International Journal of Electrical and Computer Engineering (IJECE)
Vol. 10, No. 5, October 2020, pp. 4910~4917
ISSN: 2088-8708, DOI: 10.11591/ijece.v10i5.pp4910-4917

Earlier stage for straggler detection and handling using combined CPU test and LATE methodology

Anwar H. Katrawi[1], Rosni Abdullah[2], Mohammed Anbar[3], Ammar Kamal Abasi[4]
[1,3]National Advanced IPv6 Center (Nav6), Universiti Sains Malaysia, Malaysia
[2,4]School of Computer Sciences, Universiti Sains Malaysia, Malaysia

Article Info

Article history:

Received Oct 15, 2019
Revised Mar 17, 2020
Accepted Mar 30, 2020

Keywords:

Big data
Combinatory late-machine
Hadoop
Map reduce
Straggler

ABSTRACT

Using MapReduce in Hadoop helps in lowering the execution time and power consumption for large scale data. However, there can be a delay in job processing in circumstances where tasks are assigned to bad or congested machines called "straggler tasks"; which increases the time, power consumptions and therefore increasing the costs and leading to a poor performance of computing systems. This research proposes a hybrid MapReduce framework referred to as the combinatory late-machine (CLM) framework. Implementation of this framework will facilitate early and timely detection and identification of stragglers thereby facilitating prompt appropriate and effective actions.

Corresponding Author:

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

1. INTRODUCTION

A significant amount of data (big data) is stored and transferred online by thousands of companies, organizations, and individuals. This largely unstructured data is difficult to analyze and process using conventional database management tools which creates new challenges in the analysis and the storage of data [1]. Recently, there has been an increasing interest in key areas such as real-time data extraction, which reveals an urgent need for bulk and strict performance constraints. Consequently, the adaptation of huge data to implementations on distributed computing platforms is necessary. One way of doing this is to adopt and implement the popular programming model known as MapReduce [2]. Its success lies in simplicity, scalability, efficiency and extensibility that pushes the IT industry leaders such as Google, Yahoo, Facebook and Amazon to extensively adopt MapReduce as a powerful and reliable tool for Big Data processing. There are four factors, including processing, storing, visualization, and analyzing large data in modern organizations and enterprises. MapReduce can run the applications on a parallel cluster of hardware automatically. In addition; it can process terabytes and petabytes of data more rapidly [3,4]. 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 [5]. For instance, Google runs more than 10,000 distinct programs using MapReduce including graph processing[6], text processing, machine learning, and statistical machine translation. Moreover, the famous open-source Hadoop software framework for distributed storage and processing of big data Figure 1 sets uses Mapreduce as a central tool to split the data, process it and make it not only manageable but also available for users' consumption or further processing. Also, the Hadoop MapReduce environment provides fault-tolerant solutions in case of hardware failures or software errors during the execution of tasks [7].

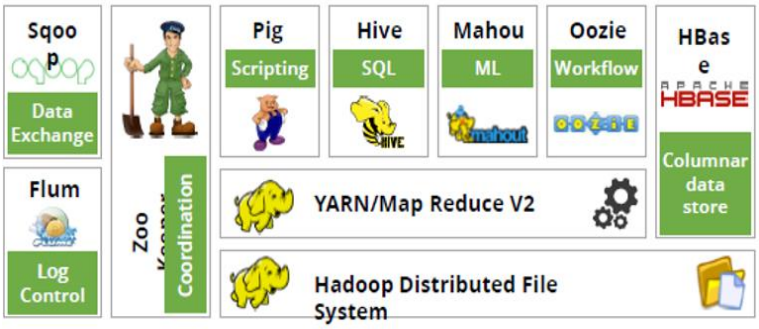

Figure 1. Hadoop framework

According to the work presented by [8], Hadoop MapReduce has the ability to tolerate several types of faults and they are as follows:

a. Nodes failure: A node in a MapReduce cluster may fail at any time. In this case, the JobTracker removes this slave node from the list of nodes available and re-executes the tasks on other nodes. It can be concluded that a node is declared failing if at least one task launched on it has failed. At the time, the JobTracker checks if the node in question should not be blacklisted. If a slave node is "blacklisted", the JobTracker will no longer assign it map or reduce. It can be removed from this list if his behavior becomes normal and does not commit faults during a certain time interval.
b. Software Failure: A task may stop because of an error or exception in the mapping or reduction program. In this case, the JobTracker orders the re-execution of the failed task to a limited number of attempts (four by default), beyond which the task and the job of the task is considered faulty.
c. Stopped Task: As an example, the process of running a task may stop unexpectedly due to a transient bug in the underlying virtual machine. In this case, the JobTracker show in Figure 2 will be notified and it will reorder the job as described above.
d. Blocked Task: It's considered faulty if after some timea mapping or reduction task remains blocked without any progress; in this case, the JobTracker orders to kill the process running this task.
e. Delayed Tasks:When some tasks are unexpectedly taking longer execution time compared with the average execution time, these tasks called stragglers.

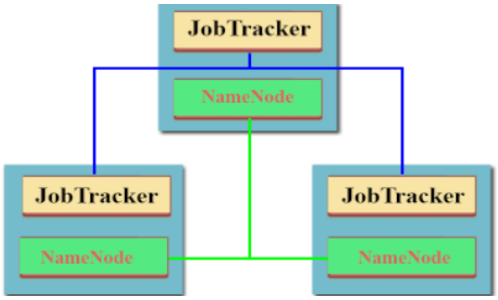

Figure 2. Job tracker

Several studies have been devoted to improving the tolerance of systems to faults. For instance, Microsoft reveals that when the CPU and core parts' errors occur in one million customers' computers, Hadoop does not have the capacity to deal with these types of potential errors. This also includes other types of errors related to the task itself (when one or more of these tasks take longer or stops before the work outcomes are realized).

Problem statement. The delayed tasks are called stragglers and play a key role in increasing the execution time of big data and energy consumption. Following the MapReduce framework, stragglers refer to the tasks that take a longer time to be executed compared to other tasks. There are various techniques of detecting and handling stragglers such as Dolly, the Hadoop native scheduler, MonTool, LATE and Mantri. Regardless of the techniques already in place, straggler detection remains problematic in the field of data analytics.

The proposed solution. We propose an algorithm that facilitates the calculation of straggler tolerance threshold using CPU test and LATE methodology. Our approach considers crucial issues such as the QoS with a major focus on the timing constraints, the progress of task execution, and the usage of cluster resources. Even though task selection does not appear to be a big problem in the initial stages, we have been able to show that it is a big issue that requires close attention. Therefore, we recommend identifying those tasks that lead to the longest response times. We also recommend that this needs to be done as early as possible so that there could be no later surprises. Taking these into account, it is possible to see that our LATE methodology is based on the estimate of the time that is left with the goal of early detection of the tasks that are running slowly. In summary, the methodology is based on making decisions early, using the finishing times and not the progress rates, not assigning speculative tasks to slow nodes and optimization of resource utilization.

Significance of the research. In the current study, a method is proposed with the purpose of addressing the straggler problem. We propose an algorithm which can enable calculation of threshold of straggler tolerance earlier using CPU test and Longest Approximate Time to End (LATE) methodology. Our approach focuses on the timing constraints, progress of task execution, and the usage of cluster resources. In this work, our major contribution is that unlike other studies that assume that it is hard to have a correlation between task execution and node status, we show that it is possible and feasible to have a correlation and detect stragglers using LATE methodology and CPU test. Our methodology is simple enough and easy to accommodate with low overheads.

Organization of the research. The rest of this paper is organized as follows: Section 2 covers the literature review, section 3 covers the methodology, section 4 the results and discussion, section 5 covers the proposed solution/recommendation, and section 6 covers the conclusion.

2. LITERATURE REVIEW

Hadoop and Map Reduce are among the most commonly used frameworks when it comes to task execution across several nodes for optimal performance. Even though the frameworks have become popular, they still face several challenges when it comes to the effectiveness of tasks execution [8]. Specifically, achieving predictable execution has become problematic because of stragglers. Due to stragglers, tasks execution takes longer to complete than originally anticipated [9]. Such delays are undesirable because they result in reduced service performance and can also potentially violate QoS (Quality of Service) requirements concerning time taken to complete tasks. For service providers, tasks that take more time to complete lead to reduced availability of systems and cause jobs to consume more time. Stragglers have become common especially in cloud data centers [10].

Therefore, it is vital to detect and mitigate them promptly. Additionally, centers with large computing infrastructure can also experience delays that can lead to ineffective job execution. Also, large data centers have a high intake of service creation which make them vulnerable to stragglers. There are several root causes to stragglers including resource contention, hardware heterogeneity, background network traffic, and operating system related-level causes [11]. Considerable effort has been made to study stragglers. Over the years, the size of computing infrastructure and jobs executed have continued to grow which has dramatically increased the impact of stragglers. Stragglers are known to extend job execution substantially which negatively affects the "Consumer Service Level Agreement" and QoS performance requirements. In a study conducted by Bortnikov, Frank, Hillel, and Rao, the authors propose two ways of dealing with stragglers namely tolerance and avoidance [12]. However, avoiding stragglers is difficult since it is impractical to pursue. Therefore, straggler tolerance is the approach adopted by most stakeholders. In straggler tolerance, the execution progress of a task is monitored using a percentage score made up of values ranging from 0 to 1 which represent start and completion. Currently, the approaches used for straggler detection can either be described as offline or online analytics [13].

Nonetheless, it is worth noting that online detection can occur too late during the execution cycle of a task. Therefore, stragglers cannot be prevented from running slower even after the implementation of speculative copies. On the other hand, offline approaches are normally applied to avoid stragglers. This approach is seen as less feasible and thus it is uncommon. However, better results can be achieved by combining both online and offline approaches. When used together, they can significantly help to improve the effectiveness of "straggler detection".

2.1. Related works

For Straggler Detection, many techniques have been developed. Ouyang et al. proposes an algorithm based on the progress score of task execution that enables dynamic threshold detection for straggler tasks [14]. This strategy has improved performance significantly by reducing job execution by 44 percent.

In a study conducted by Zaharia et al., the authors propose a new Straggler Detection that takes into account both the progress score and elapsed time with the objective of improving the progress score strategy [15]. The authors determine that the strategy does not have the capacity of determining how fast a task runs among different tasks starting at different times hence the need for improvement. Dean and Ghemawat have adopted a technique called Speculative Execution in which they launch copies of the straggler on alternative nodes with the aim of improving performance [16]. Google acknowledges that Speculative Execution improves job execution by 44 percent. However, this technique can reduce the overall throughput due to the duplication of tasks. Therefore, some Hadoop administrators prefer not to use the Speculative Execution option [17, 18]. Yanfei et al proposed another technique for Straggler handling which consists of ending the delayed tasks and reassigning them to another node without stragglers, however, Guo, Rao, Jiang, and Zhou disagrees with this technique and argue that it results in wastage of resources and therefore increases energy consumption [19].

In a similar study, Zhou, Li, Yang, Jia, and Li propose a technique known as "BigRoots" which involves incorporating both system features and framework for the analysis of root causes of stragglers especially in big data systems [20]. The authors establish that "BigRoots" is effective when it comes to identifying the "root causes" of stragglers which can significantly help in optimizing performance. An examination by Phan attempted to come up with "energy efficient straggler mitigation" technique for big data applications especially in the cloud environment [21-23]. The framework employed by the author takes into account how heterogeneity of resources affect the performance and energy consumption of big data applications.

In another studies by Harlap et al., Kim, W., the authors sought to solve the straggler problem for parallel ML [24, 25]. The authors combined a more "flexible synchronization model" together with the experiments involving real straggler behaviors and synthetic straggler behaviors to come up with near-ideal run times across all the straggler patterns they tested. Similarly, Yadwadkar et al. came up with a framework called "Wrangler" which could predict when stragglers were going to occur and aid in making scheduling decisions [26]. The formulations employed by the authors captured the shared structure in their data so that it could improve the generalization performance of their data.

3. METHODOLOGY

As noted earlier, we estimate the time remaining for each task based on the process score derived from Hadoop. In practice, this heuristic works well. However, we want to point out that there are incidences when it can backfire. When this happens, the heuristic can provide incorrect estimates and give results that a task launched later finishes earlier. To demonstrate the delay, we assume that the progress of a task grows by five percent during the first phase. We assume that during this first phase, the total score is fifty percent and that the rate reduces by one percent in the second phase. In the first phase, it is expected that the task will take ten seconds and fifty seconds in the second phase to produce a total of sixty seconds. When two copies of the same task are launched at the same time, the first task is denoted by T1, the second is noted by T2 and the first task starts at time 0 while the next starts after ten seconds. The progress rate is checked after twenty seconds whereby twenty seconds, it is expected that T1 will have finished the first phase and will be through a fifth of the second phase. Therefore, it will have a progress score of sixty percent. Its rate of progress will be 60%/20s=3%/s. On the other hand, T2 will just be through with the first and its score will be 50%. Its rate will, therefore, be 50%/10s=5%/s. The estimated time remaining for T1 will be (100%-60%)/(3%/s)=13.3s. For T2, the estimated time left will be (100%-50%)/(5%/s)=10s. Therefore, the heuristic will illustrate that T1 will take a longer time to run compared to T2. However, in reality, T2 will finish second compared to T1. We also determined the criteria that we could use to identify stragglers. We took the finish time for every task to be represented by (1).

$$EFT_j^i = t_k + \frac{1-PS(task_j^i)}{PS(task_j^i)}(t_k - t_0) \quad (1)$$

In this instance, EFT_j^i represents the estimated finish time. PS is used to represent the progress score for a $task_j^i$ whereas t0 is starting time while tk is the timestamp recording for $PS(task_j^i)$. Our proposed LATE methodology takes into account the scope of data and the speed of processing data. By taking these into account, we can determine the pattern of straggler detection and correlate this with the attributes of a system normally hypothesized to give them the ability to cause stragglers. Some of the attributes include resource utilization (memory, CPU, disk), hardware faults, unhandled requests, among others. When we

combine these runtime features with dynamic information, we can come up with better prediction and learning models that can discriminate stragglers promptly and lower the number of failures occurring due to late timing.

4. RESULTS AND DISCUSSION

This research has identified the need for improving the effectiveness of MapReduce processes to facilitate costs reduction and maximum utilization of resources. Performance improvement can be achieved by eliminating inefficiencies brought about by the existence of stragglers. Stragglers have the effect of resulting in poor user code and uneven aggregation of workloads. Poor user code is the product of looping conditions that are designed ineffectively and uneven aggregation of workloads results from extreme co-allocation of workloads due to inefficient scheduling. In the case of MapReduce tasks with a massive number of write and read queries, it is common for file requests to be overloaded thereby leading to inefficiency in handling the requests. It is notable that once the threshold of the master node is surpassed, stragglers set in which implies that the waiting queue for requests becomes long. If the requests continue increasing, then the master node becomes overloaded thereby further slowing down the handling process of the requests. This experiment focused on investigating how the occurrence of stragglers is affected by contention of resources. An observation of the occurrence of stragglers that took place over a period of 20 days yielded the results presented in Figure 3.

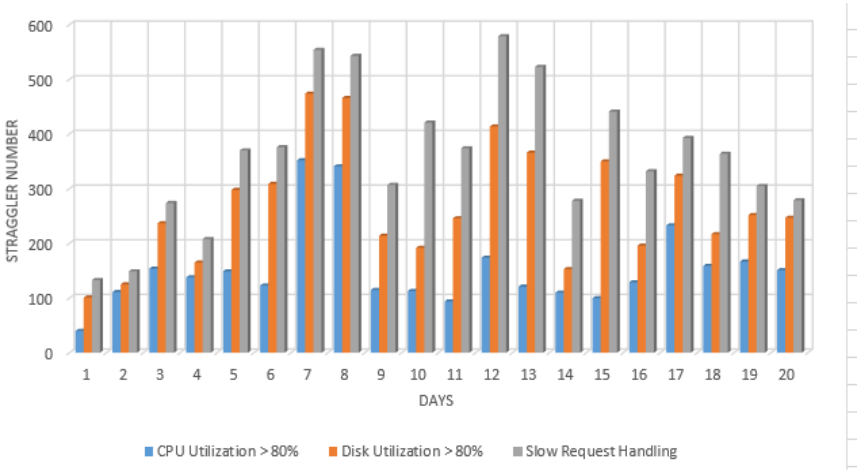

Figure 3. Graphical presentation of the findings

In the analysis, millions of tasks processed in five hundred servers in Cloud Datacenter [27] were investigated. The inclusion criteria involved:

a. Servers whose tasks had DoS-Index values greater than or equal to 10
b. Servers whose utilization of CPU was greater than or equal to 80%
c. Latency from file processing greater than 400ms which translates to slow handling of write and read requests

From the investigation, it came to our attention that 42% of stragglers are brought about by overloading of disks while 59% of stragglers exist under high server CPU conditions. The findings also revealed that slow handling of requests was responsible for the occurrence of 34.3% of stragglers. From the findings, it is evident that the existence of stragglers is significantly caused by high utilization of resources. It was also observed that stragglers can be caused by other factors such as the conditions of the network.

5. RECOMMENDATION/PROPOSED SOLUTION

In this paper, we propose a combined strategy called COMBINATORY LATE-MACHINE that tests the CPU machine. The CPU machine is tested to determine its vulnerability to stragglers. To achieve this, the CPU and the RAM are tested to determine machines that have a CPU usage less than 85. Those found to meet this condition are dropped and the ones that exceed 85 are selected and placed in a performance chart

starting with those that have the highest performance. If for any reason the CPU/Memory is lower than a certain threshold, the jobs will automatically be redirected to another machine with a higher CPU/Memory performance. This stage is run once and helps in creating a list of performances for all the machines. After this step, Straggler Detection and handling are initiated at the same time using LATE algorithm. The algorithm has several benefits that make it suitable for heterogeneous jobs since it re-executes only the slowest tasks. The primary advantage of the proposed methodology is the increasing probability to detect the straggler machines and straggler tasks with the same algorithm in earlier stages. Our LATE scheduler is designed such that it includes all the features needed for it to function well in a realistic environment. The major insight behind the LATE algorithm is that tasks believed to finish last can be executed at any time in the future because this is the best way through which the response time can be improved. The framework of the proposed combinatory Late-Machine strategy is illustrated in Figure 4.

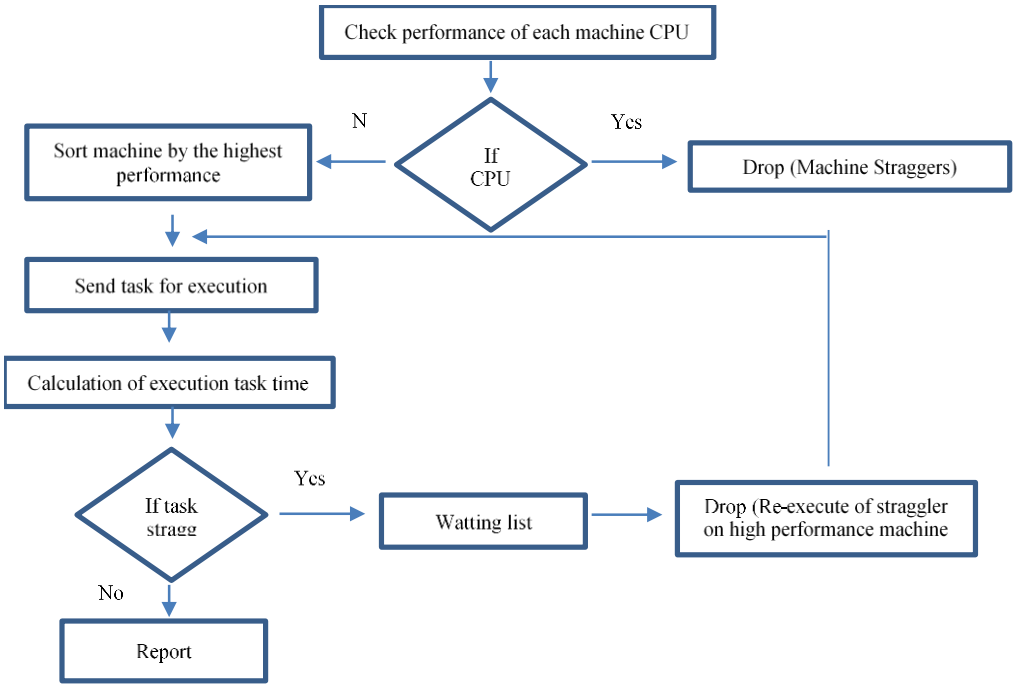

Figure 4. Proposed combinatory late-machine framework

5.1. Limitations and future research

Our observation did not focus on the conditions of the memory capacity as well as overlapping of conditions. Therefore, this warrants the need for further investigations.

6. CONCLUSION

In this paper, we proposed the Combinatory Late-Machine (CLM) strategy to identify (in earlier stages) the straggler for both nodes and tasks, and the optimal node for re-execution of slow tasks. The overall execution time is improved significantly using this technique compared to traditional job schedulers. In the future, the CLM strategy will be tested with Hadoop to evaluate the efficiency of this technique. The findings provide new insights into early straggler detection. However, the current strategy has its shortcomings and there is a need for exhaustive analytics to establish the relationship between stragglers and the contention of resources. The proposed algorithm could be used for research as well as in the industry to improve the time and cost for big data processing.

ACKNOWLEDGMENT

The authors would like to thank all the research participants for their time, effort, and contribution to the research

REFERENCES

[1] E. A. Mohammed, et al., "Applications of the MapReduce programming framework to clinical big data analysis: current landscape and future trends," *Bio Data Mining*, vol. 7, no. 1, pp. 22-44, 2014.

[2] S. Valvåg, et al., "Cogset: a high performance MapReduce engine," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 1, pp. 2-23, 2012.

[3] Sumalatha, S., and Subramanyam, R. B. V. "Distributed mining of high utility time interval sequential patterns using mapreduce approach," *Expert Systems with Applications*, vol. 141, 112967, Mar. 2020.

[4] Meddah, I. H., and Belkadi, K. "Parallel Distributed Patterns Mining Using Hadoop MapReduce Framework," *International Journal of Grid and High Performance Computing*, vol. 9, no. 2, pp. 70-85, 2017.

[5] S. 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.

[6] Prameela De'vi. Chillakuru, T. Kumanan, CH. Sarada Devi, "Content based Retrieval Management Systems in Web Engineering," *International Journal of Recent Technology and Engineering (IJRTE)*, vol. 8, no. 2S11, pp. 81-93, Sep. 2019

[7] A. Y. Pigul, "Comparative Study Parallel Join Algorithms for MapReduce environment," *Proceedings of the Institute for System Programming of RAS*, vol. 23, pp. 285-306, 2012.

[8] I. Hashem, et al., "MapReduce scheduling algorithms: a review," *The Journal of Supercomputing*, 2018.

[9] K. Mitsuzuka, et al., "Proxy Responses by FPGA-Based Switch for MapReduce Stragglers," *IEICE Transactions on Information and Systems*, vol. 101, no. 9, pp. 2258-2268, 2018.

[10] J. Rogoff, "Stragglers," *Sewanee Review*, vol. 124, no. 3, pp. 397-397, 2016.

[11] M. F. Aktas, et al., "Straggler Mitigation by Delayed Relaunch of Tasks," *ACM SIGMETRICS Performance Evaluation Review*, vol. 45, no. 2, pp. 248-248, 2018.

[12] E. Bortnikov, et al., "Predicting execution bottlenecks in map-reduce clusters," *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Computing*, pp. 1-18, 2012.

[13] A. K. Abasi, et al., "Link-based multi-verse optimizer for text documents clustering," *Applied Soft Computing*, vol. 87, 2019.

[14] X. Ouyang, et al., "Straggler Detection in Parallel Computing Systems through Dynamic Threshold Calculation," *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, pp. 414-421, 2016.

[15] J. Xie, et al., "Improving MapReduce performance through data placement in heterogeneous Hadoop clusters," *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, pp. 1-9, 2010.

[16] J. Dean and S. Ghemawat, "Map Reduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, p. 107, 2008.

[17] H. Wu, et al., "A Heuristic Speculative Execution Strategy in Heterogeneous Distributed Environments," *2014 Sixth International Symposium on Parallel Architectures, Algorithms and Programming*, pp. 268-273, 2014.

[18] 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.

[19] Y. Guo, et al., "FlexSlot: Moving Hadoop Into the Cloud with Flexible Slot Management," *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014.

[20] H. Zhou, et al., "BigRoots: An Effective Approach for Root-Cause Analysis of Stragglers in Big Data System," *IEEE Access*, vol. 6, pp. 41966-41977, 2018.

[21] T. Phan, et al., "A New Framework for Evaluating Straggler Detection Mechanisms in MapReduce," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, vol. 4, no. 3, pp. 1-23, 2019.

[22] Chaowei Yang, Qunying Huang, Zhenlong Li, Kai Liu & Fei Hu, "Big Data and cloud computing: innovation opportunities and challenges," *International Journal of Digital Earth*, vol. 10, no. 1, pp. 13-53, 2017. DOI: 10.1080/17538947.2016.1239771

[23] L. Greeshma, Pradeepini Gera, "Big Data Analytics with Apache Hadoop MapReduce Framework," *Indian Journal of Science and Technology*, Vol 9, no. 26, July 2016.

[24] A. Harlap, et al., "Addressing the straggler problem for iterative convergent parallel ML," *Proceedings of the Seventh ACM Symposium on Cloud Computing - SoCC '16*, pp. 98-111, 2016.

[25] Kim, W., Kim, Y., and Shim, K., "Parallel computation of k-nearest neighbor joins using MapRe-duce," In *Proceedings of the IEEE International Conference on Big Data*, pp. 696-705, 2016.

[26] N. Yadwadkar, et al., "Multi-Task Learning for Straggler Avoiding Predictive Job Scheduling," *Journal of Machine Learning Research*, vol. 17, pp. 1-37, 2016. [Online]. Available: http://jmlr.org/papers/volume17/15-149/15-149.pdf.

[27] Gigas the cloud computing company. [Online]. Available: https://gigas.com/en/cloud-datacenter.

BIOGRAPHIES OF AUTHORS

Anwar H. Katrawi received B. Sc. In computer science from al Mustansiriya university, Iraq and M.Sc. in the computer information system from Arab Academy for Management, Banking and Financial Sciences of Jordan. He is currently a PhD Candidate in the school of Computer Sciences (NAV6) at Universiti Sains Malaysia. His research interests include Big data, data warehouse, machine Learning and data analytics.

Rosni Abdullah is a professor in parallel computing and one of the national pioneers in the said domain. She was appointed Dean of the School of Computer Sciences at Universiti Sains Malaysia (USM) in June 2004, after having served as its Deputy Dean (Research) since 1999. She is also the Head of the Parallel and Distributed Processing Research Group at the School since its inception in 1994.

Mohammed F.R. Anbar received his bachelor of Computer System Engineering from Al-Azhar University, Palestine and M.Sc. in Information Technology from Universiti Utara Malaysia, Malaysia (UUM). He obtained his PhD. in Advanced Internet Security and Monitoring from University Sains Malaysia (USM). He is currently a senior lecturer at National Advanced IPv6 Centre (NAv6), Universiti Sains Malaysia.

Ammar Kamal Abasi received B. Sc. in computer information system from Jordan university of science and technology, and M.Sc. in the international business from the university of Jordan. He is currently a PhD Candidate in the school of Computer Sciences at Universiti Sains Malaysia. His research interests include evolutionary algorithms, nature-inspired computation, and their applications to optimization problems.