Straggler Mitigation in Hadoop MapReduce Framework: A Review

Lukuman Saheed Ajibade[1], Kamalrulnizam Abu Bakar[2]
School of Computing
Universiti Teknologi Malaysia
Johor Bahru, Malaysia

Ahmed Aliyu[3]
Dept. of Mathematics
Bauchi State University Gadau
Nigeria

Tasneem Danish[4]
Systems and Computer Engineering Dept.
Carleton University Canada

***Abstract*—Processing huge and complex data to obtain useful information is challenging, even though several big data processing frameworks have been proposed and further enhanced. One of the prominent big data processing frameworks is MapReduce. The main concept of MapReduce framework relies on distributed and parallel processing. However, MapReduce framework is facing serious performance degradations due to the slow execution of certain tasks type called stragglers. Failing to handle stragglers causes delay and affects the overall job execution time. Meanwhile, several straggler reduction techniques have been proposed to improve the MapReduce performance. This study provides a comprehensive and qualitative review of the different existing straggler mitigation solutions. In addition, a taxonomy of the available straggler mitigation solutions is presented. Critical research issues and future research directions are identified and discussed to guide researchers and scholars.**

***Keywords*—Big data; blacklisting execution; Hadoop; MapReduce; spark; speculative execution; straggler**

I. Introduction

Due to the accelerated expansion of structured and unstructured data generated by the internet of things (IoT), social media, multimedia, etc., it is becoming increasingly difficult to analyse the information and data that is being generated. Applications like MapReduce, a fault-tolerant, scalable, and user-friendly framework for data processing, allow their users to efficiently process these enormous volumes of data [1], [2] . The preparation and generation of a large amount of data can be accomplished using the MapReduce approach. This is because it provides a user-friendly environment and provides solutions for a variety of ad hoc and misses, including data sorting and web indexing, among others. Bigger businesses, including Yahoo and Google, among others, use MapReduce in their large information applications.

The variety in accessibility in the CPU, I/O conflict, or network traffic is what causes stragglers. The MapReduce Framework is complete once map and reduce have been finished [3], [4]. The job is not finished in the MapReduce framework until the very reduce and map tasks are finished. Additionally, when the range of time employment increases [5-8], the number of stragglers decreases. Some compute nodes are quicker than others in a diverse environment. Faster compute nodes will finish their work ahead of schedule and wait for the stragglers to complete. Slower compute nodes are known as stragglers (Fig. 1). Nodes can occasionally fail owing to hardware or software issues. To prevent system performance degradation, it is crucial to identify stragglers at an early stage.

Traditional database management solutions such as E-R model are no longer suitable for processing and analysing of massive amounts of data generated by today's big enterprises. The bulk processing problem has become a major difficulty, and its analytical tools are evolving quickly because of Google's creation of MapReduce, which enabled millions of users to locate material from millions of pages in less than one tenth of a second. On the other hand, stragglers are widely acknowledged as a significant bottleneck in the processing of large amounts of data and they can have a considerable effect on it. Some stragglers mitigation techniques are evaluated in this paper.

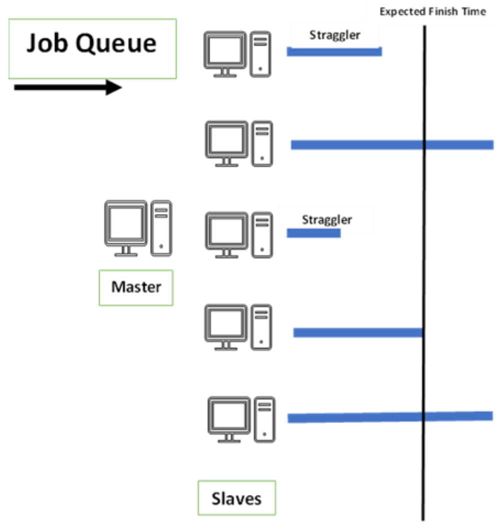

Fig. 1. Straggler Nodes in Parallel Processing.

II. MapReduce Framework and Stragglers

For significant information preparation on bunch-based figure designs, MapReduce is the ideal matching information preparing model that has been suggested [5]. Inside server centres, this system is utilised to support machine learning, data mining, and search applications. Large-scale online search applications must be addressed by the philosophy. Google was the one that originally recommended handling extremely large-scale online search applications.

Programmers are granted licences to extricate themselves from issues like parallelization, booking, and allocating so they may concentrate on creating applications. Processing, storing, visualising, and interpreting big data are the four main components of contemporary businesses and organisations. Applications on a parallel hardware cluster can be automatically run by MapReduce. Terabytes and petabytes of data can also be processed more quickly.

Due to the MapReduce capability to offer a highly effective and efficient framework for the parallel execution of applications, data allocation in distributed database systems, and fault tolerance network connections, it has recently grown in popularity in a variety of applications. Parallel map assignments, as shown in Fig. 1, are carried out as a single input data set consisting of a collection of "key value" sets that are further divided into *fixed produce* and *size blocks* transitional output. Information preparation tools called *map* and *reduce* are included in the MapReduce programming model as depicted in Fig. 2.

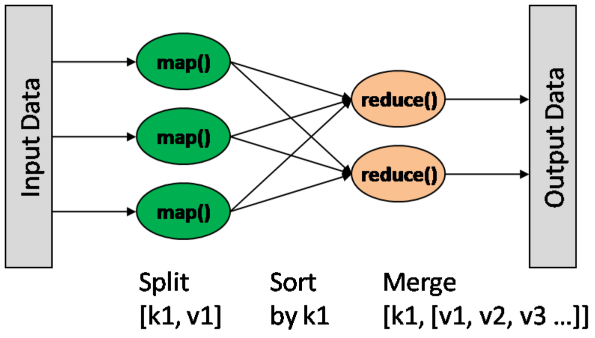

Fig. 2. MapReduce Phases.

When a user submits a job request during the Map-phase, the tasks are mapped to commodity machines for execution. In the Reduce phase, the Combiner lowers the network's data transmission rate. The Reduce-phase includes the Sort or Merging step. The time is utilised to combine the Map outputs from various nodes, and this combining is referred to as the Reduce time. The final stage in running the operation in a MapReduce fashion is the Reduce-part. The impact of each step in this process on runtime varies, so different weights should be used to estimate each job's completion time (the impact of each step on the execution process is determined by the ratio of the time of each step to the overall process's runtime).

The tasks that require more time to complete than comparable tasks are known as stragglers. There are many reasons for delaying the assignment, including the use of inefficient machines, the amount of information to process, framework obstructions, equipment heterogeneity, and competition for the available resources [6], [7].

Additionally, if one task is running slowly on a particular system, it is not important for other upcoming and current tasks to execute slowly on that same machine. Three main mechanisms must be kept in mind when addressing the straggler issue:

- If it is discovered that the anticipated remaining time is longer than the typical runtime, the procedure may be resumed up to three times.
- A speculative duplicate is scheduled if the resource measurement lowers unfavourably. The following procedures estimate the expected remaining time (trem) and the typical runtime (tnew), as shown.
- term = (telapsed * d/dread) + twrapup.
- tnew = processRate∗locationFactor∗d+schedLag.

Stragglers can occur for a variety of reasons, such as load inequality, ineffective scheduling, data localization, communication overheads, and hardware heterogeneity [8], [9]. Additionally, there have been initiatives to address one or more of these worries to lessen the issue [10]–[12]. Even if all these earlier attempts were significant and helpful in solving this issue, more rigorous analytical techniques are required to fully comprehend the effects of stragglers on the performance slowdown in huge data [13], [14].

III. Related Works

For addressing data skew for joins in a MapReduce system and avoiding stragglers, the SharesSkew algorithm was suggested by [7]. When data is skewed, the method determines the multi-way join in MapReduce. In essence, the method divides up the work of performing multi-way joins and maximises the amount of information transferred from the Mappers to the Reducers.

The technique uses a modified version of the SharesSkew algorithm to partition and share highly valued records in a distinctive way to minimise communication costs. The algorithm determines the heavy hitter value of an attribute based on the sizes of the relations or the portion of the connection with heavy hitters and how the sizes interact with one another.

In contrast to existing techniques that limit the number of Reducers employed, the SharesSkew approach merely limits the number of tuples of each Reducer. As a result, the number of tuples selected ensures that the data is distributed equally among the Reducers. (For determining the parameters of the proposed approach, both chain and symmetric joins are taken into consideration.).

A dynamic skew mitigation approach called SkewTune in MapReduce applications was proposed by [15]. The SkewTune approach tries to address the following challenges: i) the MapReduce system should not require extra input from user ii) the system should be fully transparent and iii) there should be minimal overhead even when there is no skew. If the node in the cluster is idle, the SkewTune recognizes the task with the highest anticipated remaining processing time. Afterwards, the non-processed input data of the straggling task is proactively re-partitioned such that it fully utilizes the nodes within cluster. It then conserves the ordering of the input data for the original output to be re-built by concatenation. The SkewTune is implemented as an extension to Hadoop and the efficiency is evaluated by employing several actual applications.

In the quest to address the problem of load imbalance due to data skew, a load balancing based on join algorithms in MapReduce systems was proposed by [16]. The load balancing algorithm named Fine-Grained partitioning for Skew Data (FGSD) for reduced tasks. The FGSD employs the properties of both output and input data via a proposed stream sampling algorithm. In addition, FGSD provides an approach that distributes the input data that help in handling efficient redistribution and join product skew.

Consequently, the authors declare that FGSD achieved better balancing of data distribution and minimizes execution time of jobs with different degrees of data skew. FGSD does not need any alteration to the MapReduce configuration and is suitable for handling complex jobs. Similarly, Gavagsaz et al. [17] focus on reducer phase to achieve load balancing in MapReduce system by employing scalable straightforward random sampling. The major problem in reducer phase is data skew, which lead to a significant load imbalance and degradation of performance. Therefore, a sorted balance algorithm was proposed, which is centered on sampling results. The Sorted Balance algorithm using SCalable random sampling (SBaSC). The scalable sampling algorithm is employed for monitoring a more exact approximate distribution of the keys through sampling small fraction of the intermediate data.

In MapReduce, reducer side data skew occurs due to unbalanced allocation of intermediate map-output to reducers. Therefore, [18] proposed an adaptive Learning Automata Hash Partitioning (LAHP) algorithm to address the data skew problem. The LAHP is based on learning automata game for conventional allocation of intermediate key-value pairs to designated reducers. It is achieved by setting a learning automaton on each mapper node to control the allocated load on each reducer. Thus, during execution of job, a learning automata game is enabled.

In addition, the LAHP algorithm partitions the intermediate key value pairs arbitrarily without considering the statistical distribution of pre-processing and input data. A load balancing mechanism that enhances MapReduce in Hadoop was proposed for mitigating negative impact of data skew on the performance of MapReduce [19]. Data skew has become a typical problem in MapReduce processing for handling data intensive applications. The mechanisms integrate Reservoir Sampling and Greedy (RSG) algorithms. It further slots in the concept of data locality in order to properly distribute the workload of each reducer, which is based on priority-based load-balancing mechanism (PLBM).

Wang *et al.* [20] proposed an enhanced Replication Framework of Stragglers over a Large-scale Parallel processing (RFSLP) for addressing the latency Framework of Stragglers encountered due to replication of stragglers. The framework analyzes replication latency-cost tradeoff and determines the best replication strategy. The strategy considers three design ideas including i) how many replicas are required ii) the time to replicate straggling tasks and iii) whether to terminate main copy or not. The framework analysis demonstrates that for specific execution time allocation, a small quantity of task replication can drastically minimize the cost of computing resources and latency. Further, an algorithm that estimates cost and latency based on the empirical allocation of task execution period.

In another aspect, a Framework for Assessing Stragglers Detection (FASD) mechanisms over MapReduce was proposed by [21]. It focuses on detection of stragglers because most of the existing works are centered on mitigating stragglers. In this light, an all-inclusive framework for straggler detection and mitigation was proposed. The detection strategy considers set of metrics that can be employed for characterizing and detecting stragglers. The metrics include fake positive, recall, detection latency, precision, and undetected time. Further, an architectural model was developed in such a way that the metrics can be linked to determine performance. The performance measure includes system energy overhead and execution time. To demonstrate those metrics that are effective in detecting stragglers and predict effectiveness in terms of energy efficiency performance, a number of experiments were conducted.

Similarly, a data partitioning concept, which is based on intermediate node for mitigating skew over a spark computing environment was proposed [22]. The main issue targeted in this work is unbalanced partitioning, which leads to variation in the amount of data processed by each Reducer task. Considering the mentioned issue, a Spark Key Reassigning and Splitting Partitioning (SKRSP) algorithm for handling the partition skew from the source codes of Spark-core 2.11 has been developed. The concept considers two approaches of balancing namely: partition balance for intermediate data and partition balance after shuffle operators. The contribution is in two folds first, a Key Reassigning Harsh-based Partitioning (KRHP) and rang-based Key Splitting Reassigning Partitioning (KSRP) algorithms. These algorithms can create suitable strategy for implementing the skew in the shuffle phase. The KSRP creates a weighted bound for partitioning intermediate data for the kind of sort-based applications. While KRHP stores these reassigned keys, and the new reducers of these keys are from other applications.

A proactive method named Hummer-1 for mitigating stragglers based on partial clones was proposed by [23]. In the existing solutions, different methods have been suggested including speculative execution, blacklisting and proactive mitigation. However, these solutions either waste much resource or consume much time during execution. The Hummer method trigger clones just when jobs are submitted thus, tasks in one job are assigned with clones to reduce stragglers. The initial default number of clones for a single task is three, which has been found to be the best value since there exist variations among nodes in the cluster [23]. To further improve Hummer-1, Hummer-2 was introduced which uses cloning for only tasks with high-risk delay. The authors claim that the Hummer method consumed fewer resources and minimize job delay that is, job completion time is reduced.

A Dynamic Server Blacklisting (DSB) framework was proposed for lessening stragglers to evade Quality of Service violation for time-sensitive applications [24]. The straggler task may occur due to one or more of the following reasons: heterogeneous hardware configuration, resource contention and so on. Straggler task could become severe due to increased complexity and system scale.

The DSB is developed based on the two prominent concepts namely speculative execution, which is automatic/dynamic and blacklisting, which is manual configuration. DSB considers the previous, which is historical and present behavior of server node to improve straggler mitigation efficiency. The computing servers are ranked at a given time interval according to their present performance in completing jobs instead of their physical facilities. The servers with worst performance are momentarily blacklisted dynamically. Thus, due to the strategy no new replications/tasks are allotted to those straggler-prone nodes in each time window. DSB further offers an alternative API in such a way that the top worst nodes are blacklisted based on their ranking. An optimal node is examined as a trade-off between straggler mitigation efficiency and capacity loss. Table I contains the comparison of existing load balancing solutions.

TABLE I. COMPARISON OF LOAD BALANCING SOLUTIONS FOR STRAGGLER MITIGATION

Existing Solutions	Comparisons						Remark
	Heterogeneity among Network Node	Data Processing Method	Priority-based Scheduling	Mitigatio0n Approaches			
				Speculative	Blacklisting	Proactive	
ShareSkew [7]	No	MapReduce/Hadoop	No	Yes	No	No	Multi-way joins have not been considered to investigate an efficient multi-round MapReduce algorithm.
SkewTune [8]	Yes	MapReduce/Hadoop	No	Yes	No	Yes	SkewTune approach may lead to resource contention due to high computation.
FGSD [9]	No	MapReduce/Spark	No	Yes	No	No	Similarity joins has not been considered in the Fine-grained skew data method
SBaSC [10]	No	MapReduce/Spark	No	Yes	No	No	The query level load balancing and fairness need to be optimized
LAHP [11]	Yes	MapReduce/Hadoop	No	Yes	No	No	Data skew could also occur when the sizes of the keys are different and affect the shuffle time.
PLBM [12]	No	MapReduce/Hadoop	Yes	Yes	No	No	The transfer cost is only based on splitting capability. Which may not be sufficient achieving efficient load balancing.
RFSLP [13]	No	MapReduce/Spark	No	Yes	No	No	Despite the use of scheduling concept, priority has not been assigned some critical tasks.
FASD [14]	Yes	MapReduce/Hadoop	No	Yes	No	No	The metric evaluation has limitation of not able to detect stragglers before occurrence.
SKRSP [15]	No	Hadoop/Spark	No	Yes	No	No	In the intermediate data distribution, the weight of each key is calculated, which could lead to overhead that may result in data processing delay
Hummer-1 [16]	Yes	MapReduce	No	No	No	Yes	Computation time required for deciding which task need to be cloned could also increase the execution time
Hummer-2 [16]	Yes	MapReduce	No	No	No	Yes	The approach could lead to increase in execution time
DSB [17]	Yes	MapReduce/Hadoop	No	Yes	Yes	No	Stragglers due to data skew have not been considered in this framework.
SkewTune [8]	Yes	MapReduce/Hadoop	No	Yes	No	Yes	SkewTune approach may lead to resource contention due to high computation.

IV. Scheduling in Straggler Mitigation

In this subsection, the solutions that employ scheduling concepts considering adaptive, resource allocation and data locality-aware scheduling in MapReduce framework have been analyzed and presented.

A. *Adaptive Scheduling*

A problem of omission failure due to stragglers has been addressed by proposing a Failure Detector Abstraction (FDA) based on MapReduce system [25]. The omission failure is due to timeout service adjustment, which strongly endangers the workload performance. Various algorithms have been suggested based on detector abstraction for describing the timeout. Therefore, three different levels of failure detector abstractions have been suggested namely, High Relax Failure Detector (HR-FD), Medium Relax Failure Detector (MR-FD) and Low Relax Failure Detector (LR-FD). The HR-FD serves as a non-dynamic alternative to the default timeout. The MRFD acts as non-static detector that modifies the timeout, based on progress score of each workload. While the LR-FD merges the MapReduce, non-static timeout using an exterior monitoring system to enforce accurate failure detection. The LR-FD is considering in case if there are strict deadline bounded user requests. Meanwhile, the authors claim that there is significant improvement in the timeout selection for user request regardless of the failure injection time and workload type. A Task scheduling optimization framework named ET-scheduler was proposed to handle time sensitive jobs and high resource consumption [26]. The existing scheduling technique cannot complete job within the time constraint of the user. Therefore, the ET-scheduler tries to allocate resources to the tasks of job submitted. The scheduler makes sure that jobs are completed within the time specified by user. It minimizes consumption and modifies the time allocation in the process of Map and Reduce.

A Map-Balance-Reduce (MBR) programming model was proposed for improving parallel programming model for load balancing over MapReduce [27]. The problem of load imbalance occurs if the data matching to a specific key or several keys account for majority of the data, then the Reduce node task will create unbalanced load. The MBR model runs on the custom Hadoop framework, which effectively processed the unique data with unbalance data. MBR programming model is designed based on two varied scheduling namely, processing and self-adaptation scheduling. The processing scheduling in MBR tries to find unbalanced task in advance, to compile balance function. Then value/keys are pairs outputted by Map, which are transmitted to balance the function. The values are outputted by Map and can be pre-processed for unbalanced data by calling the balance function process. In the self-adaptation scheduling, if there exists unbalanced load, the present Reduce task is terminated and then the unbalanced load is dynamically split and schedule for distribution to attain dynamic load balancing of the requested task.

Cheng *et al.* [28] proposed an enhanced MapReduce solution using Adaptive task tuning (*Ant*) over a heterogeneous environment. The solution tries to address poor performance due to heterogeneous clusters. In the existing work, there is homogeneous configuration of tasks on heterogeneous nodes, which leads to load imbalance and thus causes poor performance. The *Ant* can automatically determine the optimal configuration for distinctive tasks executed on different nodes. *Ant* algorithm performs better even when the jobs are large with more than one rounds of map task execution. At the beginning task are configured with randomly chosen settings. To evade trapping in local optima and speed-up task tuning, the algorithm employs genetic functions during task configuration.

B. *Resource Allocation Scheduling*

Huang *et al.* [29] proposed a Workload Alleviation Scheduling Framework (WASF) in order to avoid negative effect of intermediate data skew in small scale over MapReduce cloud. The intermediate data skew is caused due to unevenly allocation of intermediate data between nodes at run time. Thus, the intermediate data skew makes the nodes in the MapReduce cloud idle, which in turn leads to waste of computation resources. This also leads to prolonging of execution time, which gives user a bad experience in cloud computing. The WASF dynamically and smartly used the available computation resources for minimizing the intermediate data skew. A method that employs result analysis of profiling and relation of system parameters was proposed to address the limitation of speculative and clone execution method [30]. The limitation is in terms of performance reduction due to heterogeneous clusters and task stragglers in big data processing. The method tunes the quantity of task slots of nodes dynamically to match the processing capability of the nodes, which is based on present task progress rate and resource consumption. Therefore, a Task Progress Rate-based (TPR) approach has been developed. The tuning process is further optimized to achieve faster convergence. Thus, the method is implemented in the Hadoop MapReduce platform.

In [11], a Root-cause analysis for stragglers in Big data environment named BigRoots was proposed for handling user programs optimization problem. The BigRoots is a general method that incorporates system features and framework for root-cause analysis of stragglers in big data environment. It analyses the stragglers using features from Big data framework including system resource utilization, JVM garbage collection time and shuffle read/write bytes. The system resources include input/output, central processing unit and network, which can detect both external and internal causes of stragglers. The BigRoots is evaluated by injecting high resource utilization over dissimilar system components and different case studies were considered to analyze dissimilar workloads in Hibench to evaluate the performance.

Lakshmi [31] proposed an algorithm for enhancing Map and Shuffle Phases (MSP) over Hadoop MapReduce in Big data environment. In the MapReduce, the shuffle phase uses individual shuffle services component with efficient input/output policy. Meanwhile, the map phase’s output serves as an input to the subsequent phase. Thus, map phase requires intermediate checkpoints that regularly observe all splits created by intermediate phase. Therefore, the algorithm is designed as shuffle as a service component for decreasing the total execution time of task, monitoring map phase based on skew handling and improve resource consumption in a cluster.

C. Data Locality-aware Scheduling

A MapReduce concept based on data routing and locality was proposed to handle data imbalance in local and remote machines and to avoid network congestion [32]. A scheduling and routing algorithm named Joint Scheduler was proposed to balance task allocation to local and remote machines and to provide data routing that evade network congestion. The proposed algorithm is centered on bringing data close to computation instead of bringing computation close to data. Hence, it uses both communication network and computing resources efficiently. It is proven that the Joint Scheduler can support any load of jobs as used in the existing algorithm, which achieves the highest capacity region.

In [33], a task scheduling algorithm named rTuner was proposed to improve performance of the MapReduce job. The existing solutions are faced with the limitation of heterogeneity and resource contention, which lead to performance degradation in terms of overall job execution time. Thus, the rTuner consider the key objective to improve the reduce task execution time in both heterogeneous and homogeneous settings. Unlike the map task, the reduce task involves three phases namely, copy, shuffle and reduce phases. If the underlying situation is not analyzed by the scheduling algorithm, re-scheduling a straggler reduce task might negatively impact on the performance of the system.

Therefore, the rTuner study the reduce tasks' straggling causes and then tunes the reduce task. If tasks happen to be a straggler, then the rTuner re-schedules it to a suitable node, which depends on the situation. In summary, Table II presents the comparison of existing scheduling solutions in straggler mitigation.

TABLE II. COMPARISON OF SCHEDULING SOLUTIONS FOR STRAGGLER MITIGATION

Existing Solutions	Comparisons						Remark
	Heterogeneity among Network Node	Data Processing Method	Priority-based Scheduling	Mitigatio0n Approaches			
				Speculative	Blacklisting	Proactive	
FDA [19]	Yes	MapReduce	Yes	Yes	No	No	The failure detector abstraction did not consider data intensive computing systems.
ET-scheduler [20]	No	MapReduce/Hadoop	No	Yes	No	No	The scheduling optimization does not consider prioritization of task
MBR [21]	No	MapReduce/Hadoop	No	Yes	No	No	The pre-processing scheduling of the MBR model does not consider prioritization of the critical task
Ant [22]	Yes	MapReduce	No	Yes	No	No	The multi-tenant MapReduce settings has not been considered
WASF [23]	No	MapReduce/Hadoop	No	Yes	No	No	The scheduling framework does not consider prioritizing smaller or bigger workload
TPR [24]	Yes	MapReduce/Hadoop	Yes	Yes	No	No	This approach does not consider the shuffle phase.
BigRoots [25]	No	Spark/MapReduce	No	Yes	No	No	The relationship between locality and network utilization has not been investigated for the Root cause
MSP [26]	No	Hadoop/MadReduce	Yes	Yes	No	No	Meanwhile, heterogeneity of the network nodes has not been considered, which is important in the case of resource contention
Joint Scheduler [26]	No	MapReduce	Yes	No	No	Yes	Heterogeneity among network nodes has not been considered.
rTuner [27]	Yes	MapReduce/Hadoop	No	Yes	No	No	It is often fuzzy when deciding to declare a task as a straggler. However, the fuzziness has not been considered in rTuner.

Advantages and Disadvantages of Load Balancing Techniques (Algorithms).

Load techniques are either static or dynamic and each one has its own limitations and advantages which includes:

Advantages

- The static load balancing techniques are usually very efficient in stable environment because they do not need to monitor the resources during run-time.
- In a stable environment, operational properties do not change over time and loads are generally uniform and constant at the running time.
- Dynamic load balancing techniques are more flexible in dynamic computing environments.
- Dynamic load balancing techniques usually take into consideration different types of attributes in the environment both prior to and during run-time.
- Dynamic techniques can consider changes and provide better results in heterogeneous and dynamic environments.

Disadvantages

- Static load balancing techniques are not flexible and cannot accept changes of attributes during execution time.
- Static load balancing techniques do not consider continuous monitoring of the nodes hence they cannot consider load changes during run-time.
- When dynamic load balancing considers all changes during runtime it become more complex and dynamic to handle.
- Under certain conditions, dynamic load balancing techniques tend to have decreased performance in services.

V. Open Issues and Research Challenges

In this section, we have highlighted many research issues, which need research attention to attain efficient and effective straggler mitigation in MapReduce framework. The research issues are focused on how to balance the distribution of loads across the machines and how to efficiently schedule tasks to resource of the machines in order minimize slow tasks, which causes delay and negatively affect job completion time. The detailed discussions of the issues are as follows:

- Data Skew Caused by Inefficient Distribution of Data in Reducer Phase.

The main issues that affect the performance of the MapReduce framework is that some task take longer execution time to finish than others. This is due to data skew. The data skew is termed as inequality in the quantity of data allocated to each task or imbalance in the amount of work needed to process such data. These kinds of data are usually skewed in nature. Thus, it causes poor parallel processing, inequality of reducers input and high varied reducer execution time hence, it enlarges the completion time of the MapReduce job. Further, the intermediate data sharing in input data is not known, thus generating a strategy for the data group adjustment is difficult. It leads to the imbalance in the data distribution for a given task, which in turn causes stragglers. In addition, data skew could also occur when the sizes of keys are different and affect the shuffle time, which may cause straggler. Therefore, there is a need to develop a strategy that determines the values and keys for achieving balanced distribution of data, which mitigates the skewness of the data and improve the job completion time.

- Data Replication and Placement Issue.

The MapReduce framework is well known for its ability to handle large task and perform parallel processing during task handling. These strengths have encouraged researcher to employ replication strategy to minimize latency in job completion time. However, the replication concept has caused redundancy in task execution, which causes high resource consumption. Since the replication strategy generate redundant data there is a need for concepts that consider queuing and priority of the replicated data in terms of critical tasks for efficient job completion time. Because when there is large number of tasks that need to be executed, then the replication of these tasks will have impact on the computing resources hence, also affecting the task execution time due to resource constrain, which could cause stragglers.

- Poor Resource Allocation for Computation-Intensive Tasks.

In the existing MapReduce framework, some solutions use computational resources in a loose way on the basis that numerous idle nodes available can be used to collaboratively handle intermediate data skew. However, MapReduce system could be on a small scale and/or the task could be in a large scale. The proper utilization of the resource is very important in the case of a sophisticated system. In the existing solution, they smartly utilize computation resources in nodes and dynamically distribute workload of a node with other nodes by dispatching skewed intermediate data to a resource allocator. In addition, heterogeneity of the network nodes when allocating computation-intensive tasks to machines has not been considered, which is very significant in the case of resource contention. Consequently, the improper resource allocation to task could also lead to creation of straggler, which affect the job completion time of the Map Reduce framework. Therefore, considering the challenges, there is a need to design and develop an improved straggler mitigation solution that considers efficient resource allocation for task distribution.

- Inefficient Task-Resource Matching.

This usually results in sending simple tasks to machines with high computational capabilities and complex tasks to slow machines which may end up increasing the total job completion time.

VI. Conclusions

We have extensively reviewed existing related works and present the most recent research development in straggler mitigation approaches. The straggler issue has become challenging in MapReduce framework. Considering the negative effects of straggler, several solutions have been proposed that focus on load balancing and scheduling of the distributed task. Thus, a comprehensive review of the existing studies has been suggested in this paper. This review classified load balancing solutions into data skew and replication/placement approaches. While the scheduling approaches are classified into adaptive, resource allocation and data locality-aware scheduling. Further, open issues and research challenges are highlighted. The straggler problem degrades the performance of the existing data processing frameworks, specifically MapReduce. Therefore, there is a need to further explore more robust solution on how to effectively mitigate stragglers.

References

[1] G. Ananthanarayanan et al., "Scarlett: Coping with Skewed Content Popularity in MapReduce Clusters.".

[2] I. A. T. Hashem et al., "MapReduce scheduling algorithms: a review," Journal of Supercomputing, vol. 76, no. 7, pp. 4915–4945, Jul. 2020, doi: 10.1007/s11227-018-2719-5.

[3] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective Straggler Mitigation: Attack of the Clones.," Nsdi, pp. 185–198, 2013, doi: 10.1.1.366.6261.

[4] M. Reissig, "New Trends in the Theory of Nonlinear Weakly Hyperbolic Equations of Second Order," 1997.

[5] 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. Springer Netherlands, pp. 295–321, Sep. 01, 2017. doi: 10.1007/s10723-017-9408-0.

[6] G. Ananthanarayanan et al., "Reining in the outliers in map-reduce clusters using mantri," Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010, pp. 265–278, 2019.

[7] M. Zaharia, A. Konwinski, A. D. Joseph, and R. Katz, "Improving MapReduce Performance in Heterogeneous Environments."

[8] J. Rey, M. Cogorno, S. Nesmachnow, and L. A. Steffenel, "Efficient prototyping of fault tolerant map-reduce applications with Docker-Hadoop," in Proceedings - 2015 IEEE International Conference on Cloud Engineering, IC2E 2015, 2015, pp. 369–376. doi: 10.1109/IC2E.2015.73.

[9] U. Kumar and J. Kumar, "A Comprehensive Review of Straggler Handling Algorithms for MapReduce Framework," International Journal of Grid and Distributed Computing, vol. 7, no. 4, pp. 139–148, Aug. 2014, doi: 10.14257/ijgdc.2014.7.4.13.

[10] Y. Chen, S. Alspaugh, and R. H. Katz, "Design Insights for MapReduce from Diverse Production Workloads," 2012. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-17.html

[11] 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, doi: 10.1109/ACCESS.2018.2859826.

[12] M. Fatih, A. Aktas¸, P. Peng, and E. Soljanin, "Effective Straggler Mitigation: Which Clones Should Attack and When?"

[13] A. Kamal Abasi, A. Tajudin Khader, M. Azmi Al-Betar, S. Naim, S. Naser Makhadmeh, and Z. Abdi Alkareem Alyasseri, "A Text Feature Selection Technique based on Binary Multi-Verse Optimizer for Text Clustering; A Text Feature Selection Technique based on Binary Multi-Verse Optimizer for Text Clustering," 2019. [Online]. Available: http://www.unine.ch/Info/clef/,

[14] A. H. Katrawi, R. Abdullah, M. Anbar, and A. K. Abasi, "Earlier stage for straggler detection and handling using combined CPU test and LATE methodology," International Journal of Electrical and Computer Engineering, vol. 10, no. 5, pp. 4910–4917, Oct. 2020, doi: 10.11591/ijece.v10i5.pp4910-4917.

[15] Y. C. Kwon, M. Balazinska, B. Howe, and J. Rolia, "SkewTune in action: Mitigating skew in MapReduce applications," Proceedings of the VLDB Endowment, vol. 5, no. 12, pp. 1934–1937, 2012, doi: 10.14778/2367502.2367541.

[16] E. Gavagsaz, A. Rezaee, and H. Haj Seyyed Javadi, "Load balancing in join algorithms for skewed data in MapReduce systems," Journal of Supercomputing, vol. 75, no. 1, pp. 228–254, 2019, doi: 10.1007/s11227-018-2578-0.

[17] E. Gavagsaz, A. Rezaee, and H. Haj Seyyed Javadi, "Load balancing in reducers for skewed data in MapReduce systems by using scalable simple random sampling," Journal of Supercomputing, vol. 74, no. 7, pp. 3415–3440, 2018, doi: 10.1007/s11227-018-2391-9.

[18] M. A. Irandoost, A. M. Rahmani, and S. Setayeshi, "Learning automata-based algorithms for MapReduce data skewness handling," Journal of Supercomputing, vol. 75, no. 10, pp. 6488–6516, 2019, doi: 10.1007/s11227-019-02855-0.

[19] F. H. Syue, V. A. Kshirsagar, and S. C. Lo, "Improving mapreduce load balancing in hadoop," ICNC-FSKD 2018 - 14th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery, pp. 1339–1345, 2018, doi: 10.1109/FSKD.2018.8687158.

[20] G. Joshi and G. W. Wornell, "Efficient Straggler Replication in Large-Scale," vol. 4, no. 2. 2019.

[21] T. D. Phan, G. Pallez, S. Ibrahim, and P. Raghavan, "A new framework for evaluating straggler detection mechanisms in mapreduce," ACM Transactions on Modeling and Performance Evaluation of Computing Systems, vol. 4, no. 3, 2019, doi: 10.1145/3328740.

[22] Z. Tang, W. Lv, K. Li, and K. Li, "An Intermediate Data Partition Algorithm for Skew Mitigation in Spark Computing Environment," IEEE Transactions on Cloud Computing, vol. PP, no. c, p. 1, 2018, doi: 10.1109/TCC.2018.2878838.

[23] J. Li, C. Wang, D. Li, and Z. Huang, "Partial clones for stragglers in MapReduce," Communications in Computer and Information Science, vol. 503, pp. 109–116, 2015, doi: 10.1007/978-3-662-46248-5_14.

[24] X. Ouyang, C. Wang, and J. Xu, "Mitigating stragglers to avoid QoS violation for time-critical applications through dynamic server blacklisting," Future Generation Computer Systems, vol. 101, pp. 831–842, 2019, doi: 10.1016/j.future.2019.07.017.

[25] B. Memishi, M. S. Pérez, and G. Antoniu, "Failure detector abstractions for MapReduce-based systems," Inf Sci (N Y), vol. 379, pp. 112–127, 2017, doi: 10.1016/j.ins.2016.08.013.

[26] Y. Ren, H. Li, and L. Wang, "Research on MapReduce Task Scheduling Optimization," IOP Conference Series: Materials Science and Engineering, vol. 466, no. 1. 2018. doi: 10.1088/1757-899X/466/1/012016.

[27] J. Li, Y. Liu, J. Pan, P. Zhang, W. Chen, and L. Wang, Map-Balance-Reduce: An improved parallel programming model for load balancing of MapReduce, vol. 105. Elsevier B.V., 2020, pp. 993–1001. doi: 10.1016/j.future.2017.03.013.

[28] D. Cheng, J. Rao, Y. Guo, and X. Zhou, "Improving MapReduce performance in heterogeneous environments with adaptive task tuning," Proceedings of the 15th International Middleware Conference, Middleware 2014. pp. 97–108, 2014. doi: 10.1145/2663165.2666089.

[29] T. C. Huang, K. C. Chu, J. H. Lin, G. H. Huang, and C. K. Shieh, "Workload Alleviation Scheduling Framework to Alleviate Negative Performance Impact of Intermediate Data Skew in Small-Scale MapReduce Cloud," 2018 International Conference on System Science and Engineering, ICSSE 2018, pp. 1–6, 2018, doi: 10.1109/ICSSE.2018.8520003.

[30] X. Zhao, K. Kang, Y. Sun, Y. Song, M. Xu, and T. Pan, "Insight and reduction of MapReduce stragglers in heterogeneous environment," Proceedings - IEEE International Conference on Cluster Computing, ICCC, pp. 1–8, 2013, doi: 10.1109/CLUSTER.2013.6702673.

[31] J. V. N. Lakshmi, "Data analysis on big data: Improving the map and shuffle phases in Hadoop Map Reduce," International Journal of Data

Analysis Techniques and Strategies, vol. 10, no. 3. pp. 305–316, 2018. doi: 10.1504/IJDATS.2018.094130.

[32] W. Wang and L. Ying, “Data locality in MapReduce: A network perspective,” 2014 52nd Annual Allerton Conference on Communication, Control, and Computing, Allerton 2014, pp. 1110–1117, 2014, doi: 10.1109/ALLERTON.2014.7028579.

[33] R. Patgiri and R. Das, “rTuner: A performance enhancement of MapReduce job,” ACM International Conference Proceeding Series. pp. 176–186, 2018. doi: 10.1145/3177457.3191710.