International Journal of Grid Distribution Computing
Vol.7, No.4 (2014), pp. 139-148
http://dx.doi.org/10.14257/ijgdc.2014.7.4.13

A Comprehensive Review of Straggler Handling Algorithms for MapReduce Framework

Umesh Kumar and Jitendar Kumar
YMCA University of Science and Technology, Faridabad, 121006, India
umesh554@gmail.com, jitendar08cse28@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, Dryad, and Spark. 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. Stragglers extraordinarily impact little occupations such that employments comprising of a couple of undertakings. Such occupations are fundamentally deferred regardless of the fact that a solitary undertaking is moderate .This paper survey stragglers recognition and rescheduling systems proposed so far and brings up their strengths and shortcomings. This paper additionally displays wise attributes and impediments of the existing state- of-the- craftsmanship calculations to take care of the issue of stragglers relief.

Keywords: *Distributed computing, MapReduce, Dryad, Spark, Hadoop LATE, Mantri, MonTool, Dolly.*

1. Introduction

MapReduce, an essential model of preparing and producing large information sets, gives a simple to utilize, adhoc solution , to issues like Web Indexing, Data sorting, Data Searching and so forth. In Big Data Applications, Big Data Companies like Google, Face book, Yahoo and so forth have been utilizing MapReduce [1] as a part of one structure or the other. Stragglers are normally created because of variety in the CPU accessibility, network traffic or I/O discord.

Since a job in MapReduce Framework does not completes until all map and reduce undertakings are done, even little number of stragglers can generally deteriorate the general reaction time for the occupation. It is accordingly important to early detection or stragglers and proficient rescheduling them to other similarly quicker frameworks. Prior the identification of the straggler, better will be the general reaction time for the employment.

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 [2]. 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 [3]. At long last, Stragglers ought to be recognized as right on time as could reasonably be expected.

The Problem of Stragglers has accepted extensive consideration recently with numerous stragglers moderating methods being created [4-7]. These strategies could be

ISSN: 2005-4262 IJGDC

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. Speculative execution holds up to watch the advancement of the assignment of work and propels double duplicate of slower assignments on different machines. This paper exhibits an orderly and sorted out investigation of stragglers recognition and relief systems. The qualities and shortcomings of every method is likewise matter of center in this paper.

The major contribution of this paper will be:-

- To serve as a base for those beginning research in this bearing.
- To give them the existing state of the workmanship calculations for the exploration issue.
- To make them mindful of the challenge in the heading and the results proposed in this way.

This paper is organized as follows: Section 2 describes the Big Data processing framework MapReduce. Section 3 describes why stragglers exist in distributed computing. Section 4 describes various techniques used so far for efficiently handling stragglers and their corresponding strengths and weaknesses and at last conclusion which sums up the work conducted and the future directions for work.

2. MapReduce Framework

MapReduce is one of the parallel data processing ideal models intended for substantial data processing on cluster based computing architectures [1]. It was initially proposed by Google to handle huge scale web search applications. The 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 applications. As demonstrated in Figure 1 [9].

MapReduce Programming model consists of data processing functions: map and reduce. Parallel map tasks are run one input data which is partitioned into fixed size blocks and produce intermediate output as a collection of <key, value> pairs.

3. Why Stragglers Exist

Stragglers are the undertakings that take longer time to execute than other comparative tasks. There are numerous purposes 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.

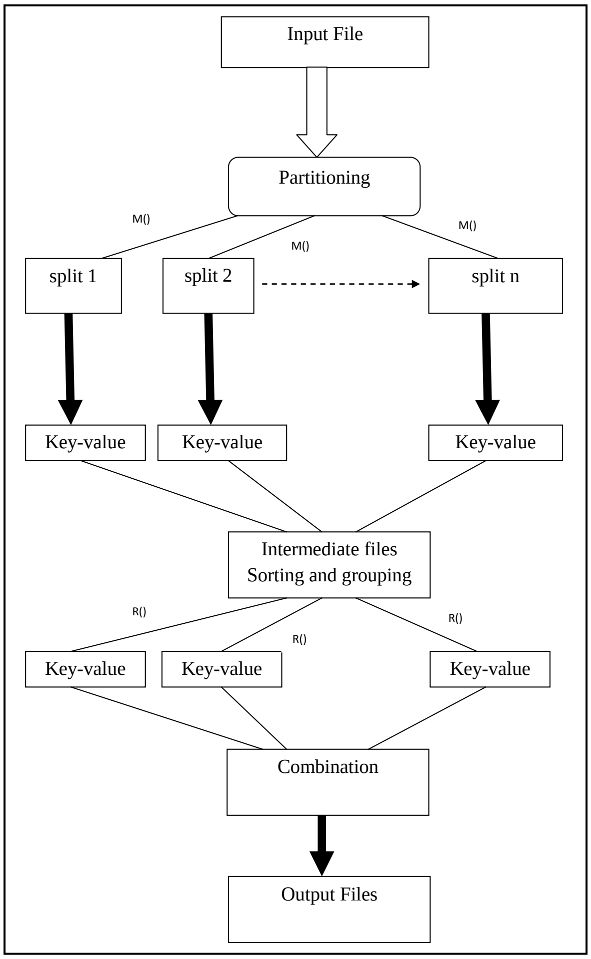

Figure 1. MapReduce Framework

4. Techniques

Various Stragglers detection and mitigation algorithms are: **Hadoop native scheduler**, **LATE, Mantri, MonTool, Dolly.**

Hadoop Native Scheduler [8] allots an advancement score between 0 and 1 to each assignment. For a map, the advancement score speak to the measure of input data read. For diminish tasks, the execution is separated into three stages, each of which records for 1/3 of the score.

i The duplicate stage, when the task fetches map yield.
ii The sort stage, where map output are sorted by key.
iii The decrease stage, when a client defined functions is applied to the list of map output.

In each phase the score is the function of data processed so far. Hadoop then computes the average process score to every class of tasks to characterize a limit for speculative execution. For each task, it holds up for a moment and after that if the tasks advancement score is short of what normal for its class less 0.2, it is stamped as a straggler. All tasks beneath the limit are acknowledged similarly abate and ties between them are determined by information region. Nonetheless, the scheduler guarantees that only one speculative copy of each one task is running at whenever, yet it does consider the genuine advancement score while rescheduling them. Some of the crucial assumptions that break in virtualized heterogeneous clusters are:-

i *Hadoop expects nodes are homogeneous and can perform work at harshly the same rate and tasks advances at a consistent rate all around time.* Since there are numerous virtual machines running on a node, each one undertaking can't execute at the same rate.

ii *While lessen operation, the duplicate, and sort and diminish stages each one take about 1/3 of the aggregate time.* By and large, the information read from every mapper is vast and it is exceptionally conceivable that duplicate and sort stage have invested much bigger time than genuine lessen stage.

Task in the same classification (map/reduce) perform comparative measure of work. This assumption especially fizzles for lessen tasks as we can have huge change in keys allotted to a specific reducer.

LATE [5] addresses the errors brought about in the Hadoop scheduler in a heterogeneous environment, LATE comes up with an alternate system that approximates the completion time for tasks in same class to predict foresee potential stragglers.

i 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.

ii Time to finish is then approximated as (1-Progress Score)/Progress Rate.

iii Tasks with advancement rates beneath an edge of 25 percentile of all tasks are acknowledged to be stragglers.

iv LATE stays informed regarding moderate nodes in the system and does not run speculative duplicates on those nodes.

v 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.

LATE enjoys following advantages:

i 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.

ii LATE takes into account node heterogeneity while deciding where to speculate tasks.

iii 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.

However, LATE scheduler has following demerits:

i 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 an candidate to be speculated resulting in wasting assets.

ii 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.
iii Starting assessment time needed by the LATE scheduler is high (1 minute) before an undertaking could be stamped as straggler.

This 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.

Mantri: Reining in outliers [4] utilizes real time advancement reports; Mantri identifies and follows up on stragglers early in their lifetime. Early activity authorizes assets that might be utilized by resulting assignments and speeds up the employment general. Acting dependent upon the causes and the resource and opportunity expense of actions lets Mantri enhance over former work that just doubles the laggards.
It uses the following techniques:

i Restarting outlier tasks conscious of asset constraints and work imbalance characteristics,
ii Network mindful position of tasks, and
iii Ensuring yield of tasks dependent upon an expense profit analysis.

Mantri spots tasks dependent upon the areas of their data sources and in addition the current usage of network links as shown in fig. 3 [4]. Mantri recognizes points at which tasks are unable to make progress at the ordinary rate and implements targeted results. The controlling standards that recognize Mantri from former outlier relief plans are cause awareness and resource perception. Notable activities are needed for different reasons.

Key Mechanism in Mantri:

i If expected remaining time (trem) for a task is more than the normal runtime of the task after a restart (tnew), the task is restarted up to a greatest of three times.
ii A speculative duplicate is scheduled just if the measure of resource obliged is decreased thusly. At most three could be three duplicates of the same tasks.

Estimation of t_{rem} and t_{new} given in eq. 1 and 2

$$t_{rem} = (t_{elapsed} * d/d_{read}) + t_{wrapup} \quad (1)$$

where d= whole data to process ,
d_{read} =data already processed and
t_{wrapup}= time to recollect results

$$t_{new} = processRate*locationFactor*d+schedLag. \quad (2)$$

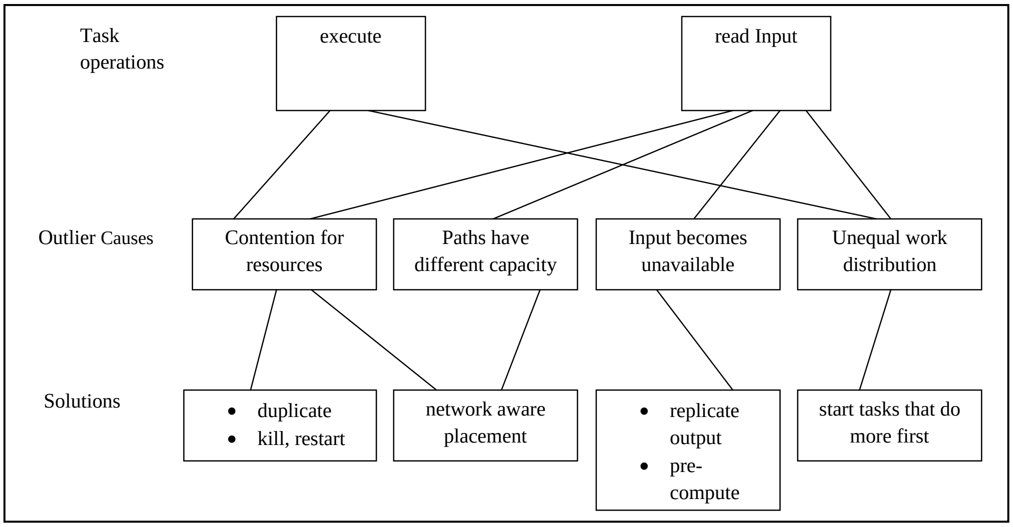

Figure 3. Outlier's Causes and Solutions

First term is a distribution of the process rate, *i.e.,* Δtime/Δdata, of all the tasks in this phase.

The second term is a relative component that records for whether the candidate machine for running this task is relentlessly slower (or speedier) than different machines or has littler (or bigger) limit on the network path to where the task inputs are found. The third term, as in the recent past, is the measure of information the task need to process. The last term is the average delay between an assignment being scheduled and when it gets to run. However, Mantri is not intended for heterogeneous situations either, and might in this manner face issues like the Hadoop's native scheduler.

MonTool [6] gathers information about the tasks by tracing system calls and analyzing them. With this information MonTool finds the stragglers as well as their causes.

- Montool assembles data about the assignments by following system calls and examining them. With this information Montool finds the stragglers and their reasons. Montool produces system call state machines for all system calls each 10msec by collecting the disk/network read and write system calls for map and reduce processes that are made by Hadoop.
- Montool filtering methodology go in sleep mode for 2 seconds and in the wake of getting up, it prepare the system calls assembled in 2 seconds and this information is then sent to the Master.
- Montool daemon running at expert gets framework call follow from all laborer hubs and looks at the state machines for all undertakings running and computes similarity score for state machines.

Similarity Score for two processes is calculated as eq. 3

Similarity Score=1-$(N_i^1 * \sum_{i=1}^{n} (N_i^1 - N_i^2)/N_i^1)$ (3)

Where N_i^1 and N_i^2 denote the number of state transition for ith state pair for process 1 and 2 respectively.

- The Threshold for straggler is situated to 85 % such that if one process makes < 85% system calls than the average, it is checked as straggler and a theoretical duplicate is launched.

Limitations:

i *It accepts all map or diminish tasks work upon similar measured workloads and access information in a similar pattern. Be that as it may this assumption reduce tasks as information size read by diminish tasks may be distinctive for each task*

ii *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.*

Attack of the Clones: DOLLY [7] this 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.

Cloning of small jobs can be achieved with few extra resources because of the heavy tail distribution of job sizes; the majority of the jobs is small and can be cloned with little overhead. The main challenge of cloning was making the intermediate data transfer efficient i.e. avoiding multiple tasks downstream in the job from contending for the same upstream output.

Intermediate data access with Dolly

Dolly defines its approaches for mitigating contention while accessing intermediate data from various map processes finishing simultaneously.

- 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 as given in eq. 4.

$$\Psi (n,c,d) = \sum^{c-d}_{i=0} (c/i)\, p^{i} (1-p)^{c-i} \qquad (4)$$

Dolly defined probability for job straggling with CAC as eq. 5

$$P= 1-\sum^{c}_{d=1}[\psi (n, c, d)-\psi (n, c, d-1)]*(1-p^{d})^{n} \qquad (5)$$

- 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.

Dolly defined probability for job straggling with CC as eq. 6

$$P= 1-\sum^{c}_{d=1}[\psi (n, c, 1)]*(1-p^{d})^{n} \qquad (6)$$

- 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.

Dolly improves the average completion time of jobs by 42% compared to Late and 40% compared to Mantri in the Facebook workload using less than 5% extra resources and Delay assignment outperforms CAC and CC by 2 times.

Comparison Table of Various Tools

COMPARISON	HADOOP NATIVE SCHEDULER [8]	LATE[5]	MANTRI [4]	DOLLY[7]	MONTOOL [6]
Metric for Speculative Execution	Equal Weight age for Map And Reduce Tasks	Finish Time	Causes Detection and Solution	Simple Cloning	System Calls
Cap on Number of Speculative Execution	No	No	No	Yes	No
Data Processing Technique	MapReduce	MapReduce	MapReduce / Dryad	MapReduce / Dryad	MapReduce
Heterogeneity among Network Nodes	No	Yes	Yes	Yes	Yes
Priority Wise Scheduling	Yes	No	No	No	No
Overhead	More	Moderate	Moderate	More	More

5. Conclusion and Future Work

Capability to make Parallel data processing resilient to problems and henceforth enhancing their performance under heterogeneous situations is one of the emerging research area that gets the vast majority of the researchers as just a slight change in execution can spare a great deal of resources throughout the year of processing. This paper tries to survey all the prevalent calculations for detecting and mitigating stragglers. We have attempted our best to audit all prominent algorithms, but the study is by no means complete as there are newer algorithms discovered at a faster rate because of the growing interest of researchers in this domain. Future work will consider scheduling in distributed map reduce using mobile agents in heterogeneous clusters.

References

[1] J. Dean and S. Ghemawat, “Mapreduce: Simplified data processing on large clusters”, OSDI'04 Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation, vol. 6, (2004), pp. 10-10.

[2] M. Isard, M. Budiu, Y. Yu, A. Birell and D. Fetterly, “Dryad: Distributed Data-parallel Programs from Sequential building Blocks”, Eurosys’07 Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems, vol. 41, (2007), pp. 59-72.

[3] M. Zaharia, M. Choudhury., T. Das, A. Dave, J. Ma, M. Mccauley, M. J. Franklin, S. Shenker, and I. Stoica, “Resilient distributed data sets: A fault- tolerant abstraction for in memory cluster computing”, USENIX

Symposium on Networked Systems Design and Implementation, **(2012)**. https://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf

[4] G. Ananthanarayanan, S. Kandula, A. Green berg, I. Stoica, E. Harris and B. Shaha, “Reining in the outliers in MapReduce Clusters using Mantri”, 9th USENIX Symposium on Networked Systems Design and Implementation, **(2010)**. https://www.usenix.org/legacy/event/osdi10/tech/full_papers/Ananthanarayanan.pdf.

[5] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, I. Stoica, “Improving MapReduce Performance in heterogeneous environment”, OSDI'08 Proceedings of the 8th USENIX conference on Operating systems design and implementation, **(2008)**, pp. 29-42.

[6] Rohan Gandhi and Amit Sabne, “Finding Stragglers in Hadoop”, European Conference on Computer Systems, **(2012)**. https://engineering.purdue.edu/~ychu/ee673/Projects.F11/detectstraggeler_finalrpt.pdf.

[7] G. Ananthanarayan, A. ghodsi, Scott Shenker, Ion Stoica, “Effective Straggler Mitigation: Attack of the Clones”, Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation, **(2013)**, pp. 185-198.

[8] http://hadoop.apache.org/.

[9] http://mapreduce.wiki.org/.

Authors

Umesh Kumar, received his M.Tech. degree in Computer Engineering from YMCA University of Science and Technology, Faridabad, India. Presently he is working as an Assistant Professor in the Computer Engineering department of same University. His research interests include wireless security, mobile agent, distributed computing.

Jitendar Kumar, is a post graduate student of Mr. Umesh Kumar, studying at YMCA University of Science and Technology, Faridabad, India. His research interests include software application development, Big Data processing and distributed computing.