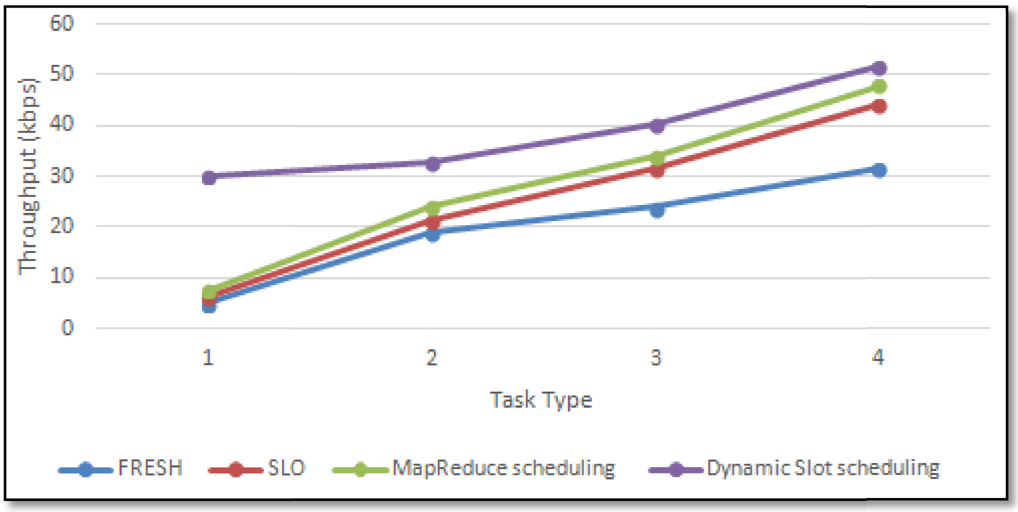

Figure 14: System Throughput for four different tasks

The Table 8 and Figure 14 show the System Throughput between the proposed and existing methods for four different tasks. The result shows that proposed system exhibit a System Throughput of DSS scheduling for Task 1 is 30.24, which is lesser than the System Throughput of MapReduce scheduling and SLO and FRESH. The result shows that proposed system exhibit a System Throughput of DSS scheduling for Task 2 is 32.76, which is lesser than the System Throughput of MapReduce scheduling and SLO and FRESH. The result shows that proposed system exhibit a System Throughput of DSS scheduling for Task 3 is 40.32, which is lesser than the System Throughput of MapReduce scheduling and SLO and FRESH. The result shows that proposed system exhibit a System Throughput of DSS scheduling for Task 4 is 51.66, which is lesser than the System Throughput of MapReduce scheduling and SLO and FRESH.

V. SUMMARY

In this work, task scheduling in cloud is optimized using CGWO framework that reduces the waiting time in queue. The proposed framework selects the idle VMs in a most efficient way using CGWO that reduces the total resource consumption and overall execution time. The framework sets certain criteria to ensure that all the task allocated is of independent one i.e. it does not depend on other tasks. The framework executes the task in an automated way during the scheduling the task for execution. The experimental results shows that proposed method is bandwidth friendly and it uses reduced time to execute a task.

VI. CONCLUSION

Task planning is carried out in this study using two different approaches, namely cluster self-adaptive groups forming data analysis mapping and Hadoop slot optimization scheduler. These two methods reduce the total queue time through several experiments. The former model uses various probability distribution formulas to complete the planning task and the latter model uses a meta-heuristic scheduler to improve cloud scheduling.

The research focuses on the algorithm planning through subtask resources that increase the performance of data analysis to support the client's task needs.

The focus is also on the planning of the work with a cloud computing model, total calculation task, speed and probability of data set distribution via cluster formation using MapReduce slot to supply the associated resources. The research focuses on the effective selection of idle VMs to reduce the overall runtime and resource consumption by using the Hadoop dynamic slot allocation. It considers that the whole task is independent. The tasks are carried out automatically during task planning, which reduces the programming load and runtime.

This technique effectively reduces every job's scheduling time, increases network performance, and significantly improves cloud computing model scalability. The results of the simulation show that the proposed Hadoop dynamic slot assignment and timing is effective in selecting VMs than other existing techniques.

REFERENCES

[1] Apache Hadoop. http://hadoop.apache.org.
[2] HadoopDistributedFileSystem,http://hadoop.apache.org/hdfs

[3] Neda Maleki, Amir Masoud Rahmani and Mauro Conti. (2021). SPO a secure and performance aware optimization for mapreduce scheduling. Journal of Network and Computer Applications, 176, 1-24.
[4] Frederik Wulf, Tobias Lindner, Markus Westner and Susanne Strahringer. (2021). IaaS, PaaS or SaaS the why of cloud computing delivery model selection vignettes on the post-adoption of cloud computing. 54th Hawaii International Conference on System Sciences, Grand Wailea, Maui, Hawaii.
[5] Neda Maleki, Amir Masoud Rahmani and Mauro Conti. (2021). SPO a secure and performance aware optimization for mapreduce scheduling. Journal of Network and Computer Applications, 176, 1-24.
[6] Swathi, M and Sreedhar, K. C. (2020). A cloud-based privacy-preserving e-healthcare system using particle swarm optimization; 1st Edition; Springer, Singapore, 133-143.
[7] Khushboo Kalia and Neeraj Gupta. (2020). Analysis of hadoopmapreduce scheduling in heterogeneous environment. Ain Shams Engineering Journal
[8] askaran Singh Thind and Rajbala Simon, "Implementation of Big Data in Cloud Computing with optimized Apache Hadoop", *2019 3rd International conference on Electronics Communication and Aerospace Technology (ICECA)*, pp. 997-1001, 2019.
[9] Selvitopi O, Demirci GV, Turk A, Aykanat C. Locality-aware and load-balanced static task scheduling for MapReduce. Future Gener Comput Syst. 2019;90:49–61.
[10] Vaishnnave, M. P, Suganya Devi, K and Srinivasan, P. (2019). A survey on cloud computing and hybrid cloud. International Journal of Applied Engineering Research. 14, 429-434.
[11] SawmyaNandar and ThaintZarliMyint. (2019). Hadoop mapreduce performance improvement in distributed system. IRE Journals, 3, 487-493.
[12] Glushkova, D.; Jovanovic, P.; Abelló, A. Mapreduce performance model for Hadoop 2.x. Inf. Syst. **2019**, 79, 32–43.
[13] Suyash Mishra, AnuranjanMisra and Suryakant yadav. (2018). Improving mapreduce performance using late scheduling in big data. 5th International Conference on “Computing for Sustainable Global Development, New Delhi
[14] Kh, E. Ali, Sh, A. Mazen and Hassanein, E. E. (2018). A proposed hybrid model for adopting cloud computing in e-government. Future Computing and Informatics Journal, 3, 286-295.
[15] Li J, Wang J, Lyu B, Wu J, Yang X. An improved algorithm for optimizing MapReduce based on locality and overlapping. Tsinghua Sci Technol. 2018;23(6):744–53.
[16] Sukhpreet Singh, Amrik Singh and Navdeep Kaur. (2017). A study on use of big data in cloud”, International Conference on Recent Innovations in Science, Agriculture, Engineering and management, Bathinda, Punjab (India).
[17] Yanfei Guo, Jia Rao, Changjun Jiang and Xiaobo Zhou. (2017). Moving hadoop into the cloud with flexible slot management and speculative execution. IEEE Transactions on Parallel and Distributed Systems, 28, 798-812
[18] Dazhao Cheng, Xiaobo Zhou, Palden Lama, Jun Wu, Changjun Jiang, Cross-platform resource scheduling for spark and MapReduce on YARN, IEEE Trans. Comput. PP (99) (2017), http://dx.doi.org/10.1109/TC.2017.2669964, 1–1.
[19] Xianjin Luo and Chenggang Zhen. (2016). Analysis and optimization of the hadoop speculative execution mechanism. International Journal of Simulation Systems, Science and Technology, 17, 1-4.