View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Institute of Advanced Engineering and Science

International Journal of Electrical and Computer Engineering (IJECE)
Vol. 11, No. 1, February 2021, pp. 375~382
ISSN: 2088-8708, DOI: 10.11591/ijece.v11i1.pp375-382

Straggler handling approaches in mapreduce framework: a comparative study

Anwar H. Katrawi[1], Rosni Abdullah[2], Mohammed Anbar[3], Ibrahim AlShourbaji[4], Ammar Kamal Abasi[5]
[1,3]National Advanced IPv6 Center (Nav6), Universiti Sains Malaysia, Malaysia
[2,5]School of Computer Sciences, Universiti Sains Malaysia, Malaysia
[4]Department of Computer and Engineering, Jazan University, Saudi Arabia

Article Info

Article history:

Received Mar 22, 2020
Revised May 21, 2020
Accepted Aug 5, 2020

Keywords:

Big data
Hadoop
MapReduce
Spark
Straggler

ABSTRACT

The proliferation of information technology produces a huge amount of data called big data that cannot be processed by traditional database systems. These Various types of data come from different sources. However, stragglers are a major bottleneck in big data processing, and hence the early detection and accurate identification of stragglers can have important impacts on the performance of big data processing. This work aims to assess five stragglers identification methods: Hadoop native scheduler, LATE Scheduler, Mantri, MonTool, and Dolly. The performance of these techniques was evaluated based on three benchmarked methods: Sort, Grep and WordCount. The results show that the LATE Scheduler performs the best and it would be efficient to obtain better results for stragglers identification.

This is an open access article under the CC BY-SA license.

Corresponding Author:

Anwar H. Katrawi,
National Advanced IPv6 Center (Nav6),
Universiti Sains Malaysia,
11800 USM, Penang, Malaysia.
Email: akatrawi@student.usm.my

1. INTRODUCTION

With the excessive growth in information and data, their analysis becomes a challenge and more complex due to the increased volume of structured and unstructured data that are produced by the internet of things (IoT), social media, multimedia etc. Application such as MapReduce is a fault tolerant, scalable and simple framework for data processing that enables its users to process these massive amounts of data effectively [1, 2]. MapReduce is a significant model of preparing and generating a set of enormous information. This is because; it gives simple utilization environment, offer solution to ad hoc and to misses like Data sorting, Web indexing among several others. MapReduce is utilized in Big Information Applications in bigger Companies such as Yahoo and Google among several others.

The MapReduce is unlisted as a section of one structure or the other. The reason for creating stragglers is the diversity in accessibility in the CPU, I/O discord or network traffic. When the map and reduce are completed, that is when the MapReduce Framework is accomplished [3, 4]. In MapReduce Framework the job is not accomplished till very reduce and map undertakings are completed. Moreover, the quantity of the stragglers weakens with the wide-range of the time occupation [5-8]. In a heterogeneous environment, some compute nodes are faster than the other. Slower compute nodes are called stragglers node and these fast nodes will finish their tasks early and wait for the stragglers to finish.

Sometimes nodes fail due to hardware or software failures. Therefore, it is significant to detect stragglers in an early stage.to avoid performance discarion in the systems

Nowadays organizations with a large amount of data have complexity in processing and analyzing using traditional database management systems. By designing MapReduce, Google has made millions of users around the world find content from millions of pages within a hundredth of a second; therefore, the bulk processing problem has become a big challenge and their analysis technologies are changing rapidly. On the other side, stragglers are well recognized as a major bottleneck in big data processing and they can have significant impacts on big data processing. This work aims to evaluate five stragglers identification methods: Hadoop native scheduler, longest approximate time to end (LATE) Scheduler, Mantri, MonTool, and Dolly The performance of these techniques was assessed based on three benchmarked methods: Sort, Grep and WordCount.

The remainder of this paper is structured as follows: The second section depicts MapReduce and struggles. In the third segment, five stragglers identification approaches are presented. In the forth segement, expermentalresults are presented This section is followed by the conclusion and future work of this study.

2. MAPREDUCE FRAMEWORK AND STRAGGLERS

MapReduce is the matching information preparing perfect models proposed for considerable information preparation on bunch-based figuring designs [9]. This system is used inside in facilitating data mining, search applications and machine learning at server centers. The philosophy needs to deal with broad scale web search applications. It was at first suggested by Google to deal with tremendous scale web search applications. The focus is licensing programmers in extracting from problems such as parallelization, booking, allocating, thus allowing them to focus on developing applications. In modern organizations and enterprises, there are four factors, including processing, storing, visualization, and analyzing large data. MapReduce can run the applications on a parallel cluster of hardware automatically. In addition; it can process terabytes and petabytes of data more rapidly.

Recently, it has gained popularity in a wide range of applications due to its ability to provide a highly effective and efficient framework for the parallel execution of the applications, data allocation in distributed database systems, and fault tolerance network communications. As illustrated in Figure 1, parallel map assignments are run as one input data as a gathering of <key, value> sets which is that is divided into fixed produce and size blocks transitional output. The model of MapReduce Programming comprises of information preparing capacities: map and reduce.

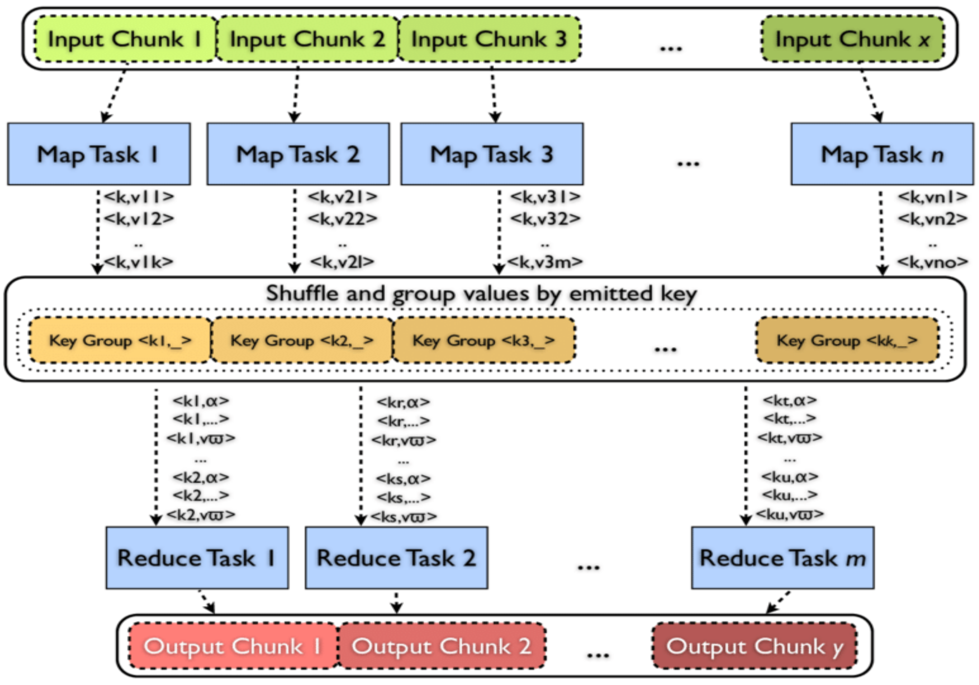

Input Chunk 1
Input Chunk 2
Input Chunk 3
...
Input Chunk x
Map Task 1
Map Task 2
Map Task 3
...
Map Task n
<k,v11> <k,v12> .. <k,v1k>
<k,v21> <k,v22> .. <k,v2l>
<k,v31> <k,v32> .. <k,v3m>
<k,vn1> <k,vn2> .. <k,vno>
Shuffle and group values by emitted key
Key Group <k1,_>
Key Group <k2,_>
Key Group <k3,_>
...
Key Group <kk,_>
<k1,α> <k1,...> <k1,vϖ> ... <k2,α> <k2,...> <k2,vϖ>
<kr,α> <kr,...> <kr,vϖ> ... <ks,α> <ks,...> <ks,vϖ>
<kt,α> <kt,...> <kt,vϖ> ... <ku,α> <ku,...> <ku,vϖ>
Reduce Task 1
Reduce Task 2
...
Reduce Task m
Output Chunk 1
Output Chunk 2
...
Output Chunk y

Figure 1. MapReduce framework

In the Map-phase, when the user requests to perform a job, the tasks are sent to the Map machines to run. The Combiner reduces the amount of data transmission in the network in the Reduce phase. Sort or Merging part is a part of the Reduce-phase. The time is used to integrate Map outputs from different nodes and this integration is considered as a Reduce time. The Reduce-part is the last step to run the job in a MapReduce way. The effect of each part of this process on runtime is different and to estimate the end time of each job, appropriate weights (the effect of each part on execution process is obtained from the ratio of the time of each part to the total time) should be used for each part. For more details see [9].

Stragglers are the endeavors that set aside longer effort in execution other than comparative assignments. Behind the assignment, there are various purpose to set aside longer effort, such as, imperfect machines, proportion of information to process, framework blockage, heterogeneity among equipment, and contention for the current resources [10, 11]. In any case, it is not significant for an assignment to be slower all around its execution. In addition, in the event that one task running slow on a given machine, it isn't significant for the completely future and present assignment to run slower on that particular machine. In dealing with Mantri, it is essential to keep in mind three primary mechanisms:

- In the event it is found that the expected remaining time exceeds the normal runtime, then the process will be restarted up to thrice.
- In the event that the resource measurement decreases undesirably, then scheduling of a speculative duplicate takes place. The expected remaining time (trem) and the normal runtime (tnew) are estimated as illustrated in the following algorithms.
- term = (telapsed * d/dread) + twrapup
- tnew = processRate∗locationFactor∗d+schedLag.

Many reasons for such stragglers to occur including load imbalance, scheduling inefficiencies, data locality, communication overheads hardware heterogeneity [12, 13]. There have also been efforts looking to address one or more of these concerns to mitigate the problem [14-16]. Sesipite all of these prior efforts are important and useful in overcoming this problem, we believe that a rigorous set of analytical tools is needed in order to better understand the consequences of stragglers on the performance slowdown in big data [17, 18].

3. STRUGGLERS IDENTIFICATION TECHNIQUES

Various methods are proposed for stragglers identification. In this section, five strugglers identification methods including Hadoop native scheduler, LATEscheduler, Mantri, MonTool, and Dolly to assess their sutabiltuy for stragglers identification.

3.1. Hadoop native scheduler

Hadoop native scheduler allocates a progression score somewhere in the range of 0 and 1 to each task. For lessen assignments, to execute is isolated into 3 phases, every one of that records for 1/3 of the score. In the case of a map, the progression count addresses the proportion of input information [19]. The stages are:

- At the first stage (copy stage), the assignment gets the map output.
- In the second stage (sort arrange), the map output is arranged by key.
- The decline stage, the customer-characterized capacities are connected to the rundown of guide yield.

For every task, it grasps up per minute and the assignment headway score is found to be short of typical for the group less 0.2, the unit is stepped as a straggler. In addition, at every stage, the score is the capacity of data prepared. At that point, Hadoop figures the normal procedure value to each classification of assignments for a description of a farthest point for theoretical completion. Task below the farthest point are recognized comparatively lessen and the ties between them are controlled by data locale. In any case, the scheduler ensures that just a single theoretical duplicate of every one assignment is running at whatever point, however reschedules scores while considering their authentic progression. A few of the pivotal suspicions that break in virtualized, heterogeneous bunches are-

- In decrease activity, the sort, copy, and lessen phases every one proceeds around 1/3 of the total time.
- Nodes expected by Hadoop can execute the job at a harshly equivalent rate. In addition, these are homogeneous, while undertakings propel at a reliable rate all around time.

Task in a similar characterization performs relative proportion of work. This presumption particularly fails for reduce undertakings in having tremendous modification in keys apportioned to an explicit reducer.

3.2. LATE scheduler

LATE considers another framework that estimates the finishing time for assignments in matching class to anticipate prediction to the stragglers that have potential. In addition, LATE addresses in errors in the Hadoop scheduler achieved in a varied condition [20].

- LATE uses a cap on the measure of theoretical assignments that can keep running at once, to deal with the manner in, which that theoretical assignments cost resources.
- LATE remains educated in regards to direct nodes in the framework and never runs theoretical copies on those nodes.
- Assignments with headway rates underneath 25 percent edge of summarized undertakings are recognized as stragglers.
- Score for the progress is found out using the Hadoop scheduler locally. The rate of Advancement is determined as progression score/T whereby T represents the running time of the assignment.
- Completion time is estimated by the formula (1-Progress Score)/Progress Rate.

LATE appreciates following favorable circumstances:

- Likewise, by monitoring surveyed left time rather than headway rate, LATE speculatively execute just assignments, which advance job reaction time, as contradicted to any slower tasks.
- LATE considers nodes heterogeneity while choosing where to hypothesize assignments.
- It is strong to heterogeneity of node considering that that re-impelling only slows the assignments and only a few amounts of undertakings.

However, LATE Scheduler has the following weakness [21]:

- Time required by LATE Scheduler is found to be higher, usually one minute, in starting evaluation, prior to an activity being stepped as a straggler.
- In the final activity, time for a task is determined using the determined out-progression rate middle value of the out-headway rate beside the present progression rate, the ending time foreseen is inclined to mixed up.
- Enormous undertaking will tend to take to a greater degree of risk than the rests to process, along these lines it is possible to be labelled as a possibility to be conjectured bringing about squandering resources.

If there is no explanation of the temperate nature of the acknowledged stragglers searched, the straggler assurance may be wrong. This essentially prompts much response time.

3.3. Mantri

Mantri is superior concerning outliers since it uses continuous progression reports; Mantri follows up and distinguishes the stragglers from the start. Early operations permit the assets used by coming assignments and accelerate the generally the employment [21]. Mantri is given an opportunity to upgrade over previous work that just duplicates the slouches by the acting dependently upon the effects and the asset and opportunity cost of activities. It utilizes the accompanying methods:

- Restarting exception tasks aware of advantage imperatives and work unevenness attributes,
- Ensuring yield of assignments subordinate upon a cost benefit analysis, and
- Network careful position of assignments.

Mantri perceives focuses at which assignments are unfit to gain ground at the customary rate and executes focused on results. Mantri spots undertakings subordinate in the area of their information sources,The controlling gauges that perceive According to Mantri previous anomaly relief outlier, plans are effects to mindfulness and asset discernment. Remarkable exercises are required for various reasons. Key Mechanism in Mantri:

- A theoretical copy is scheduled just if the proportion of asset obliged is diminished along these lines. At most three could be three copies of similar errands (tasks).
- If the anticipated residual time for an errand is much higher than the ordinary running time of the assignment after restarting, the assignment is started again until the highest value is achieved.

The approximation of trem and t can be written as:

$$Trem = \left(telapsed * \frac{d}{fear}\right) + twrapup \tag{1}$$

where d= entire information to process, fear=data officially prepared and twrapup= time to recall results

$$Tnew = processRate * \ locationFactor * d + schedLag \tag{2}$$

3.4. Mon tool

It assembles information on the errands by succeeding framework calls and investigating them. In addition, MonTool discovers the stragglers, stragglers effects and their reasons using the data. The daemon running of Montool at masters gives a structure pursue call from all workers center point. Moreover, it takes the state machines for various endeavors in figures and running likeness state machines; score. Montool sifting approach enter rest mode for 2 seconds and in the in returning active mode, it set up the framework calls gathered in 2 seconds [22]. The master receives the sent information at that point. Then the Montool gathers information concerning the assignment by succeeding the framework calls and looking at them. In addition, it produces framework call state machines for all framework calls every 10mseconds by gathering plate/organize readings and compose framework calls for guide and diminish forms made by Hadoop. Similitude (Similarities) Score for two procedures is determined as:

$$Similarity\ score = 1 - \left(N_i^1 \sum_{i=1}^{n} (N_i^1 - N_i^2)/N_i^1\right) \quad (3)$$

where Ni 1 furthermore, Ni 2 stands for the quantity of progress state for *ith*pair state for first and second separately.

The Threshold for straggler is arranged to 85 percent to such an extent that on the off chance one procedure makes <85percentageframework calls, it is patterned as a straggler while the hypothetical copy is launched.

Confinements:

- Associating framework calls can't be accomplished without information about the keys. In addition, the case of keys is consistently unavailable in Hadoop.
- It acknowledges all the maps or reduces assignments work upon comparable estimated remaining tasks and get to data in a comparative pattern. In any case, this supposition lessens tasks as data size perused by lessen assignments might be particular for each undertaking

3.5. DOLLY

DOLLY oversees this system in managing the stragglers in a proactive manner .The primary challenge of cloning was coming up with intermediate information transfer effective such as maintaining a strategic distance from various assignments downstream in the activity from battling for the equivalent upstream yield. Cloning of little occupations can be accomplished with a couple of additional assets in view of the overwhelming tail circulation of occupation estimates [23, 24]; most of the employments are little and can be cloned with nearly nothing overhead. As restricted to hold up and endeavoring to foresee stragglers, then it takes theoretical implementation to its outrageous and dispatches various clones of all undertakings of the relevant occupation and simply utilize the consequence of the first clone to finish [25]. Intermediary information access with Dolly characterizes its methodologies for moderating dispute while getting to intermediary information from different map procedures completing at the same time. Table 1 provides comparison between stragglers identification methods used in this work.

- Contention avoidance cloning (CAC): Here when an upstream assignment clone completes, the yield is directed to precisely a single task on the downstream order for clone-clone collection.

Ψ (n, c, d) =Probability [n upstream errands of c clones with > = d clones per collection.]

where, p =likelihood of an errand straggling as:

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

Dolly characterized likelihood for occupation straggling with CAC as:

$$\mathrm{P} = 1 - \sum_{d=1}^{c} [\Psi(n,c,d) - \Psi(n,c,d-1)(1-p^d)^n] \quad (5)$$

- Contention cloning (CC): After the completion of the upstream clone assignment, all downstream errands read the clones' upstream output, which eases the issue of conflict. Dolly characterized likelihood for task straggling with CC as:

$$\mathrm{P} = 1 - \sum_{d=1}^{c} [\Psi(n,c,1)(1-p^d)^n] \quad (6)$$

- Every downstream clone waits for an ideal period (ώ) to see whether it can catch an elite duplicate of intermediary information. In the event that the downstream clones do not get its restrictive duplicate even subsequent to hanging tight for ώ, it peruses dispute to one of the completed upstream clones yield. The ideal time of ώ considers typical varieties amongst upstream clones.

Table 1. Methods comparison of stragglers identification

Comparison	Hadoop native scheduler	LATE Scheduler	Mantri	Mon Tool	Dolly
Metric for Speculative Execution	Weight equivalence for Reduce and Map jobs	Time taken to complete task	Bases Identification and corrective measure	System Call	Unsophisticated Cloning
Cap on Number of Speculative Execution	Negative	Negative	Negative	Negative	Affirmative
Data Processing Technique	MapReduce	MapReduce	MapReduce / Dryad	MapReduce	MapReduce/ Dryad
Heterogeneity among Network Nodes	Negative	Affirmative	Affirmative	Affirmative	Affirmative
Priority Wise Scheduling	Affirmative	Negative	Negative	Negative	Negative
Overhead	Increased	Medium	Medium	Increased	Increased

In Table 1, the metric for speculative execution for the five straggler detection techniques were found to be different for each one the techniques. However, this was not the case for the Cap on number of speculative executions. All the techniques did not exhibit any characteristic for this category. The data processing technique was found to be similar for Hadoop native scheduler, LATE and Montool. The heterogeneity among network nodes were found to be similar for all techniques except the Hadoop native scheduler technique. This was also found to be the same case for priority wise scheduling.

4. EXPERMENTAL RESULTS AND DISCUSSION

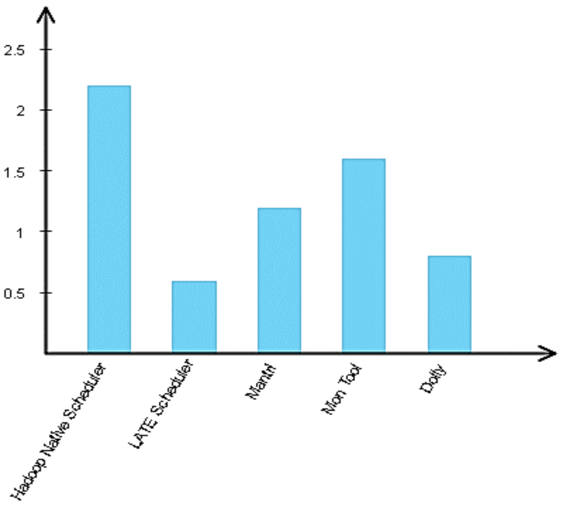
To assess the process of diagnosing straggler tasks and assigning them to other nodes i.e. speculative execution for the Hadoop native scheduler, LATE scheduler, Mantri, MonTool, and Dolly, three benchmarked methods are used which include Sort, Grep and WordCount. We manually slowed down 8 virtual machines (VMs) in a cluster of 90 with background processes to simulate stragglers.– faulty nodes. The other machines were allocated between 1 and 10 VMs per host. As our workload, we evaluated the work with a "sort" task in a total dataset of 40 GB and each of 128 MB for each host. Each job has 575 tasks on the map and 510 reduced tasks (it should be noted that Hadoop leaves some free capacity for speculation and failed missions). The stragglers were generated by running four CPU-intensive processes (tightened loops manipulating 900 KB arrays) and four disk-intensive processes (FDS tasks for generating huge files in a loop). The average results of 5 experiments for the used struggeles methods are shown in Figure 2 (see in Appendix). Based on the obtained results, the LATE Scheduler methods out performed Hadoop native scheduler, Mantri, MonTool, and Dolly.

As it can be seen in the Figure 2, the results attained by the LATE Scheduler by Grep, is smaller than the results achieved by WordCount and Sort. This can be explained by considering the workload. WordCount and Sort write huge quantities of data across the network and to the computer. In the other side, Grep provides each reducer just a limited amount of bytes i.e, a count for each word. Therefore, it could be concluded that LATE Scheduler performed the best compared to other methods when Sort, Grep and WordCount are used with respectively 60%, 50% and 80%.

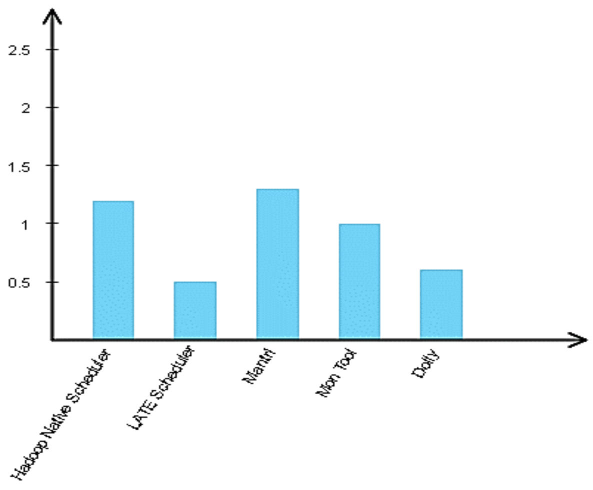
5. CONCLUSION AND FUTURE WORK

With the excessive growth in the information and data that produced by the internet of things (IoT), social media, multimedia and etc. applications, their analysis became a challenge and more complex due to the increasing volume of structured and unstructured data. MapReduce is a fault tolerant, scalable and simple framework for data processing that enables its users to process these massive amounts of data effectively. This work aimed to evaluate five stragglers identification methods: Hadoop native scheduler, LATE Scheduler, Mantri, MonTool, and Dolly using Sort, Grep and WordCount benchmarked methods. According to experimental results, LATE scheduler outperformed the other methods used in this work. We can conclude that LATE Scheduler would be efficient to obtain better results for stragglers identification. For the future work, machine learning methods can enable us to know the story behind the data, for instance, deep learning approach can be used to identify the proper nodes for running the straggler tasks and also it can provide us more information about the number of failures in different phases and correlation between different featuresto obtain more accurate results.

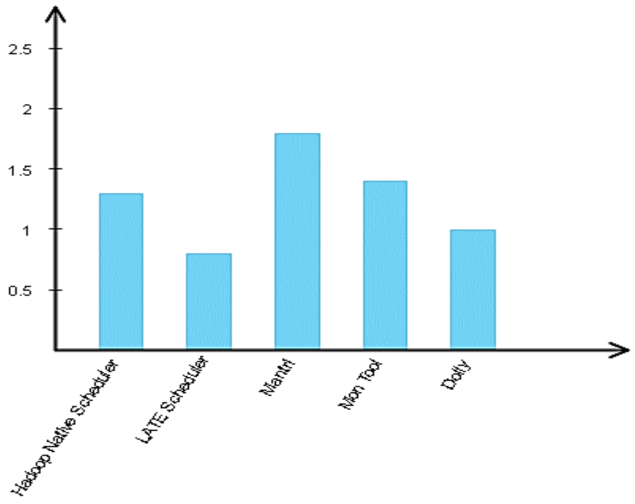
APPENDIX



(a)



(b)



(c)

Figure 2. Average task execution time using (a) Sort, (b) Grep, (c) WordCount

REFERENCES

[1] G. Ananthanarayanan, et al., “Scarlett: coping with skewed content popularity in mapreduce clusters,” *Proceedings of the sixth conference on Computer systems (EuroSys '11)*, pp. 287-300, 2011.
[2] I. A. T. Hashem, et al., “MapReduce scheduling algorithms: a review,” *The Journal of Supercomputing*, vol. 76, pp. 4915-4945, 2018.
[3] G. Ananthanarayan, et al., “Effective Straggler Mitigation: Attack of the Clones,” *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, pp. 185-198, 2013.
[4] P. D'Ancona and M. Reissig, “New trends in the theory of nonlinear weakly hyperbolic equations of second order,” *Nonlinear Analysis: Theory, Methods & Applications*, vol. 30, no. 4, pp. 2507-2515, 1997.
[5] R. Gandhi, et al., “Yoda: a highly available layer-7 load balancer,” *Proceedings of the Eleventh European Conference on Computer Systems (EuroSys '16)*, pp. 1-16, 2016.
[6] L. Zhao, et al., “Effective Straggler Mitigation with Cross-Layer Interference-Aware Optimization,” in *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 177-182, 2019.
[7] J. Alwidian and A. AlAhmad, “Hadoop MapReduce Job Scheduling Algorithms Survey and Use Cases,” *Modern Applied Science*, vol. 13, no. 7, pp. 38-48, 2019.
[8] A. Ghodsi, et al., “Choosy: max-min fair sharing for datacenter jobs with constraints,” *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys '13)*, pp. 365-378, 2013.
[9] S. N. Khezr and N. J. Navimipour, “MapReduce and Its Applications, Challenges, and Architecture: a Comprehensive Review and Directions for Future Research,” *Journal of Grid Computing*, vol. 15, no. 3, pp. 295-321, 2017.

[10] "OSDI (9th Usenix Symposium on Operating Systems Design and Implementation) [advertisement]," *IEEE Security & Privacy Magazine*, vol. 8, no. 4, pp. c4-c4, 2010.

[11] M. Zaharia, et al., "Improving MapReduce Performance in Heterogeneous Environments," *8th USENIX Symposium on Operating Systems Design and Implementation*, pp. 29-42, 2008.

[12] J. Rey, et al., "Efficient Prototyping of Fault Tolerant Map-Reduce Applications with Docker-Hadoop," *2015 IEEE International Conference on Cloud Engineering*, pp. 369-376, 2015.

[13] U. Kumar and J. Kumar, "A Comprehensive Review of Straggler Handling Algorithms for MapReduce Framework," Int. J. Grid Distrib. Comput., vol. 7, no. 4, pp. 139–148, 2014.

[14] Y. Chen, et al., "Design insights for MapReduce from diverse production workloads," Technical Report UCB/EECS-2012-17, EECS Dep., University of California, Berkeley, 2012.

[15] H. Zhou, Y. Li, H. Yang, J. Jia, and W. Li, "BigRoots: An Effective Approach for Root-Cause Analysis of Stragglers in Big Data System," *IEEE Access*, vol. 6, pp. 41966–41977, 2018.

[16] M. F. Aktas, et al., "Effective straggler mitigation: Which clones should attack and when?" *ACM SIGMETRICS Performance Evaluation Review*, vol. 45, no. 2, pp. 12-14, 2017.

[17] A. K. Abasi, et al., "A Text Feature Selection Technique based on Binary Multi-Verse Optimizer for Text Clustering," *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, Amman, Jordan, pp. 1-6, 2019.

[18] A. H. Katrawi, et al., "Earlier stage for straggle detection and handling using combined CPU test and LATE methodology," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 10, no. 5, pp. 4910-4917, 2020.

[19] F. Nzanywayingoma and Y. Yang, "Task scheduling and virtual resource optimising in Hadoop YARN-based cloud computing environment," *International Journal of Cloud Computing*, vol. 7, no. 2, pp. 83-102, 2018.

[20] M. Usama, et al., "Job schedulers for Big data processing in Hadoop environment: Testing real-life schedule with benchmark programs," *Digital Communications and Networks*, vol. 3, no. 4, pp. 260-273, 2017.

[21] M. N. Akhtar, et al., "Map-Reduce based tipping point scheduler for parallel image processing," *Expert Systems with Applications*, vol. 139, 2020.

[22] J. Lin and M. Lee, "Performance evaluation of job schedulers on Hadoop YARN," *Concurrency and Computation: Practice and Experience (CCPE)*, vol. 28, no. 9, pp. 2711-2728, 2016.

[23] M. Hanif and C. Lee, "Jargon of Hadoop MapReduce scheduling techniques: a scientific categorization," *The Knowledge Engineering Review*, vol. 34, 2019.

[24] Q. Chen, et al., "Libra: Lightweight data skew mitigation in mapreduce," *IEEE Transactions on Parallel & Distributed Systems*, vol. 26, no. 9, pp. 2520-2533, 2015.

[25] G. Liu, et al., "Sppartitioner: A novel partition method to handle intermediate data skew in spark streaming," *Future Generation Computer Systems*, vol. 86, pp. 1054-1063, 2018.