ANN-inspired Straggler MapReduce Detection in Big Data Processing

Ajay Bansal · Manmohan Sharma · Ashu Gupta

Abstract One of the most challenging aspects of using MapReduce to parallelize and distribute large-scale data processing is detecting straggler tasks. Identifying ongoing tasks on weak nodes is how it’s described. The total computation time is the sum of the execution times of the two stages in the Map process (copy, combine) and the three stages in the Reduce phase (shuffle, sort, and reduce). The main aim of this paper is to estimate the accurate execution time in each location. The proposed approach uses a backpropagation neural network on Hadoop to detect straggler tasks and calculate the remaining task execution time, which is crucial in straggler task identification. The comparative analysis is done with some efficient models in this domain, such as LATE, ESAMR, and the real remaining time for WordCount and Sort benchmarks. It was found that the proposed model is capable of detecting straggler tasks in accurately estimating execution time. It also helps in reducing the execution time that it takes to complete a task.

Keywords Fog Analytics · Machine Learning · Environment Monitoring · Health Monitoring · Cyber Physical Framework

1 Introduction

Human life is becoming increasingly reliant on information technology [1–5]. Over the last two decades, increasing data output and the rapid growth of information technology have resulted in the development of a significant amount of data in multiple formats through various sources such as RFID tags [6], weblogs, data from scientific researches such as healthcare [7–9], Network management [10–15]. Numerous studies [14,16–18] have been performed to model data and partition query loads across multiple hosts. The authors [19] focus on collecting data in wireless sensor networks in a cloud structure in a short amount of time. In [21], the authors propose a new form of deep neural network that generates interpretable expression in the hidden layer. Hadoop is an open-source software platform that was first published in 2003 and functions similarly to an operating system to process and handle vast amounts of data across several machines. The Hadoop center is made up of a Hadoop distributed file system, and MapReduce processing [22]. This article is based on Yahoo’s Yet Another Resource Negotiator (YARN) system [23], which was introduced in 2010. It summarises various research areas in Hadoop, as shown in Fig. 1 and adapted from [24]. The emphasis of this study is task scheduling. We focus on dynamic adaptive schedulers based on speculative execution between all the approaches for task scheduling. ESAMR [20], and SECDT [25] are two current methods in the literature that are relevant to this group. The grey boxes in Fig. 1 reflect the proposed solution. As part of the Big Data processing infrastructure, the MapReduce method “chops” jobs into multiple functions. Using a traditional database management system to store and analyze data in an enterprise with a large data volume is difficult. Millions of users worldwide can find their information across the internet in a couple of seconds due to MapReduce’s architecture. The parallelization of these two phases gives this method its power (Map and Reduce). In reality, MapReduce is a framework for creating a distributed program that works with large amounts of data. When developing applications in a distributed environment, programmers must consider several factors [25]. For instance, how big a file is and how to break a large file into smaller pieces. Each mapping task’s input data is a small portion of the total input data. Reading data, applying the mapping function, sorting, and merging the inputs are all handled by mapping. The

Ajay Bansal
Lovely Professional University, Jalandhar, 144411, Punjab, India
E-mail: ajayg13@rediffmail.com

Manmohan Sharma
Lovely Professional University, Jalandhar, 144411, Punjab, India
E-mail: manmohan.sharma71@gmail.com

Ashu Gupta
Prince Sattam Bin Abdulaziz University, Saudi Arabia
E-mail: guptashu1@rediffmail.com

ISSN: 0010-8189
© CONVERTER 2020
www.converter-magazine.info
reduction step in the previous version of Hadoop did not begin until the mapping phase was completed. Still, in the next iteration (also known as YARN), the mapping phase is no longer required until the reduced feature can begin. Tasks are first sent to the mapping machine, which performs two stages: copy and combines, in the MapReduce structure. The total execution time is the sum of all the stages’ execution times. Since data is transferred during the copy and shuffle phases, they have the greatest effect during execution. Every stage has a weight assigned to it, calculated as the ratio of the stage’s execution time to the overall execution time. Errors can also be calculated by comparing approximate weights to a real one. The time it takes to complete the whole job is determined by the speed at which the slowest task is completed. A straggler task is a running task on the slowest node that should be identified and allocated to another node using speculative execution. In the LATE, the weights for and stage in stages are the same, while this is not the case in the experiments. Given the heterogeneous nature of Big Data processing, assigning the same weight to each stage is inefficient. The K-means algorithm is used by ESAMR to measure the weights of each task. In the proposed study, a dynamic approach is developed for estimating execution time that can be used in both heterogeneous and homogeneous settings. The weights are estimated using neural networks based on task execution time. This research aims to improve the efficiency of the Big Data computing infrastructure by reducing the error of calculating the remaining time from task execution and recognizing straggler tasks.

Article Structure

2 Related Work

The proposed approach is focused on addressing the flaws in existing literature-based solutions. For example, in the LATE process, the remaining time for each step of the running task is treated the same, while in the Reduce phase, the shuffle stage takes longer to complete than the other stages. Since they are dependent on the previous mission, the SAMR, ESAMR, and SECDT methods cannot estimate the running time accurately. It is not precise enough due to the differences in characteristics between previous and current tasks. ESAMR only considers executable information and ignores node requirements, which is necessary because node processing times vary depending on their features (CPU and memory). Pruning of branches in the decision tree-based method, or SECDT, results in knowledge loss and, as a result, false estimation. With the support of previously-stored running information on previous tasks, the proposed neural network approach seeks related functions to the current one and determines suitable weights.

In [37], the authors present a real-time task-scheduling algorithm that considers energy consumption. It works by grouping people according to the most recent cutoff period and then using a dynamic optimization approach to make scheduling decisions for each group. For efficient data transmission, the authors propose a comprehensive transmission (CT) model. They also devise a two-phase resource sharing (TPRS) protocol, which usually consists of a pre-filtering stage and a verification phase, to accomplish permitted resource sharing in the CT model easily and personally.
Table 1 The Equation used in the literature

| Author            | Equation                                                                 | Author            | Equation                                                                 |
|-------------------|--------------------------------------------------------------------------|-------------------|--------------------------------------------------------------------------|
| Satapathy et al.  | $P_i = \frac{x}{N}(1)$                                                   | Chen et al.       | $P_r = ps/t (5)$                                                         |
| Satapathy et al.  | $P_i = \frac{1}{(K + \frac{x}{N})}(2)$                                   | Zaharia et al.    | $pr = ps/t (5)$                                                          |
| Satapathy et al.  | $Avg(p_i) = \sum_{i=0}^{N} P_i[i]/N (3)$                                 | Chen et al.       | $STT = 0.4$                                                              |
| Javadpour et al.  | $P_i \leq Avg(p_i) - 20\% (4)$                                          | Zaharia et al.    | $ATTE = 10% \times TTE(i)/N (11)$                                       |
| Chen et al.       | $\sum_{i=0}^{N} P_i[i]/N (3)$                                            | Zaharia et al.    | $TTE = \frac{10% \times ATTE(i)}{N}$                                    |
| Zaharia et al.    | $TTE = (1 - P_r)Pr (6)$                                                  | Chen et al.       | $TTE = ATTE > ATTE \times STT (12)$                                      |
|                   |                                                                           | Zaharia et al.    |                                                                           |

2.1 Hadoop Naïve method

To control the nodes in the Hadoop, a method called Hadoop Naive Algorithm is utilised by [38, 39]. In the proposed technique, the weights in Map and Reduce phases are denoted by (m1=0, m2=1) and (r1=r2=r3=1/3), respectively. After the task has been completed for at least 1 minute, the following calculations are performed. Table 1 represented all the mathematical equations used in this literature. The progress score of (Ps) is calculated in Equation 3 assigned to the threshold, where N is denoted as the total number of executing tasks. Finally, Eq. 4 detects the straggler role by ensuring that the value of $P_s$ is less than 20% of the average ($P_s$) [17].

2.2 LATE (longest approximate time to end) approach

LATE aims to figure out how much time is left to complete tasks [17, 23]. While the weights are the same as the Hadoop naive process, Equation 5 considers the remaining time when selecting straggler tasks. Pr denotes the progress rate, and the task consumed time is indicated by $t$. Eq. 6 is used to determine how much time is left to complete tasks in a given stage (TTE). The tasks are sorted by the amount of time they have left. Straggler tasks are chosen from 10% of total tasks (known as a speculative cap). They will be reassigned to a node that meets the requirements of Equation 7. LATE has no way of knowing how much time is left. It uses constant numbers for each step’s weight, even though the impact of each step for each task varies, and the ratios are not always constant. The proposed method finds related tasks and determines acceptable weights for each move using stored executable information.

2.3 Self-adaptive MapReduce Scheduling Algorithm method (SAMR)

This method saves the weight of completed tasks at each point in an XML file and uses it to predict the next task execution time [26]. The consequences for the first stage are (1,0,1/3,1/3,1/3). Every 100 milliseconds, $P_s$ is measured. Equation 8 calculates the average execution speed, while Equation 9 identifies the straggler job. Straggler tasks are indicated by the closeness of Stac to zero. Equation 10 is often used to measure the number of backup tasks (BackupNum), where TaskNum is the number of executing tasks. $B_p$ has been calculated to be 0.2. Equation 11 calculates the average time to completion of all running tasks (ATTE). It is a slow task if it fulfills Equation 12. The Slow Task Threshold (STT) is a scale that measures how fast or slows a task is in the range [0,1]. We use STT = 0.4 in this article since STT less than 0.4 considers many fast tasks to be slow. Many slow tasks are expected to be quick when STT is greater than 0.4.

2.4 Enhanced Self-adaptive MapReduce Scheduling Approach (ESAMR)

In 2012, ESAMR [42] was implemented to enhance the SAMR algorithm. The K-means algorithm divides this knowledge into k (k=10) groups and stores weights from previous tasks. A temporary weight will be determined for that process after at least 20% of the total tasks in the Map and Reduce stages have been completed. The algorithm looks for a cluster with the lowest weight to assign the partially completed tasks to them. If a node does not have any completed tasks, the average weight of all collections will be utilized.

2.5 Speculative Execution Algorithm Based On Decision Tree (SECDT)

3 Proposed Model

There are two parts to the speculative execution developments with the ANN system. To begin with, the proposed approach can be used in both heterogeneous and homogeneous settings. Second, it accurately calculates the weights of each level,
resulting in a shorter total execution period. The proposed method employs two types of variables: dependent (weights allocated to each stage and remaining MapReduce task execution time) and independent (execution time, amount of processed data, and progress rate) variables. The proposed approach has three key components, as shown in Fig. 2: an information storage database for saving data from previously performed tasks, weight estimation by the neural network, and task execution time estimation. Each new task will be separated into 2 phases: Map and reduce. As input values for NN to estimate weights, the produced reports are divided into train and test datasets. The test data results can also be used to train the model and increase the accuracy of the results. It is possible to estimate the remaining time of task execution by estimating NN weights, which is the secret to locating straggler tasks.

A flowchart in Figure 3 depicts the relationship between the components shown in Figure 2. The resource manager can devote staff to this job if you join a job through Hadoop. After t seconds, the machine manager looks for straggler tasks. In this stage, progress score, success rate, and residual executing time for each task will be determined using executing information from working nodes in order to identify straggler tasks. The remaining execution time is used to sort the tasks. If the number of speculative tasks exceeds 10% of total tasks, the task with the most time left will be completed. The user can add a job to Hadoop using the suggested form, which causes the system manager to devote resources to the new job. The system manager will search for straggler tasks after each task has run for t seconds. The next move is to rank all of the remaining tasks according to how much time they have left to complete them. Slave nodes save information about existing jobs that are being run speculatively. The system manager collects all information about currently running tasks and applies the NN algorithm. If the number of speculative tasks is greater than 10% of the total number of tasks, the success rate, remaining time execution, and task progression are determined. In that case, no modifications to the execution procedure will be made. Otherwise, the task with the least amount of time left would be started. This procedure will be followed for all new hires. In this study, we estimated weights using a backpropagation neural network. After each task is completed, the weight of the stage is calculated by dividing the execution time of each stage (in Map and Reduce) by the total execution time of each step. The neural network uses this stored executable information to estimate runtime. The implementation of the neural network algorithm is focused on comparing the real value to the expected value. A multilayer feedforward neural network is used to learn in a backpropagation neural network.

3.1 Scheduler

The proposed approach is intended to enhance the process of speculative execution. Each task in a newly elected job will be broken down into MapReduce tasks and assigned to a node. The mission implementation report is sent to the source manager regularly and stored in the repository. To execute assigned tasks, we use a container on each node. When the tasks are running, the source manager keeps an eye on them to see any stragglers.
Table 2 Parameters used in proposed solution

| Features                                      | Values                  |
|-----------------------------------------------|-------------------------|
| Hadoop cluster installation mode              | Fully distributed       |
| Number of the cluster node                   | 5                       |
| RAM level node 1, 2                          | 4G                      |
| RAM level node 3, 4                          | 3G                      |
| Network topology                             | Slave master            |
| The size of virtual machines hard drive      | 50 G                    |
| Master node                                   | Contains a job follower |
| Slave node                                    | Contains a task follower and a data node |
| Distributed file system block size            | 128 MB                  |

3.2 Nural Network Weight calculation

The real weight value is compared to the approximate one in each iteration of the proposed process. The neural network calculates the weights of all three stages of the reduction process. In this study, backpropagation NN is used, and the approximate weight is compared to the actual one in each iteration. The learning will either proceed to the next iteration or stop with the current estimations, depending on the accuracy. The number of epochs has also been established as a termination factor. \( P_s \) and SubPS will be computed. \( P_s \) will be determined using Eq. 1; \( k \) represents the current stage number. The ratio of processed key/value pairs (\( N_f \)) to all key/value pairs is called Subps (\( N_a \)).

\[
p_s = \text{SubPs} \times \sum_{k=1}^{r_k} \frac{n_f}{n_a} \quad (1)
\]

\[
\text{SubPs} = \frac{n_f}{n_a} \quad (2)
\]

3.3 Reduce phase

The estimated weights of each stage are used to estimate the total and remaining execution time. Subps and execution time are calculated using the amount of information to be collected and the current task progress stage. In the speculative execution, the task with the highest execution time is allocated to another node for implementation. All straggler tasks do not re-execute and are organized depending on the remaining execution time. The Mapping process is considered one stage in the proposed method; hence, the neural network with the inputs of Subps and \( N_f \) and the remaining execution time is calculated and stored in the TTE.

4 Performance Evaluation

On each virtual machine, the proposed approach is implemented using Hadoop 2.7.3 and Ubuntu 16.0. (VM). The simulation computer has 16 GB of RAM and a 3.2 GHz Intel i7 processor. HDFS (High-Definition File System) has a web console for monitoring and reviewing file systems that allow users to access the HDFS file system's content. All of the experiments are focused on three blocks of the 128 MB HDFS distributed file system's block size. The interaction of nodes is depicted in Figure 13 in the Appendix. For each of the Map and Reduce tasks, we considered two slaves. The Hadoop version 2.7.3 speculate section has been updated. Based on task execution time, job execution time, and stage weights, the results are produced using the t-word counting software [17, 18, 24] and compared to LATE, No-Speculate, and ESAMR. WordCount reads text files and counts how many times each word is repeated. Mapping is in charge of dividing lines into words and generating key/value pairs for each letter. A collection of words is used as the input dataset. We have compared the results of our proposed method to the Sort benchmark, which is commonly used in the literature to assess the consistency of proposed methods. Hadoop 2’s standard block size is 128 MB. When a larger file is inserted into HDFS, it is split into 128 MB chunks and distributed through data nodes. The detail of each parameter is described in Table 2.

A node manager, a job follower, a master, and four slave nodes are part of the cluster he built. A data node and a mission follower are located on each slave. We use four separate experiments to test errors in weight estimation, mission execution time, and work execution time. A comparison of the proposed method and SVR is included in the first experiment. The second experiment uses data from the execution process to measure weights for the Map and Reduce phases. The remaining execution time is calculated in the third experiment using the weights that have been obtained. The final experiment will assess the impact of speculative execution on execution time as a function of data volume and the number of nodes. The constant and variable parameters of experiments in a neural network are shown in Table 3.
Table 3 Experimental Parameter and constant

| Number of experiments | Constant Parameters | Changeable Parameters | Explanation |
|-----------------------|---------------------|-----------------------|-------------|
| 1                     | Weight of map-reduce phase of executed tasks | Comparing the estimated weight of backup vector machine algorithms and artificial neural network | Learning rate=0.05, Epoch =100 |
| 2                     | Weight of map-reduce phase of executed tasks, Estimated weight with the help of an artificial neural network algorithm in reduce phase, Progress rate of executed tasks | Estimated weight with the help of an artificial neural network algorithm in Map phase | Learning rate=0.05, Epoch =100 |
| 3                     | Weight of map-reduce phase obtained from the previous experiment | Remaining time of tasks’ execution | 0.251G–13G |
| 4                     | Weight of map-reduce phase of executed tasks | the execution time of the job, Number of nodes, Amount of input source manager after 60 s of execution searches about the straggler task | Measuring execution time with changing the number of nodes and amount of input, 1–5, 0.251G–13G |

Table 4 Weight calculation using different algorithms

|                  | LATE Algorithm | SVR | Neural Network |
|------------------|----------------|-----|---------------|
|                  | Real amount   | Estimated value | Real amount   | Estimated value | Real amount   | Estimated value |
| 0.78             | 0.77           | 0.46          | 0.91          | 0.76           | 0.811         |
| 0.214            | 0.181          | 0.66          | 0.893         | 0.449          | 0.462         |
| 0.28             | 0.211          | 0.34          | 0.592         | 0.616          | 0.611         |
| 0.147            | 0.145          | 0.25          | 0.231         | 0.235          | 0.213         |
| 0.93             | 0.94           | 0.16          | 0              | 0.561          | 0.512         |

4.1 Experiment 1

Regression methods such as support vector machines and artificial neural networks effectively predict the continuous nature of weights. The values are defined based on error minimization in the proposed method. In contrast, SVR converts the risk of incorrect classification into an objective function and adjusts and optimizes the parameters based on this target function. In this part, the proposed method’s results are compared to SVR and decision tree. The bellow equation is used to measure errors. The proposed method outperforms SVR by 99% and the decision tree method by 81% Table 4.

$$Error_{methods} = \frac{1}{N} \sum_{i=1}^{N} e_i^2$$

(3)

4.2 Experiment 2

There are two parts to the mapping process and three parts to the reduce phase. In this part, we use the presented methods, LATE and ESAMR, to compare the approximate weights in these two phases. For this portion, the stored executive data is captured in a file and given to the neural network as training data, including the volume of processed data and the extent to which activities are completed. The proposed technique uses backpropagation to find weights, while the ESAMR algorithm uses a value of k of 10. Compared to ESAMR, the results show an 85% increase and a 99% improvement compared to the LATE. Table 5 compares the proposed method’s weight calculation to the ESAMR and LATE algorithms. Each cell in this table contains two numbers. The real weight is represented by one, and the estimated weight by the algorithm is represented by the other. These two numbers must be as similar as possible to improve the method’s accuracy by correctly identifying straggler tasks.
Table 5  Weight estimation using different methods versus real weights of a WordCount for second experiment

|                | Proposed method | ESAMR   | LATE   |
|----------------|-----------------|---------|--------|
| S1             | 0.78571−0.70432 | 0.8538−0.7425 | 0.33−0.7432 |
| S2             | 0.7−0.65        | 0.46−0.587 | 0.33−0.08062 |
|                | 0.9019−0.8991   | 0.8604−0.7401 | 0.33−0.065 |
|                | 0.33−0.021      | 0.9941−0.8954 | 0.33−0.002 |
|                | 0.33−0.0002     | 0.9754−0.7451 | 0.33−0.04072 |
|                | 0.33−0.0525     | 0.9921−0.8971 | 0.33−0.05028 |

Fig. 3  Comparision of error handling with sate-of-the-art

Fig. 4  Run time with two slaves

4.3 Experiment 3

Speculative execution aims to cut down on execution time. Consequently, the remaining time estimate in the proposed method is taken into account in this experiment. The first step was to save the results of running the word count program with various input values. Then 20 tasks (map and reduce) are chosen to assess the execution time estimate. The application of ESAMR [30] is tested on six machines, one of which is the manager and the others are working nodes. Experiments for this study are carried out on a Hadoop cluster of five computers (one manager and four working nodes). Estimated time and the execution time for Map and reduce are shown in Figure 3 and Figure 4.

As is well known, the proposed method is the closest case and has the most accurate estimate of the actual runtime and the shortest runtime, among other techniques. According to Fig. 5, the proposed approach has a 55% improvement
in error rate compared to ESAMR and a 77% gain compared to LATE. The primary benefit of our approach over current methods is the accurate calculation of task remaining time. For the Map phase of LATE, ESAMR, and our process, the average distance between the actual and projected weights is 68.3 33.85 18.1 and for the Reduce phase, it is 176.75 89.95 27.3, respectively. ESAMR is more accurate than LATE, and we’ve included Fig. 14 in the Appendix to demonstrate how effective the proposed method is compared to ESAMR in Word Count. It measures the difference estimation error for each task in Map and Reduce calculated by ESAMR and our method. When compared to ESAMR, points above the line y=0 indicate that our approach performs better.

4.4 Experiment 4

The impact of data and the number of slave nodes on execution time are investigated in this experiment. The word count program was run with different inputs in the same conditions for all three algorithms for this reason. Fig 6 shows the results obtained with two slave nodes; Fig 7 shows the results obtained with three slave nodes, and Fig 8 shows the results obtained with four slave nodes. As shown, increasing the number of nodes does not always result in a faster execution time. Because of the longer data transmission time between nodes, as shown in Fig. 9, adding nodes in high-volume data is cost-effective, while adding nodes in low-volume data does not increase or alter the execution time. The proposed method’s remaining execution time is estimated to be 35% faster than real-time and 15 times faster than ESAMR. The execution time is reduced by identifying straggler tasks early and assigning them to another node.

4.5 Qualifying the proposed method by Utilizing Sort benchmark

On a Sort 10 GB job, Figures 10 and 11 show a comparison of the time to end estimation error of Map and Reduce tasks using ESAMR, LATE, and the proposed method. Even though the ESAMR has a small error on different tasks, the proposed method outperforms the ESAMR in the vast majority of them. The differences between estimated and actual time to completion of Map and Reduce tasks with ESAMR are 2 and 8 seconds, respectively, whereas our method’s error is 2 and 7 seconds. For the Map phase of LATE, ESAMR, and the proposed process, the average distance between the actual and projected weights is 17.9, 3.25, 2.95, and for the Reduce phase, it is 126.55, 9.4, 9.25, respectively. ESAMR produces similar
Fig. 7 Runtime with three slaves

Fig. 8 Runtime with four slaves

Fig. 9 Percent improvement compared with the neural network algorithm implementation method and LATE ESAMR
results to the model we’ve provided. In the Appendix to compare estimation error between the proposed approach and ESAMR for the Sort benchmark. That is, we determined the difference between ESAMR's estimation error and the proposed method’s estimation error for each task in Map and Reduce. Points above the y=0 line indicate that the proposed approach performs better than ESAMR.

5 Conclusion

The speed with which data is analyzed is critical in Big Data processing. This research aims to improve the efficiency of Big Data’s computing infrastructure to speed up data processing by identifying straggler tasks through speculative execution. Speculative execution using the ANN algorithm is proposed to achieve this aim. The weight estimate improved by 86 percent compared to ESAMR and by 99 percent compared to the LATE as a result of this growth. In addition, as compared to LATE and ESAMR, the proposed approach reduced execution time by 24% and 15%, respectively. The proposed method can be improved in the future, given the value of fast execution time. The neural network can be used to determine the appropriate node for running a straggler task, for example, to enhance the process of speculative execution. Other information, such as the number of failures in different steps, may also be used to calculate the input data for this purpose. The proposed approach begins the support task from the beginning, and the execution can be sped up by continuing the straggler task’s execution. Before assigning a mission, it is preferable to consider the number of previous failures of a node. In the future, a mixture of heuristic and artificial intelligence algorithms may be used to process data in parallel and, as a result, perform tasks more quickly. We estimate the remaining time of tasks, identify straggler tasks, and calculate weights in this article using NN. Other prediction approaches such as Taylor, learning automata, and reinforcement learning should be used in future work.
1. Nanduri, R., Maheshwari, N., Reddyraja, A., & Varma, V. (2011, November). Job aware scheduling algorithm for mapreduce framework. In 2011 IEEE Third International Conference on Cloud Computing Technology and Science (pp. 724-729). IEEE.

2. Song, G., Meng, Z., Huet, F., Magoules, F., Yu, L., & Lin, X. (2013, November). A hadoop mapreduce performance prediction method. In 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (pp. 820-825). IEEE.

3. Javadpour, A. (2019). An optimize-aware target tracking method combining MAC layer and active nodes in wireless sensor networks. Wireless Personal Communications, 108(2), 711-728.

4. Javadpour A (2019) Providing a way to create balance between reliability and delays in SDN networks by using the appropriate placement of controllers. Wirel Pers Commun

5. Javadpour A, Kazemi Abharian S, Wang G (2017) Feature selection and intrusion detection in cloud environment based on machine learning algorithms. In: 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications 2017 IEEE International Conference on Ubiquitous Computing and Communications, pp 1417–1421.

6. Kaur, N., & Sood, S. K. (2015). An energy-efficient architecture for the Internet of Things (IoT). IEEE Systems Journal, 11(2), 796-805.

7. Javadpour, A., & Mohammadi, A. (2016). Improving brain magnetic resonance image (MRI) segmentation via a novel algorithm based on genetic and artificial neural network. Journal of biomedical physics & engineering, 6(2), 95.

8. Javadpour, A., & Memarzadeh-Tehrani, H. (2015, March). A wearable medical sensor for provisional healthcare. In 2015 2nd international symposium on physics and technology of ISPTs (pp. 293-296). IEEE.

9. Javadpour, A., Memarzadeh-Tehrani, H., & Saghaﬁ, F. (2015, May). A temperature monitoring system incorporating an array of precision wireless thermometers. In 2015 international conference on smart sensors and application (ICSSA) (pp. 153-160). IEEE.

10. Rezaei S, Radmanesh H, Alavizadeh F, Nikoofar H, Lahouti F (2016) Automatic fault detection and diagnosis in cellular networks using operations support systems data. In: NOMS 2016—2016 IEEE/IFIP Network Operations and Management Symposium, pp 468–473.

11. Park, J., Adeli, H., Park, N., & Wonang, I. (2014). Mobile, Ubiquitous, and Intelligent Computing. Lecture Notes in Electrical Engineering.

12. Zhang, Z., Barbary, K., Nothaft, F. A., Sparks, E., Zahn, O., Franklin, M. J., & Perlmutter, S. (2015, October). Scientific computing meets big data technology: An astronomy use case. In 2015 IEEE International Conference on Big Data (Big Data) (pp. 918-927). IEEE.

13. Aggarwal, J. V., Bhatnagar, V., & Mishra, D. K. (2018). Big data analytics. Springer Singapore.

14. Zhang, Z., Barbary, K., Nothaft, F. A., Sparks, E., Zahn, O., Franklin, M. J., & Perlmutter, S. (2018, October). Power curtailment in cloud environment utilising load balancing machine allocation. In 2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UITC/ATCBerlin/IOP/SCI) (pp. 1364-1370). IEEE.

15. Javadpour, A., Wang, G., & Li, K. C. (2019, November). A high throughput MAC protocol for wireless body area networks in intensive care. In International Conference on Smart City and Informatization (pp. 23-34). Springer, Singapore.

16. Liu, Q., Wang, G., Liu, X., Peng, T., & Wu, J. (2017). Achieving reliable and secure services in cloud computing environments. Computers & Electrical Engineering, 59, 153-164.

17. Javadpour, A., Saedifar, K., Wang, G., & Li, K. C. (2020). Optimal execution strategy for large orders in big data: order type using Q-learning considerations. Wireless Personal Communications, 112(1), 123-148.

18. Javadpour, A. (2019). Improving resources management in network virtualization by utilizing a software-based network. Wireless Personal Communications, 106(2), 505-519.

19. Wang, T., Li, Y., Wang, G., Cao, J., Bhuiany, M. Z. A., & Jia, W. (2017). Sustainable and efficient data collection from WSNs to cloud. IEEE Transactions on Sustainable Computing, 4(2), 252-262.

20. Javadpour, A., Adelpour, N., Wang, G., & Peng, T. (2018, October). Combing fuzzy clustering and PSO algorithms to optimize energy consumption in WSN networks. In 2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UITC/ATCBerlin/IOP/SCI) (pp. 1371-1377). IEEE.

21. Dai, Y., Wang, G., & Li, K. C. (2018). Conceptual alignment deep neural networks. Journal of Intelligent & Fuzzy Systems, 34(3), 1631-1642.

22. Phan, T. D., Pallez, G., Ibrahim, S., & Raghavan, P. (2019). A new framework for evaluating straggler detection mechanisms in mapreduce. ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS), 4(3), 1-23.

23. Yao, Y., Gao, H., Wang, J., Sheng, B., & Mi, N. (2019). New scheduling algorithms for improving resource and utilization in hadoop YARN clusters. IEEE Transactions on Cloud Computing.

24. Polato, I., Ré, R., Goldman, A., & Kon, F. (2014). A comprehensive view of Hadoop research—A systematic literature review. Journal of Network and Computer Applications, 46, 1-25.

25. Ouyang, X., Garraghan, P., Primas, B., McKee, D., Townsend, P., & Xu, J. (2018). Adaptive speculation for efficient internetware application execution in clouds. ACM Transactions on Internet Technology (TOIT), 18(2), 1-22.

26. Wang, N., Yang, J., Lu, Z., Li, X., & Wu, J. (2016, November). Comparison and improvement of Hadoop MapReduce performance prediction models in the private cloud. In Asia-Pacific Services Computing Conference (pp. 77-91). Springer, Cham.

27. Pol, V. V., & Patil, S. M. (2016, August). Implementation of on-process aggregation for efficient big data processing in Hadoop MapReduce environment. In 2016 International Conference on Inventive Computation Technologies (ICICT) (Vol. 3, pp. 1-5). IEEE.

28. Sakr, S., Liu, A., & Fayoumi, A. G. (2013). The family of mapreduce and large-scale data processing systems. ACM Computing Surveys (CSUR), 46(1), 1-44.

29. Zhang, X., Yu, W., & Zhao, C. (2016). MrHeter: improving MapReduce performance in heterogeneous environments. Cluster Computing, 19(4), 1691-1701.

30. Sun, X., He, C., & Lu, Y. (2012, December). ESAMR: An enhanced self-adaptive MapReduce scheduling algorithm. In 2012 IEEE 18th International Conference on Parallel and Distributed Systems (pp. 148-153). IEEE.

31. Giachetta, R. (2015). A framework for processing large scale geospatial and remote sensing data in MapReduce environment. Computers & Graphics, 39, 47-56.

32. Chen, Q., Liu, C., & Xiao, Z. (2013). Improving MapReduce performance using smart speculative execution strategy. IEEE Transactions on Computers, 62(4), 945-967.

33. Sun M, Zhang U, Li C, Lu K, Zhou X (2016) Scheduling algorithm based on prefetching in mapreduce clusters. Appl Soft Comput 38(C):1109–1118

34. Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010, May). The hadoop distributed file system. In 2010 IEEE 26th Symposium on Mass Storage Systems and Technology: An astronomy use case. In 2015 IEEE International Conference on Big Data (Big Data) (pp. 918-927). IEEE.

35. Fan, L., Gao, B., Zhang, F., & Liu, Z. (2014). OS4M: achieving global load balance of mapreduce workload by scheduling at the operation level. arXiv preprint arXiv:1406.3901.

36. Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., & Wu, A. Y. (2002). An efficient k-means clustering algorithm: Analysis and implementation. IEEE transactions on pattern analysis and machine intelligence, 24(7), 881-892.

ISSN: 0010-8189 © CONVERTER 2020

www.converter-magazine.info

References
37. Peng, Z., & Wang, G. (2017). An optimal energy-saving real-time task-scheduling algorithm for mobile terminals. International Journal of Distributed Sensor Networks, 13(5), 1550147717707891.
38. Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R. H., & Stoica, I. (2008, December). Improving MapReduce performance in heterogeneous environments. In Osdi (Vol. 8, No. 4, p. 7).
39. Shen, H., & Li, C. (2018, June). Zeno: A straggler diagnosis system for distributed computing using machine learning. In International Conference on High Performance Computing (pp. 144-162). Springer, Cham.
40. Satapathy, S. C., Raju, K. S., Mandal, J. K., & Bhateja, V. (Eds.). (2015). Proceedings of the Second International Conference on Computer and Communication Technologies: IC3T 2015, Volume 1 (Vol. 379). Springer.
41. Chen, Q., Zhang, D., Guo, M., Deng, Q., & Guo, S. (2010, June). Samr: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment. In 2010 10th IEEE International Conference on Computer and Information Technology (pp. 2736-2743). IEEE.
42. Yang, G. (2011, October). The application of mapreduce in the cloud computing. In 2011 2nd International Symposium on Intelligence Information Processing and Trusted Computing (pp. 154-156). IEEE.
43. Li, Y., Yang, Q., Lai, S., & Li, B. (2015, January). A new speculative execution algorithm based on C4. 5 decision tree for Hadoop. In International Conference of Young Computer Scientists, Engineers and Educators (pp. 284-291). Springer, Berlin, Heidelberg.