IJETST- Vol.||02||Issue||01||Pages 1711-1725||January||ISSN 2348-9480 2015

Open access Journal International Journal of Emerging Trends in Science and Technology

Mobile Agent Based MapReduce Framework for Big Data Processing

Author
Sonu Yadav
M.Tech Stholar, Computer Science & Engineering, RPSGOI Mahendragarh,
Maharshi Dayanand University, Rohtak-124001, Haryana, India
Email: *sonu25.90@gmail.com*

Abstract
Distributed computing accomplished broad appropriation because of consequently parallelizing and transparently executing tasks in distributed environments. Straggling tasks is an essential test confronted by all Big Data Processing Frameworks, for example, Mapreduce[3], Dryad[4], Spark[5]. Stragglers are the assignments that run much slower than different tasks and since a job completes just when it's last undertaking completions, stragglers postponement work fruition. The literature reviews stragglers recognition and rescheduling systems proposed so far and brings up their strengths and shortcomings. This thesis additionally displays wise attributes and impediments of the existing state- of-the- craftsmanship calculations to take care of the issue of stragglers relief. This thesis presents a systematic and organized study of community detection techniques. The literature survey shows that most of the algorithms fail to efficiently reschedule the stragglers. Innocently one may anticipate that straggler taking care of will be a simple assignment, doubling tasks that are sufficiently slower. Actually it is a complex issue for a few reasons. In the first place, Speculative assignments are not free they seek certain assets, for example, system with other running tasks. Second, picking the node to run speculative task on is as significant as picking the task. Third, in Heterogeneous environment it may be challenging to recognize nods that are marginally slower than the mean and stragglers. At long last, Stragglers ought to be recognized as right on time as could reasonably be expected. The proposed framework uses mobile agent approach for rescheduling because the agent can start the execution at the other place from the very same place they left in the earlier machine. The implementation and results shows the proposed work is efficient and improves the overall performance of a big data processing framework.

Keywords: *MR, MAS, JADE, LATE, CORBA*

1. Introduction

Big Data is considered to be a data collection that has grown so large it can't be effectively or affordably managed (or exploited) using conventional data management tools: e.g., classic relational database management systems (RDBMS) [11] or conventional search engines [12], depending on the task at hand.

1.1 The 3 Vs that define Big Data are Variety, Velocity and Volume.

1.1.1 Variety: Data can be stored in multiple formats. For example database, excel, csv, access or for the matter of the fact, it can be stored in a simple text file. Sometimes the data is not even in the traditional format as we assume, it may be in the form of video, SMS, pdf or something we might have not thought about it. It is the need of the organization to arrange it and make it meaningful. It will be easy to do so if we have data in the same format, however it is not the case most of the time. The real world has data in many different formats and that is the challenge we need to overcome with the *Big Data*. This variety of the data represents Big Data.

1.1.2 Velocity: The data growth and social media explosion have changed how we look at the data. There was a time when we used to believe that data of yesterday is recent. The matter of the fact newspapers is still following that logic. Today, people reply on social media to update them with the latest happening. On social media sometimes a few seconds old messages (a tweet, status updates etc.) is not something interests users. They often discard old messages and pay attention to recent updates. The data movement is now almost real time and the update window has reduced to fractions of the seconds. This high velocity data represent Big Data.

1.1.3 Volume: The exponential growth in the data storage as the data is now more than text data. We can find data in the format of videos, music and large images on our social media channels. It is very common to have Terabytes and Petabytes of the storage system for enterprises. As the database grows the applications and architecture built to support the data needs to be reevaluated quite often. Sometimes the same data is re-evaluated with multiple angles and even though the original data is the same the new found intelligence creates explosion of the data. The big volume indeed represents Big Data.

1.1.4 Big Data Applications include

1. Faceted Search at Scale Faceted search is the process of iteratively refining a search request by selecting (or excluding) clusters or categories of results.
2. Multimedia Search Multimedia Content is the fastest growing type of user generated content, with millions of photos, audio files and videos uploaded to the web on daily basis.
3. Sentimental Analysis uses semantic technologies to automatically discover, extract and summarize the emotions and attitudes expressed in unstructured content.
4. Database Enrichment is done after collecting the data. This collected data is then analyzed and organized so that we can further use it to enhance and contextualize existing structured data resources like databases and data warehouses.

1.2 MapReduce Paradigm

MapReduce is a programming framework popularized by Google and used to simplify data processing across massive data sets. As people rapidly increase their online activity and digital footprint, organizations are finding it vital to quickly analyze the huge amounts of data their customers and audiences generate to better understand and serve them. MapReduce is the tool that is helping those kinds of organizations. This is a methodology need to handle extensive scale web search applications. This methodology is utilized within creating machine learning, data mining and search applications in data centers. The point of interest is that it permits programmers to extract from the issues of booking, parallelization, parceling, replication and concentrates on creating their application.

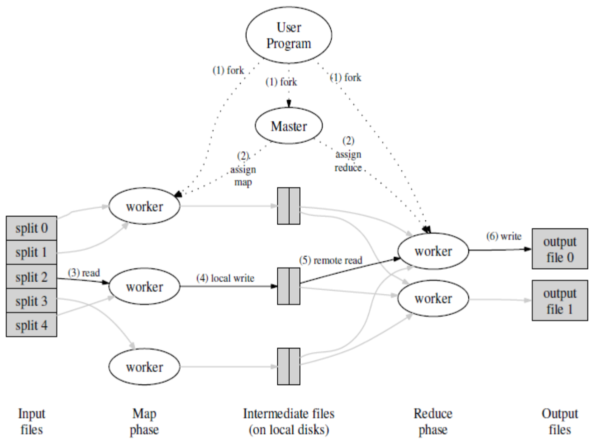


Figure 1: MapReduce Framework

The MapReduce library in the user program first splits the input files into M pieces of typically 16 megabytes to 64 megabytes (MB) per piece (controllable by the user via an optional parameter). It then starts up many copies of the program on a cluster of machines.

One of the copies of the program is special, the master. The rest are workers that are assigned work by the master. There are M map tasks and R reduces tasks to assign. The master picks idle workers and assigns each one a map task or a reduce task.

A worker who is assigned a map task reads the contents of the corresponding input split. It parses key/value pairs out of the input data and passes each pair to the user-defined *Map* function. The intermediate key/value pairs produced by the *Map* function are buffered in memory.

Periodically, the buffered pairs are written to local disk, partitioned into R regions by the partitioning function. The locations of these buffered pairs on the local disk are passed back to the master, who is responsible for forwarding these locations to the reduce workers.

When a reduce worker is notified by the master about these locations, it uses remote procedure calls to read the buffered data from the local disks of the map workers. When a reduce worker has read all intermediate data, it sorts it by the intermediate keys so that all occurrences of the same key are grouped together. The sorting is needed because typically many different keys map to the same reduce task. If the amount of intermediate data is too large to fit in memory, an external sort is used.

The reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user's Reduce function. The output of the Reduce function is appended to a final output file for this reduced partition.

When all map tasks and reduce tasks have been completed, the master wakes up the user program. At this point, the MapReduce call in the user program returns back to the user code. After successful completion, the output of the mapreduce execution is available in the R output files (one per reduce task, with file names as specified by the user). Typically, users do not need to combine these R output files into one file. They often pass these files as input to another MapReduce call, or use them from another distributed application that is able to deal with input that is partitioned into multiple files.

1.2.1 Responsibilities of Execution Framework

The developer submits the job to the submission node of a cluster (in Hadoop, this is called the job tracker) and execution framework (sometimes called the "runtime") takes care of everything else: it transparently handles all other aspects of distributed code execution, on clusters ranging from a single node to a few thousand nodes. Specific responsibilities include:

Scheduling:

Each MapReduce job is divided into smaller units called tasks. It is not uncommon for MapReduce jobs to have thousands of individual tasks that need to be assigned to nodes in the cluster. In large jobs, the total number of tasks may exceed the number of tasks that can be run on the cluster concurrently, making it necessary for the scheduler to maintain some sort of a task queue and to track the progress of running tasks so that waiting tasks can be assigned to nodes as they become available.

Data/Code Co-location: The phrase data distribution is misleading, since one of the key ideas behind MapReduce is to move the code, not the data. However, the more general point remains in order for computation to occur; we need to somehow feed data to the code. In MapReduce, this issue is inexplicably intertwined with scheduling and relies heavily on the design of the underlying distributed file system. To achieve data locality, the scheduler starts tasks on the node that holds a particular block of data (i.e., on its local drive) needed by the task. This has the effect of moving code to the data. If this is not possible (e.g., a node is already running too many tasks), new tasks will be started elsewhere, and the necessary data will be streamed over the network.

Synchronization: In general, synchronization refers to the mechanisms by which multiple concurrently running processes "join up", for example, to share intermediate results or otherwise exchange state information. In MapReduce, synchronization is accomplished by a barrier between the maps and reduces phases of processing. Intermediate key-value pairs must be grouped by key, which is accomplished by a large distributed sort involving all the nodes that executed map tasks and all the nodes that will execute reduce tasks. This necessarily involves copying intermediate data over the network, and therefore the process is commonly known as "shuffle and sort". A MapReduce job with m mappers and r reducers involves up to m * r distinct copy operations, since each mapper may have intermediate output going to every reducer. Note that the reduce computation cannot start until all the mappers have finished emitting key-value pairs and all intermediate key-value pairs have been shuffled and sorted, since the execution framework cannot otherwise guarantee that all values associated with the same key have been gathered. This is an important departure from functional programming: in a fold operation, the aggregation function g is a function of the intermediate value and the next item in the list which means that values can be lazily generated and aggregation can begin as soon as values are available. In contrast, the reducer in MapReduce receives all values associated with the same key at once.

Error and Fault Handling: The MapReduce execution framework must accomplish all the tasks above in an environment where errors and faults are the norm, not the exception. Since MapReduce was explicitly designed around low-end commodity servers, the runtime must be especially resilient. In large clusters, disk failures are common [34] and RAM experiences more errors than one might expect [37]. Datacenters suffer from both planned outages (e.g., system maintenance and hardware upgrades) and unexpected outages (e.g., power failure, connectivity loss, etc.).

1.3 Mobile Agents Paradigm

Agents are software entities that have some kind of autonomy and certain ‘intelligence’. An agent is often assumed to represent another entity, such as a human or an organization on whose behalf it is acting. They are given some goals and they try to achieve these goals according to their intelligence. The basic dictionary definition of agent is one who acts [20]. In order to achieve these goals they communicate and interact with other agents, they exchange information and take back the results to the user. The software agents behave in the same manner too.

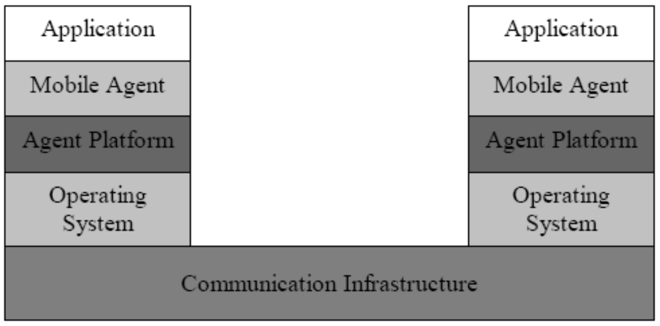


Figure 2: Mobile agent in network system

1.3.1 Mobile Agents’Applications:

Computation bundles - converts computational client/server round trips to relocatable data bundles, reducing network load.

Parallel processing -asynchronous execution on multiple heterogeneous network hosts

Dynamic adaptation - actions are dependent on the state of the host environment

Tolerant to network faults - able to operate without an active connection between client and server

Flexible maintenance - to change an agent's actions, only the source (rather than the computation hosts) must be updated

1.3.2 Stragglers

Stragglers [10], tasks running slowly compared to their peers, also impacts on mapreduce performance. Stragglers need to be speculated [37] on faster machines if overall performance is to be improved as even a single task running on a slower machine can delay the further execution of already finished tasks.

Stragglers’ problem: Due to unavailable input, tasks have to be recomputed if we want to proceed further otherwise we have to wait for the time, which the task takes to complete.

There are numerous reasons [23] behind the task to take longer time, for example, flawed machines, heterogeneity among hardware, measure of data to process, system blockage and contention for the existing assets. Be that as it may if one task runs slower on a given machine it is not important for the entire present and future task to run slower on that specific machine. Likewise it is not important for a task to be slower all around its execution.

But, if one task runs slower on a given machine, it is not necessary for all the current and future tasks to run slower on that host. Also, it is not necessary for a task to be slower throughout its execution. e.g., it is possible that a task is slower because there are other processes contending for the resources, which can be very transient. Therefore, we introduce another term, straggler effect, to denote the transient behavior of the straggler. Previous efforts mainly provided speculation mechanisms at the application layer by modifying the Hadoop framework. However, stragglers are not entirely caused because of the faults at the application [38], but they may be caused even because of the other processes running on the same host. e.g., the disk access is delayed because there is one more process that is performing disk operations and hence contending for the disk. Therefore, even if the disk is not faulty, because of the other IO bound processes, a straggler effect can be present. Similarly, the Hadoop processes have to contend for the CPU in case there are other VMs running on the same host. Due to this fact, no mechanism can optimally resolve the straggler at the application, but the straggler effect can be effectively resolved at the operating system level, where the information about all the processes running on the host can be obtained and used.

2. COMPARISON BETWEEN VARIOUS ALGORITHM

The Problem of Stragglers has accepted extensive consideration recently with numerous stragglers moderating methods being created. These strategies could be comprehensively considered Blacklisting and Speculative Execution. Boycotting distinguishes machines in terrible health and abstains from scheduling tasks on these machines. Nonetheless, Stragglers happen on non boycotted machines, regularly because of characteristically complex reasons like I/O contention, obstruction by occasional support operations and background services and network behaviors.

2.1 HADOOP NATIVE SCHEDULER

Here, the mechanism used by Hadoop to distribute work across a cluster is described along with its method to detect straggler and then speculating them. Assumptions made by the scheduler have been identified which hurt its performance under normal load. These motivate our straggler detecting and mitigating algorithms scheduler, which can improve Hadoop performance.

Hadoop implementation of MapReduce closely resembles Google's [1]. There is a single master managing a number of slaves. The input file, which resides on a distributed file system throughout the cluster, is split into even-sized chunks replicated for fault-tolerance. Hadoop divides each MapReduce job into a set of tasks. Each chunk of input is first processed by a map task, which outputs a list of key-value pairs generated by a user defined map function. Map outputs are split into buckets based on key. When all maps have finished, reduce tasks apply a reduce function to the list of map outputs with each key.

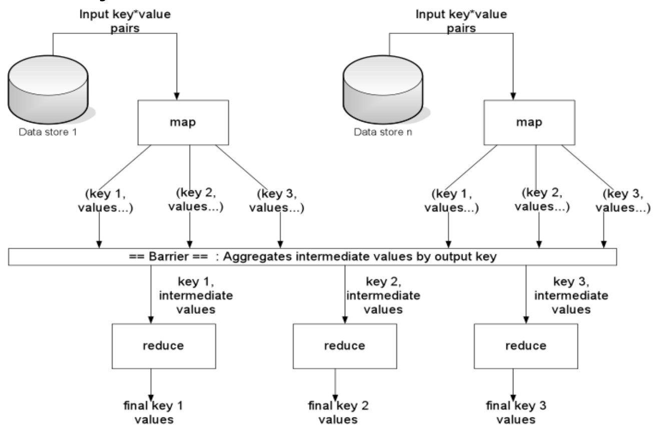


Figure 3: Hadoop Map Reduce

Figure 3, illustrates a MapReduce computation. Hadoop runs several maps and reduces concurrently on each slave – two of each by default – to overlap computation and I/O. Each slave tells the master when it has empty task slots. The scheduler then assigns it tasks. The goal of speculative execution is to minimize a job's response time. Response time is most important for short jobs where a user wants an answer quickly, such as queries on log data for debugging, monitoring and business intelligence. Short jobs are a major use case for MapReduce.

2.1.1 Assumptions in Hadoop Native Scheduler: Hadoop scheduler makes several implicit assumptions [6] which are:

1. Nodes can perform work at roughly the same rate.
2. Tasks progress at a constant rate throughout time.
3. There is no cost to launching a speculative task on a node that would otherwise have an idle slot.
4. A task's progress score is representative of fraction of its total work that it has done. Specifically, in a reduce task, the copy, sort and reduce phases each take about 1/3 of the total time.
5. Tasks tend to finish in waves, so a task with a low progress score is likely a straggler.
6. Tasks in the same category (map or reduce) require roughly the same amount of work.

2.1.2 How the Assumptions Break Down: These assumptions [6] break down in current sort of parallel processing network as follows:

1. Heterogeneity: Assumptions (1) and (2) are about homogeneity among nodes in the network. Hadoop assumes that any detectably slow node is faulty. However, nodes can be slow for other reasons. In a non-virtualized data center, there may be multiple generations of hardware. In a virtualized data center where multiple virtual machines run on each physical host, such as Amazon EC2, co-location of VMs may cause heterogeneity. Heterogeneity seriously impacts Hadoop scheduler. Because the scheduler uses a fixed threshold for selecting tasks to speculate, too many speculative tasks may be launched; taking away resources from useful tasks (assumption 3 is also untrue). Also, because the scheduler ranks candidates by locality, the wrong tasks may be chosen for speculation first.

2. Other Assumptions: Assumptions (3), (4) and (5) stated above are broken on both homogeneous and heterogeneous clusters, and can lead to a variety of failure modes.

Assumption (3) breaks down when resources are shared. For example, the network is a bottleneck shared resource in large MapReduce jobs. Also, speculative tasks may compete for disk I/O in I/O-bound jobs. Finally, when multiple jobs are submitted, needless speculation reduces throughput without improving response time by occupying nodes that could be running the next job.

Assumption (4), that a task's progress score is approximately equal to its percent completion, can cause incorrect speculation of reducers. In a typical MapReduce job, the copy phase of reduce tasks is the slowest, because it involves all-pairs communication over the network. Tasks quickly complete the other two phases once they have all map outputs. However, the copy phase counts for only 1/3 of the progress score. Thus, soon after the first few reducers in a job finish the copy phase, their progress goes from 1/3 to 1, greatly increasing the average progress.

Assumption (5), that progress score is a good proxy for progress rate because tasks begin at roughly the same time, can also be wrong. The number of reducers in a Hadoop job is typically chosen small enough so that they can all start running right away, to copy data while maps run. However, there are potentially tens of mappers per node, one for each data chunk. The mappers tend to run in waves. Even in a homogeneous environment, these waves get more spread out over time due to variance adding up, so in a long enough job, tasks from different generations will be running concurrently. In this case, Hadoop will speculatively execute new, fast tasks instead of old, slow tasks that have more total progress.

2.2 LONGEST APPROXIMATE TIME TO END (LATE) SCHEDULER

Progress Score is ascertained as in Hadoop local scheduler. Progress rate is then calculated as advancement score/T where T is the time for which the task has been running. Time to finish is then approximated as (1-Progress Score)/Progress Rate.Tasks with advancement rates beneath an edge of 25 percentile of all tasks are acknowledged to be stragglers.LATE stays informed regarding moderate nodes in the system and does not run speculative duplicates on those nodes.LATE additionally utilizes a cap on the amount of speculative task that can run at once, to handle the way that speculative tasks cost assets.

2.2.1 LATE Algorithm:

A node asks for a new task and there are fewer than Speculative Cap speculative tasks running: Ignore the request if the node's total progress is below Slow Node Threshold. Rank currently running tasks that are not currently being speculated by estimated time left. Launch a copy of the highest-ranked task with progress rate below Slow Task Threshold. Like Hadoop scheduler, it also waits until a task has run for 1 minute before evaluating it for speculation. In practice, it has found that a good choice for the three parameters to LATE are to set the Speculative Cap to 10% of available task slots and set the Slow Node Threshold and slow Task Threshold to the 25th percentile of node progress and task progress rates respectively.

Advantages: LATE enjoys following advantages:

1. It is robust to node heterogeneity, in light of the fact that it will re-propel just the slowest tasks and just a little number of tasks.
2. LATE takes into account node heterogeneity while deciding where to speculate tasks.
3. Likewise, by keeping tabs on assessed time left instead of advancement rate, LATE hypothetically executes just assignments that will enhance job response time, as opposed to any slow tasks.

Disadvantages: LATE comes up with some demerits too, which are:

1. A bigger undertaking will have a tendency to take more of a chance than the rests to process, in this way it is conceivable to be tagged as a candidate to be speculated resulting in wasted assets.
2. As the end time for an assignment is ascertained utilizing the averaged out progress rate of out advancement rate against the current advancement rate, the end time anticipated is prone to be mistaken.
3. Starting assessment time needed by the LATE scheduler is high (1 minute) before an undertaking could be stamped as straggler.
4. LATE basically prompts longer reaction time. Since no clarification for the moderate nature of the accepted stragglers is looked for, the straggler determination might be inaccurate.

2.3 REINING IN OUTLIERS USING "MANTRI"

Mantri is a system that monitors tasks and mitigate outliers using cause and resource aware techniques. Mantri's strategies include restarting outliers, network-aware placement of tasks and protecting outputs of valuable tasks. Using real-time progress reports, Mantri detects and acts on outliers early in their lifetime. A task that has to run for long because it has more work to do will not be restarted; if it lags due to reading data over a low-bandwidth path, it will be restarted only if a more advantageous network location becomes available. Early action on outliers frees up resources that could be used for pending tasks, doing so is nontrivial.

2.3.1 Mantri's Restart Algorithm

```
1: let $\Delta$ = period of progress reports
2: let c = number of copies of a task
3: periodically, for each running task, kill all but the fastest $\alpha$ copies after$\Delta$ time has passed since begin
4: **while** slots are available **do**
5:**if** tasks are waiting for slots **then**
6: kill, restart task if $t_{rem} > E(t_{rem}) + \Delta$, stop at $\gamma$ restarts
7: duplicate if $P((t_{rem} > t_{new})*(c+1/c)) > \delta$
8: start the waiting task that has the largest data to read
9: **else** all tasks have begun
10: duplicate iff $E(t_{new} - t_{rem}) > \rho\Delta$
11: **end if**
12: **end while**
```

Mantri's restart algorithm is independent of the values for its parameters. Setting γ to a larger and ρ, δ to a smaller value trades off the risk of wasteful restarts for getting larger speedup. The default values that are specified here err on the side of caution. To not thrash on inaccurate estimates, Mantri kills a task no more than $\gamma = 3$ times. By scheduling duplicates conservatively and pruning aggressively, Mantri has a high success rate of its restarts. As a result, it reduces completion time and conserves resources.

2.4 MONTOOL: FINDING STRAGGLERS IN HADOOP

MonTool takes an alternative approach to determine the relative ordering of system calls in order to make a relation of system calls with stragglers. It tracks disk and network system calls for this analysis. MonTool is designed on the underlying assumption that tasks in same category (Map/Reduce) make similar system calls in an ordered sequence. A straggler would show a slower system calls access pattern. MonTool runs a daemon on each slave node

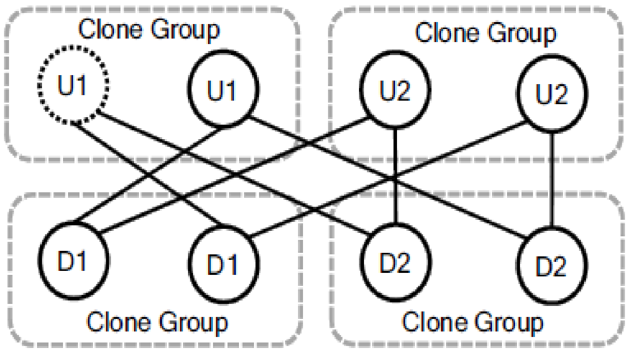


Figure 4: Contention Avoidance Cloning

which periodically sends monitoring information to the master node. Further, the master can query slaves to understand the causes for the task delays.

2.4.1 Working: Mon Tool gathers information about the tasks by tracing system calls and analyzing them. With this information Mon Tool finds the stragglers as well as their causes. For this Mon Tool performs two important functions which are:

1. Effective Gathering of System Calls: Mon Tool used System Tap to monitor the system calls. System Tap uses the hooks provided by the OS, where user code can be executed. System Tap generates comparatively lesser amount of overhead data while gathering the system calls and efficiently differentiates the network and disk read/writes.

2. Detecting the stragglers based on the pattern of system calls made by different machines: This task was based on the correlation of various patterns produced by various machines carrying out similar tasks.

2.4.2 Limitations: It accepts all maps or diminishes tasks work upon similar measured workloads and access information in a similar pattern. Be that as it reduces this assumption reduce tasks as information size read by diminish tasks may be distinctive for each task. Associating system calls can't be attained without any data about the keys and the example of the keys is regularly not accessible in Hadoop.

2.5 ATTACK OF THE CLONES: DOLLY

Current mitigation techniques, all involve an element of waiting and speculation of stragglers whenever detected. Dolly instead propose full cloning of small jobs, avoiding waiting and speculation altogether. Cloning of small jobs only marginally increases resource utilization because workloads show that while the majority of jobs are small, they only consume a small fraction of the resources. Dolly methodology manages stragglers in proactive way. As opposed to holding up and attempting to predict stragglers, it take speculative execution to its extreme and launches different clones of each task of a job and just utilize the result of the clone that completes first. This introduces some challenges like extra handful of resources and then accessing the data from the fastest clone, i.e. which finishes first in the group. Dolly defines new approaches for intermediate data access.

2.5.1 Intermediate Data Access: Avoiding Contention: Dolly defines its approaches for mitigating contention while accessing intermediate data from various map processes finishing simultaneously.

1. CAC Contention Avoidance Cloning: Here as soon as an upstream task clone finishes, its output is sent to exactly one downstream task clone per clone group.

Ψ (n,c,d)=Probability[n upstream tasks of c clones with >= d clones per group.

p is the probability of a task straggling.

$$\Psi\ (n,c,d) = \left(\sum_{i=0}^{c-d} \binom{c}{i} p^{i} (1-p)^{c-i} \right)^{n}$$

Contention Avoidance Cloning Dolly defined probability for job straggling with CAC as

$$P= 1-\sum^{c}_{d=1}[\psi(n,c,d)-\psi(n,c,d-1)]*(1-p^{d})^{n}$$

1. CC Contention Cloning: As soon as an upstream task clone finishes, all the downstream tasks read the output of the upstream clone, alleviating the problem of contention.

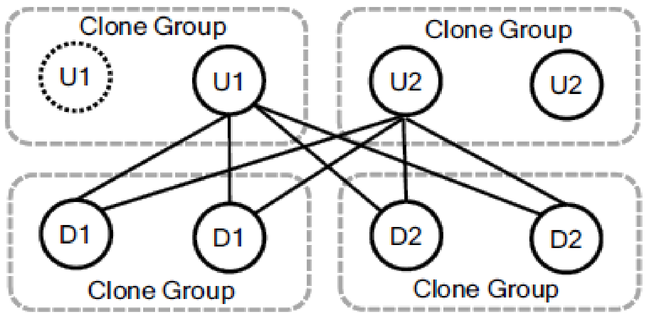


Figure 5: Contention Cloning [7]

Dolly defined probability for job straggling with CC as

$$P= 1-\sum^{c}_{d=1}[\psi (n,c,1)]*(1-p^{d})^{n}$$

Every downstream clone waits for a small window of time (ώ) to see if it can get an exclusive copy of the intermediate data. The wait time of ώ allows for normal variations among upstream clones. If the downstream clones does not get its exclusive copy even after waiting for ώ, it reads with contention from one of the finished upstream clones output.

2.6 ISSUES WITH EXISTING ALGORITHMS

There exist a few algorithms for straggler detection and mitigating their effects. All of them in one way or the other launches speculative copies of the tasks which are running slower as compared to the others. These speculative launches, however, reduces the time for overall execution but uses comparatively more resources. During the review of the current state of art algorithms it was found that the major issue with the existing algorithms was their inefficiency in launching the same task which is staggering on one machine to the other machine and resuming it there from the very same place from where it had stopped execution on the previous machine. So there is a sheer need for algorithms which can link the already mapped and reduced portion to the portion of the task which is to be executed next on the other machine. Another issue was the problem of homogeneity. None of them have employed mobile agents for addressing heterogeneity of the machines in the network. Mobile agents addresses this issue efficiently as they can resume their execution from the same place they had left but also don't mind the heterogeneity in the network.

3. PROPOSED WORK AND IMPLEMENTATION

The previous chapter described various techniques which are used by other researchers to solve the problem of straggler detection and mitigation in parallel data processing network; the chapter reviewed the broad literature on stragglers and shows important insights in the domain. It is observed that there are a lot of techniques are proposed for the detection of stragglers in parallel processing data networks, but none of them used mobile agents. A mobile agent is a set of code and data which can execute the code with the data as parameter in agent platform. Due to stragglers a part of resources gets wasted because we have multiple speculative copies of a single task running at the same time but we need only one for one final execution. So if we can limit the number of speculative execution to a single copy at any time we can have a hundred percent resource utilization for the useful because we run only one instance for any file at a time.

3.1 ALGORITHMS FOR PROPOSED SOLUTION

The proposed solution ABMR algorithm uses one mobile agent for each split it gets from the user and then that mobile agent moves from one location to another based on how the scheduler schedules it based on the performance score the agent sent to the scheduler. The scheduler maintains two arrays one for slower machines and one for faster machines. The scheduler reschedules the agents running on slower machines to the faster machines one at a time. The scheduler finally recollects all the results after each agent has done its work and then presents it to the original user of the application.

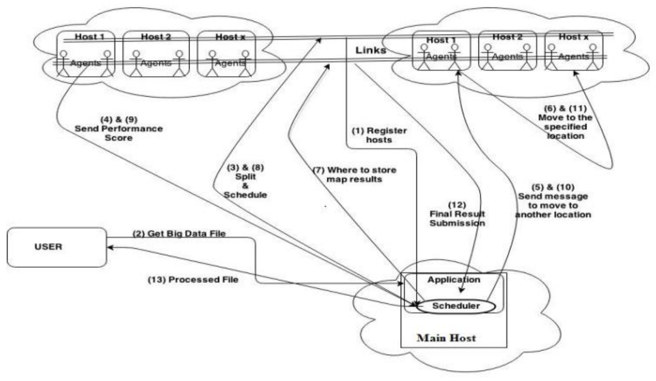

Host 1
Host 2
Host x
Agents
Links
(1) Register hosts
(2) Get Big Data File
(3) & (8) Split & Schedule
(4) & (9) Send Performance Score
(5) & (10) Send message to move to another location
(6) & (11) Move to the specified location
(7) Where to store map results
(12) Final Result Submission
(13) Processed File
USER
Application
Scheduler
Main Host

Figure 6: Information Flow Diagram

3.1.1 Algorithm for Scheduler

1. All hosts in the network which want to take part in big data processing register themselves with their IP Addresses.

These IP addresses are used to create agent containers in the main program.

2. A Scheduler agent is created in by the main application which takes file name as argument and finds all the agent containers running on the home platform.

3. Scheduler agent creates the mobile agents after splitting the file and assigns a file split to each of them.

Each agent sends its performance score to the scheduler after every Ts seconds.

score_agent[i] =Total executed/Total to be executed*Ts

4. Meanwhile, when the agent moves from one location to another, it saves its current state so that they can start from the very same place where they stopped the execution.

5. After performing all the operations, the agent submits the result back to the scheduler.

6. The scheduler forwards the result to the user after accumulation of results from all the reducers.

Algorithm for Scheduler

MainScheduler(file[],containers[])

```
1. ratio=file.length/container.length
2. for j=1 to container.length
   for i=1 to ratio
   container[j].createAgent(file[i],container[j])
3. for j=1 to file.length
   score[j] = (receive (msg[j]); avg+=score/file.length;
4. for i=1 to no of agents if score[i]<0.5 * avg slowContainers[k++]=container[i];
   else if score[i]>= avg fastContainers[k++]=container[i];
5. for i=1 to no of slow Containers agent[i].move(fastContainer[i]
6. if allMessageFinished
7. return
```

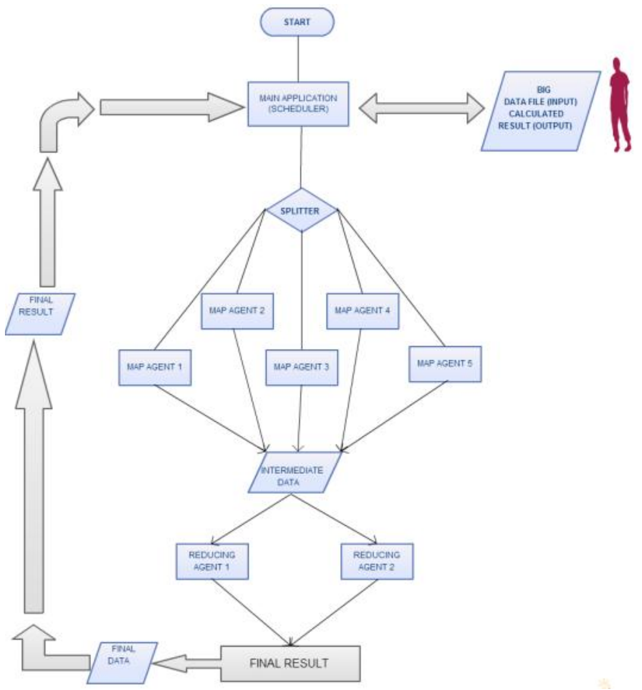


Figure 7: Flow Chart for the Proposed Solution

3.2 WORKING STEPS FOR CURRENT SOLUTION

3.2.1 Registration Phase: Each of the nodes who want to be the part of the computing grid first of all registers itself with the main node where the application has been running. This registration process is carried out for a fixed time period. Each node sends its IP Address to the main application so that it can further communicate with these nodes. These IP addresses are stored in the array and a corresponding agent container is launched in the jade runtime environment for each of the agent. This agent container controls all the operations to be performed on a particular agent.

Each node registering itself admits that it is meeting all the conditions it is required to fulfill and is readily offering its services to the parallel processing network.

3.2.2 File Fetching and Scheduler Agent Creation: In this step the address of the big data file to be processed it is fetched and then the scheduler agent is created along with the file address and the array of the agent containers created earlier. The scheduler then contacts all the agent containers and they are then used for the creation of agents. The scheduler in the meantime splits the file among multiple pieces based on the size of the split which it takes from the user.

3.2.3 Agent Score Calculation: Each agent calculates the score based on the number of words it has processed in unit time out of the total number of words it had to process using the formula

Score = (Number of words processed/Total number of words to be processed)*(unit time)

where the Number of words processed are counted by a variable,the Number of words to be processed are found by the String library in JAVA,the unit time is a constant set to value 100 milliseconds.

3.2.4 Agent Score Submission: Each agent then submits its partially calculated score and then sends it back to the scheduler; the agent uses the message format from the Agent class in JADE execution environment [44] to send the score of the current execution. The application has an array for the slowest and fastest running agents to store the exact values.

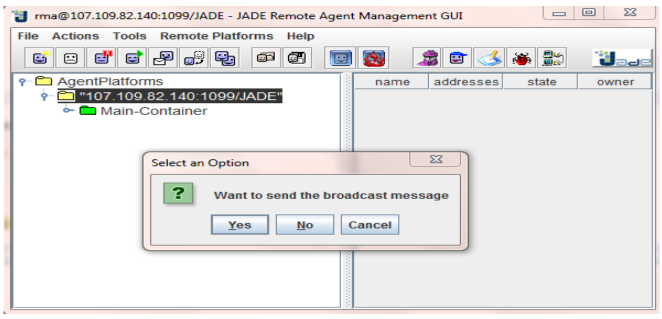
3.2.5 Agent Relocation: The arrays stored for slowest and fastest machines are used to send the location where the slow agents have to move where they can complete the original activity of the agent. The location is sent to the agent in a message particularly to the agent which the scheduler declares the straggler.

3.2.6 Agent Result Submission: The agent then submits the result to the application, which then merges all the results into the main result and then directs the final result to user. Final result is in the form of the file which contains the overall result.

3.2.7 Final Result Submission: The application then sends the result back to the users who have requested the task.

3.3 IMPLEMENTATION

It all starts with the main host starting which initially informs all the members of their network that they can now register with the main host if they want to be the part of the parallel data processing network.


rma@107.109.82.140:1099/JADE - JADE Remote Agent Management GUI

AgentPlatforms
"107.109.82.140:1099/JADE"
Main-Container

Select an Option
Want to send the broadcast message
File Actions Tools Remote Platforms Help
name addresses state owner
Yes No Cancel

Figure 8: Requesting Connection from slaves

Hosts then register with the IP address of the server (main host) which is sent to them in the message the main host broadcast the registration message. A main popup menu appears at the client which can then accept the offer or reject it.

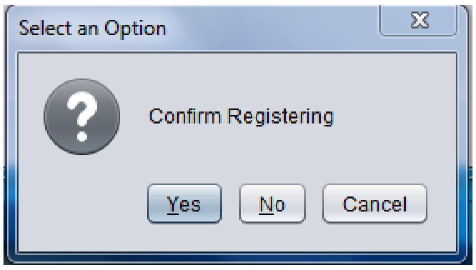
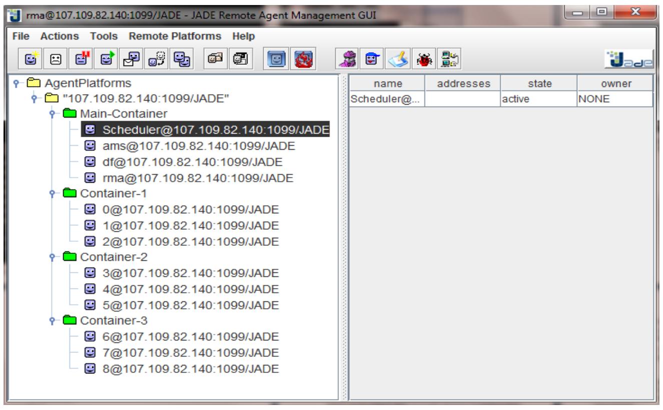

Select an Option
Confirm Registering

Figure 9: Confirming Request

The client who accepts the offer replies the message containing their IP addresses which is sent to the main host. Now after a pre specified time limit is over, the user sends the big data file address, i.e. where it is located and the scheduler is activated at the main host, which in turn activates the splitter to get the big data file splitted. The splits are assigned to the agents and the agents are assigned a specific location where it has to be created by the main container.

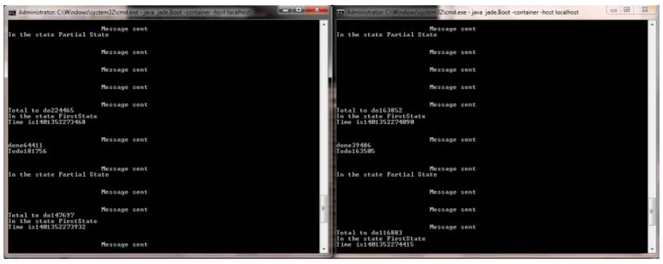


```
rma@107.109.82.140:1099/JADE - JADE Remote Agent Management GUI
AgentPlatforms
  "107.109.82.140:1099/JADE"
    Main-Container
      Scheduler@107.109.82.140:1099/JADE
      ams@107.109.82.140:1099/JADE
      df@107.109.82.140:1099/JADE
      rma@107.109.82.140:1099/JADE
    Container-1
      0@107.109.82.140:1099/JADE
      1@107.109.82.140:1099/JADE
      2@107.109.82.140:1099/JADE
    Container-2
      3@107.109.82.140:1099/JADE
      4@107.109.82.140:1099/JADE
      5@107.109.82.140:1099/JADE
    Container-3
      6@107.109.82.140:1099/JADE
      7@107.109.82.140:1099/JADE
      8@107.109.82.140:1099/JADE
```

name	addresses	state	owner
Scheduler@...		active	NONE

Figure 10: Created Agents and Scheduler Agent

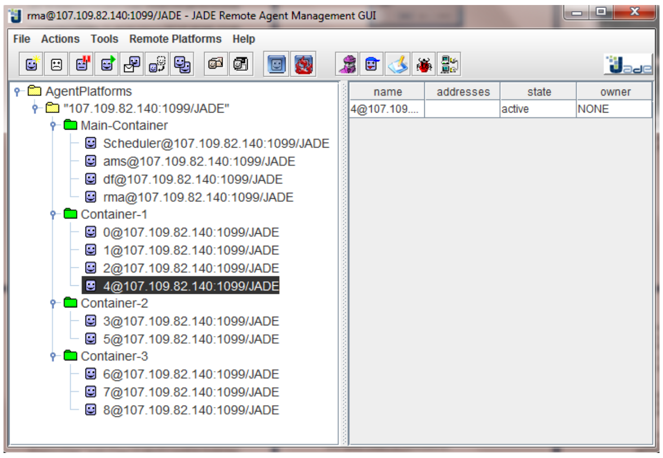
After the creation of all the agents the scheduler waits for a response from the agents i.e. the performance score is sent to the scheduler in the form of a message.



```
Administrator: C:\Windows\system32\cmd.exe - java jade.Boot -container -host localhost
                        Message sent
In the state Partial State

                        Message sent

                        Message sent

                        Message sent

                        Message sent
Total to do224465
In the state FirstState
Time is1401352273468

                        Message sent
done64411
Todo181756

                        Message sent
In the state Partial State

                        Message sent

                        Message sent
Total to do147697
In the state FirstState
Time is1401352273932

                        Message sent
```

```
Administrator: C:\Windows\system32\cmd.exe - java jade.Boot -container -host localhost
                        Message sent
In the state Partial State

                        Message sent

                        Message sent

                        Message sent

                        Message sent
Total to do163852
In the state FirstState
Time is1401352274090

                        Message sent
done39486
Todo163585

                        Message sent
In the state Partial State

                        Message sent

                        Message sent

                        Message sent
Total to do116883
In the state FirstState
Time is1401352274415
```

Figure 11: Message sending from various agents



```
rma@107.109.82.140:1099/JADE - JADE Remote Agent Management GUI
AgentPlatforms
  "107.109.82.140:1099/JADE"
    Main-Container
      Scheduler@107.109.82.140:1099/JADE
      ams@107.109.82.140:1099/JADE
      df@107.109.82.140:1099/JADE
      rma@107.109.82.140:1099/JADE
    Container-1
      0@107.109.82.140:1099/JADE
      1@107.109.82.140:1099/JADE
      2@107.109.82.140:1099/JADE
      4@107.109.82.140:1099/JADE
    Container-2
      3@107.109.82.140:1099/JADE
      5@107.109.82.140:1099/JADE
    Container-3
      6@107.109.82.140:1099/JADE
      7@107.109.82.140:1099/JADE
      8@107.109.82.140:1099/JADE
```

name	addresses	state	owner
4@107.109....		active	NONE

Figure 12: Rescheduled Agents

Finally the reducer's output file is sent to the scheduler who in turn sends it back to the user of the application. The Scheduler in turn collects the overall result and then sends this result to the user.

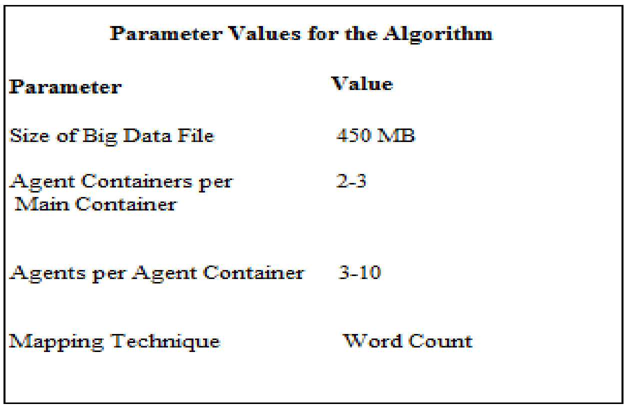
4. RESULT AND ANALYSIS

The results obtained by running various tests on the algorithm, and running it with different types of straggler arrangements. The proposed Algorithm is tested on both scenarios for split sizes as well as for various setting up of stragglers. All the results collected by the experiments and provide the comparison of the results with results of other technique proposed for solving the problem of straggler detection and mitigation.

4.1 TESTING

The implementation of the proposed algorithm is put to test on an Intel i5 laptop with 6GB RAM is used for the execution of the program for both scenarios with a 450MB text file. The map and reduce operations are defined for word count procedures. JADE is used for creating agents on the various machines.

The algorithm is run with the following values of the parameters:



Parameter Values for the Algorithm

Parameter	Value
Size of Big Data File	450 MB
Agent Containers per Main Container	2-3
Agents per Agent Container	3-10
Mapping Technique	Word Count

Figure 13: Parameter Values for the Algorithm

This gives the algorithm the flexibility to be applied to various types of networks. By varying these parameters one can get different execution times. If one provides values which are below or beyond those constrains, the results will not be same. The algorithm is tested to provide good results with these values of the parameters within constraints.

4.2 Comparison with HADOOP Native Scheduler based on Straggler Percentage

The proposed algorithm is compared with the HADOOP Native Scheduler for its straggler detection and mitigation technique's wide implementation. The following figure 14 clearly shows that the proposed technique outperforms this technique in execution time for randomly generated graphs. This algorithm provided better results than the algorithms used for comparison. Not only it outperforms the Hadoop Native Scheduler in its execution time but it does so without any DFS. It simply saves the extra time Hadoop takes to create the DFS and then resumes them from the place where they have left earlier.

Table 1: Execution Overview on Straggler %

Straggler %	HADOOP Native Scheduler Time(ms)	Mobile Agent Based MapReduce Time(ms)
10%	48587	51237
20%	58548	54893
40%	87373	74120
50%	120252	99365

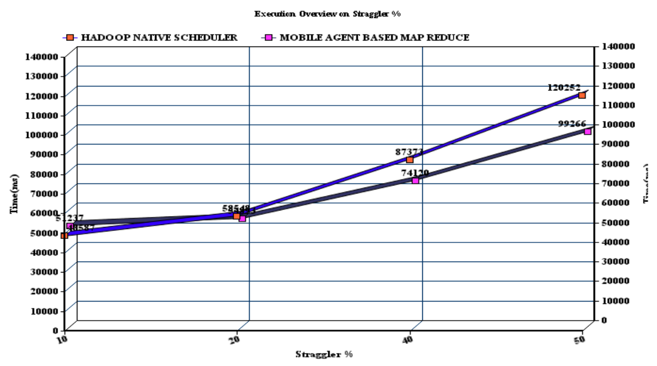


Figure 14: Line Graph Based On Straggler %

4.3 Comparison with HADOOP Native Scheduler based on Split Sizes

The proposed algorithm is compared with the HADOOP Native Scheduler for its straggler detection and mitigation technique's wide implementation. The following figure 15 clearly shows that the proposed technique outperforms this technique in execution time for randomly generated graphs for various split sizes. Our algorithm provided better results than the algorithms used for comparison. Not only it outperforms the Hadoop Native Scheduler in its execution time but it does so without any DFS. Split sizes however, can increase our problems because we used the network to transfer information from one machine to the other. While the information or splits are being transmitted the network speed comes into picture which can increase the overall execution time.

Table 2: Execution Overview on basis of various Split Sizes

Setup Scenario Size*Container*Agents/Container	HADOOP Native Scheduler Time(seconds)	Mobile Agent Based MapReduce Time(milliseconds)
50*3*3	45	48
45*2*5	47	48
30*3*5	50	49
15*3*10	55	55

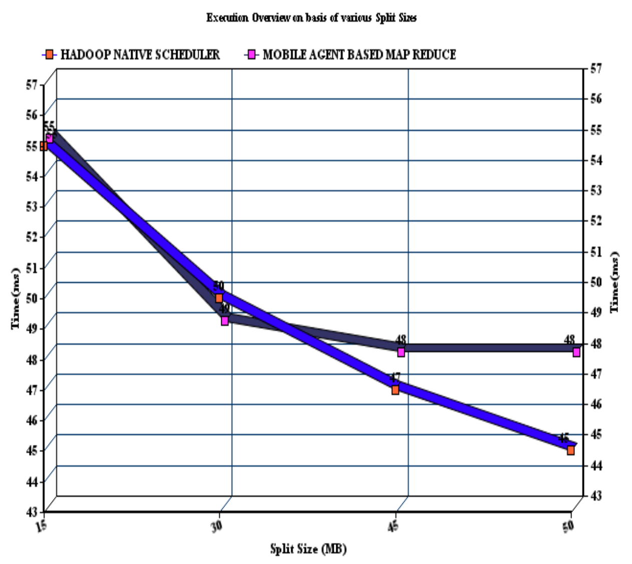


Figure 15: Line Graph Based On Split Size

4.4 RESULT ANALYSIS

From the previous section it is evident that the MBMR Algorithm performs well on single machine as well as for network of machines. The algorithm improves the execution time for a single machine and can serve the same purpose for if we have enough bandwidth. The significant insights from the result of the implementation of MBMR are as follows:

1. Outliers or stragglers are detected and are then rescheduled to other machine without wasting a single machine cycle because it reschedules them in an efficient manner.
2. Rescheduling process is delayed so that if a node has stuck in some other useful work it is not marked straggler and can continue its work. This

delay does not cost us much because we are using mobile agents which are known for their ability to start from the very place, they left their execution.

3. There is a significant improvement in the execution time as compared to existing algorithms i.e. about 10-12% and that as well without using any DFS.

5. CONCLUSION

5.1 Conclusion

Straggler detection and mitigation approaches have attracted a lot of attention of researchers in recent years and there is a considerable increase in the number of algorithms published for solving the issue as it has applications in various domains like big data processing and parallel computation. This report tries to review all popular algorithms for straggler detection and mitigation along with their rescheduling with their strengths and weaknesses. The report tries its best to review all popular algorithms, but the study is by no means complete as there are newer algorithms discovered at a fast rate because of the growing interest of researchers in this domain. This report describes nearly all the algorithms which exist for straggler detection, and also reviews their strengths and weaknesses. The basic concepts required for understanding the problem of straggler detection are described in great detail. The main goal was to come up with a technique which is better than the current state of art solutions. The proposed technique of Greedy Mobile Agent based MapReduce for Big Data Processing and straggler detection is described. Extensive tests are also performed and the results are also shown for the sake of validity of the proposed technique. The proposed technique performs well as compared to the classical algorithms and the current state of art algorithms. The report showed that the problem of outliers and their mitigation can be handled by the proposed greedy local technique.

5.2 Future Scope

In future the efforts will be concentrated on further optimizing the technique, so that the algorithm can scale to larger number of nodes and hence can be applied to large social networks. Another area of future work will be to come up with a parallel version of the algorithm, so that large networks can be processed in parallel, to reduce the execution time. Further scaling the algorithm to larger networks will also be considered.

The whole algorithm can be broken down into three main steps namely: finding containers, mobile agents and scheduler. In future the optimizations that can be performed in each of these steps are:

Optimization in the finding containers step

During this phase the algorithm simply wants the master to initiate the process which in turn increases the task for the master, adding this task to the slaves we can reduce the amount of the work for the master and can split this to the slaves.

Optimization in the mobile agent creation step

During the mobile agent creation phase the algorithm simply creates the mobile agent which serve the purpose of each split's processing. Mobile agent can be created in a more efficient way for parallel processing if we redefine the data structures and their behavior accordingly.

Optimization in the scheduling step

Scheduling starts when all agents are created and we have containers available for their scheduling. The scheduler's logic can be further optimized to reduce the execution time for the application.

Future work will also be concentrated on finding better local heuristic functions to reduce the execution time of the algorithm. Finally, better benchmarking abilities are required by which we can compare the algorithms faster.

References

1. Laura Wilber, Steve Mills, and Bill Perlowitz. "*Demystifying Big Data.",* Notices of TA Foundation, (2009).
2. Chris Eaton, Dirk Daroos, Tom Deustch., and George Lapis, Paul Zikopolous. "*Understanding Big Data:*.", (2011)
3. J. Dean and Sanjay Ghemawat. "*MapReduce: Simplified Data processing on large clusters* ", Commun. ACM, 51:107-113(2004).

4. Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, Dennis Fetterly, D., "*Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks*", EuroSys'07, 424, 175-308, (2007).
5. Spark homepage: http://www.spark-project.org
6. M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica. "*Effective Straggler Mitigation: Attack of the Clones"*. In Proceedings of the 8th USENIX conference on Operating systems design and implementation, OSDI'08, pages 29–42, 2008..
7. Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, Ion Stoica, "*Effective Straggler Mitigation: Attack of the Clones*", In USENIX NSDI,2012.
8. G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, E. Harris, and B. Saha. Reining in the Outliers in Map-Reduce Clusters using Mantri. In USENIX OSDI, 2010.
9. Rohan Gandhi, Amit Sabne "*Finding Stragglers in Hadoop*", In Proceedings of the 8th USENIX conference on Operating systems design and implementation, OSDI', 32, 425-443, (2008).
10. Freeman, L. C., "*Finding Stragglers in Parallel Computation*", ACM, 1, 215-239, (2009).
11. Carl W. Olofson, Randy Perry, "*IDC Analyze the future*", White Paper, 104, 36-41, (2011).
12. Guimerà, R., Sales-Pardo, M.. Amaral, L. A. N., "*Search Engine Architectures from Conventional to P2P*", Physical Review E, 70, 025101, (2012).
13. Hadoop Native Scheduler, Hadoop http://hadoop.apache.org/docs/r2.2.0/hadoop-project-dist/hadoop-common/NativeLibraries.html
14. Neil Raden, Hired Brains, "*Big Data Analytics Architecture*", Physical Review E, 70, 056131, 20 July (2012).
15. Chris Eaton, Dirk Daroos, Tom Deustch., and George Lapis, Paul Zikopolous. "*Understanding Big Data:.*", (2011)

Author Profile

This is to certify that this dissertation entitled **"Mobile Agent Based MapReduce Framework For Big Data Processing"** embodies the work carried out by **Sonu Yadav** pursuing M.Tech degree in Computer Science & Engineering at RPSGOI Mahendragarh, Haryana, India Affiliated from Maharshi Dayanand University, Rohtak-124001, Haryana, India under Ms. Komal Garg (Assistant Professor at Computer Science & Engineering Deptt.) and that it is worthy of consideration for the award of M.Tech degree.