

Input/output optimization scheduler for cloud-based map reduce framework

Farha Naaz, Sameena Banu

Department of Computer Science and Engineering, KBN University, Kalaburagi, India

Article Info

Article history:

Received Dec 26, 2023

Revised Apr 27, 2024

Accepted May 7, 2024

Keywords:

Cloud computing
Hadoop MapReduce
MapReduce
Memory
Quality of services
Scientific workflow

ABSTRACT

Hadoop MapReduce (HMR) provides the most common MapReduce (MR) framework, and it is available as open source. MR is a famous computational framework for evaluating unstructured, and semi-structured big data and executing applications in the past ten years. Memory and input/output (I/O) overhead are just two of the many problems affecting the current HMR scheduler system. This study aims to improve systems resource use including the processing of data in real-time by creating a memory I/O optimized scheduler (MIOOS) for HMR. The disk I/O seek can be reduced by using MIOOS, which analyzes the entire memory management. Additionally, the MIOOS makespan approach is used to reduce the occurrence of problems in intermediary tasks. Both the MIOOS approach and the current approach are assessed by using complex scientific workflow applications with extreme task inter-dependencies. Further, the comparison study demonstrates that the MIOOS framework outdoes the current approach regarding makespan and overall memory usage.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Farha Naaz

Department of Computer Science and Engineering, KBN University

Kalaburagi, India

Email: farhanaaz_12@rediffmail.com

1. INTRODUCTION

Many businesses, including educational institutions, government departments, and industries, amass substantial quantities of unorganized information from multiple places like the World Wide Web, computational biology, social media platforms, wireless sensor networks, and more, with distinct objectives in mind. In addition, the analysis of complex data-intensive and scientific workflow has emerged as a highly sought-after task for numerous organizations [1]. Nevertheless, it is worth noting that existing state-of-the-art approaches often struggle to effectively handle real-time scenarios involving direct acyclic graph (DAG) workflow applications with high interdependence among intermediate tasks [2]. In the context of real-time scenarios, it is noteworthy to mention that data-driven platforms such as Google have come up with a parallel computation method known as the MapReduce (MR) framework [3]. This framework, specifically aimed at enabling parallel processing in a distributed approach, holds significant relevance. Hadoop MapReduce (HMR) [3] is a widely accepted and extensively utilized method in the field of data processing and analysis. One of the key factors contributing to its widespread adoption is its open-source nature, allowing for greater accessibility and community-driven development. The HMR approach encompasses several different stages, namely Setup, Mapping, Reduce, and Shuffling and Sorting. These stages are visually represented in Figure 1.

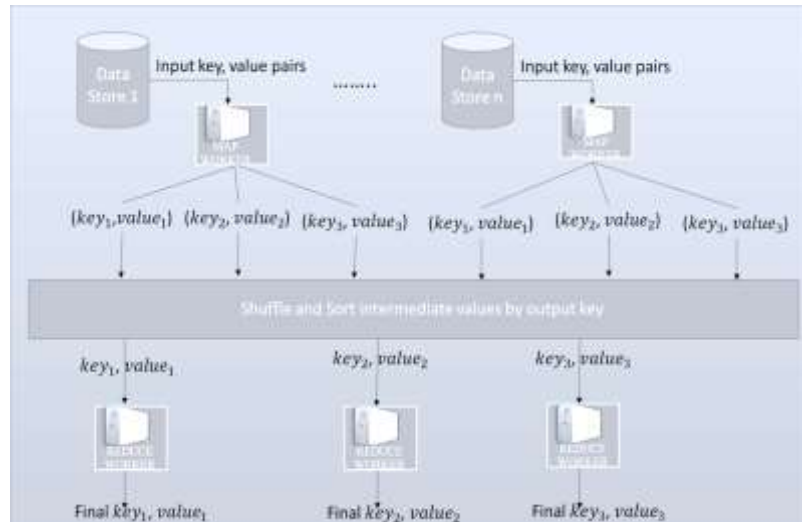


Figure 1. The architecture of standard Hadoop MapReduce framework

The Azure-HDInsight structure offers a high degree of adaptability to accomplish maximum efficiency by allowing the workflow to be deployed in large-scale clusters in the cloud platform. However, the current cloud-based computational model fails to workflow application task deadline requirements. Furthermore, it remains the responsibility of the consumer to determine the appropriate resource allocation to fulfill task deadlines, which is widely recognized as a difficult task. Moreover, the consideration of Hadoop processing time is of utmost importance in determining the necessary resources for meeting task deadlines. It has been noticed that Hadoop tasks consist of multiple processing stages, specifically the Reduce, Shuffling, and Mapping phases, which are intricately interconnected. In this context, it is important to note that every stage necessitates both input/output (I/O) operations [4] alongside memory operations [5]. Additionally, during the shuffle stage, the first sub-stage is executed concurrently alongside the Mapping stage, which can be regarded as a kind of overlapping stage. Conversely, the remaining sub-stages within the shuffle stage are executed following the conclusion of the mapping stage, which is deemed to be a non-overlapping stage [6]. Furthermore, to optimize the utilization of cloud resources, several makespan approaches have been established and discussed in [7]-[9]. While these approaches demonstrate potential, they suffer from inefficiency and high computational overhead [10]. This is primarily due to their failure to consider the scenarios of non-overlapping and overlapping shuffling stages. Thus, the proposed work is focused on addressing the different scheduling problems from memory resources, I/O, and computational resources to reduce the overall makespan of workflow execution in the cloud-based Map-Reduce framework [11].

The scientific workflow applications have grown exponentially in recent years due to the availability of parallel and distributed computing platforms, and their efficiency has improved as a result. There have been many ways for modeling Hadoop, among which some of the most efficient methods have been discussed in [12]. Methods created in [13] for a homogeneous framework execute unsatisfactorily in a heterogeneous framework due to the need for the I/O and memory optimization method, despite providing lockless first in first out (FIFO) which incorporates HMR and other applications. In the HMR system, the scheduler technique used during the shuffling stage is the primary determinant of the makespan for completing tasks [14], [15]. In [16], a combined scheduling strategy has been offered to shorten the time it takes to complete tasks by taking into account the overlap between the mapping and shuffling stages. A topology-aware hierarchy MapReduce (MR) scheduling was introduced for lowering I/O overhead [17]. Reduced I/O demands mean less time spent waiting for tasks to finish. For quicker stage reshuffling operation, [18] introduced a system for planning called shadow. Both spatial awareness in Map task completion and load-balancing during the shuffling stage benefit from the approaches' tradeoffs. As demonstrated in [19], prior approaches to performing heterogeneous tasks have not taken task dependence models into account, leading to inefficient use of available resources. So, they devised a new yet another resource negotiator (YARN) scheduling scheme that cuts down the makespan of various industrial tasks. Makespan efficiency, particularly when dealing with complicated repetitive applications, is negatively impacted by the fact that the models described in [20], and [21] don't account for failures at intermediary tasks. This increases the difficulty of coming up with a workable scheduler plan for the MR framework for effective scientific workflow execution [22], [23] in a heterogeneous cloud platform [24].

2. METHOD

The present study aims to propose a novel approach for scheduling tasks in the cloud-based MR framework leveraging the principle of HMR, with an emphasis on optimizing resources and memory management. Our proposed method, memory I/O optimized scheduler (MIOOS), introduces a thread-based approach to execution to enhance memory usage and reduce I/O overhead. Furthermore, this research endeavor emphasizes the development of a memory dynamic distribution mechanism for tasks across threads within a single virtual machine in the cloud platform. In addition, this study aims to enhance the utilization of memory for the central processing unit (CPU) and I/O by developing an I/O approach. Additionally, the memory optimization technique is employed to prevent redundant data re-reading before transfer, thereby reducing the workload by caching the final output of an operation in memory. In conclusion, we provide a comprehensive makespan approach that incorporates data dependence for the execution of complex scientific workflows. The proposed MIOOS approach can reduce overall makespan with minimal memory usage in comparison with current methodologies.

2.1. Memory I/O optimized scheduler for complex scientific execution in cloud MapReduce framework

Here, we introduce an entirely novel architecture called MIOOS which is presented in Figure 2. The overall framework of MIOOS is initially examined in this discussion. Next, the discussion of the input/output (I/O) optimization method that has been implemented in the MIOOS framework to minimize the hard drive search time. In this context, it is significant to discuss the memory scheduling method employed to achieve universal memory management. Finally, this section presents a makespan approach that aims to address the issue of intermediate task execution efficiency and its impact on the whole scheduling performance.

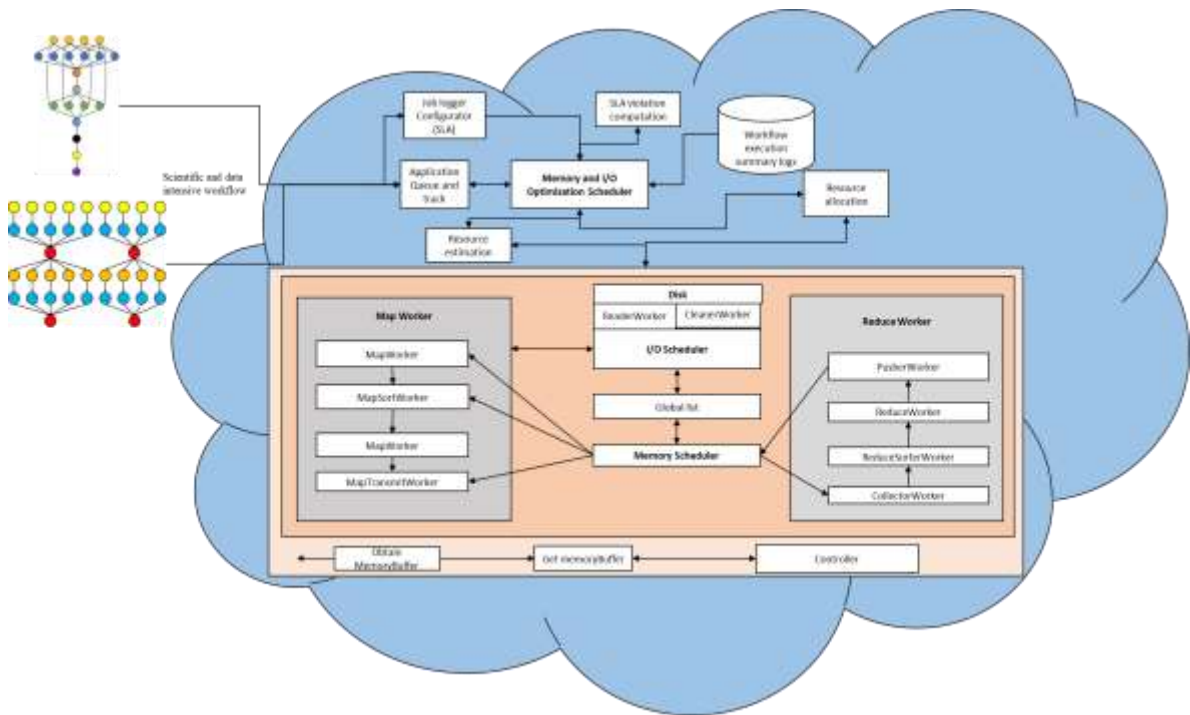


Figure 2. Memory and I/O optimization scheduler for cloud-based MapReduce framework

2.2. System model

The mathematical representation of MIOOS is presented in this section. In the conventional Hadoop-MR structure, the execution of tasks occurs in a distributed manner across multiple nodes. In the MIOOS structure, the execution of the task is facilitated by employing MIOOS in conjunction with the memory-scheduler. Memory management is handled by the memory scheduler, which schedules the release of tasks and new allocations. The memory levels of individual tasks may vary, and these details regarding memory resource availability are easily obtained through the global list. The input/output (I/O) planner initiates communication with the read-worker component to retrieve data stored on disk. Simultaneously, the clean-worker component is responsible for removing information stored in the global list data structure. The MIOOS system implements global allocation of memory by utilizing the global-list data

arrangement framework. In the global list framework, the intermediary information generated by various tasks is systematically organized and stored. The I/O scheduler employs a technique known as multiple-buffers to facilitate the cleaning and reading of data stored on disk. The utilization of memory resources in this manner results in increased efficiency, thereby contributing to the decreased amount of processing time.

2.3. I/O optimization

The I/O scheduler has been specifically developed to enhance parallelization efficiency while minimizing the hard drive seek period through the optimization of Merge-Sort activities. The utilization of conventional I/O scheduling mechanisms can result in elevated I/O latency due to the independent execution of tasks on individual computing nodes with no intercommunication. This work presents a novel approach to executing sequential I/O operations by integrating both central processing unit (CPU) along hard drive I/O functionalities. The I/O planner consists of two main components: the Reader-Worker, responsible for executing reading activities, and the Cleaner-Worker, responsible for executing writing activities. Both the Cleaner-Worker and Reader-Worker components consist of several buffers that possess distinct goals, enabling the effective organization and management of both writing and reading activities. In the present study, we investigate the distinction between dynamic and static I/O, with a particular emphasis on the prioritization of dynamic I/O. The rationale behind this prioritization stems from the fact that dynamic I/O necessitates prompt memory allocations. In contrast, it is observed that dynamic I/O is generally assigned higher priorities than static I/O. The initiation of static I/O occurs exclusively under the circumstance where the buffer being used is incapable of accommodating intermediate information, necessitating the temporary cleansing of the disk. The dynamic I/O response of a buffer that operates using a higher priority is characterized by a higher emphasis on reading activities and a lower emphasis on cleaning activities.

2.4. Memory optimization scheduler

The MIOOS memory scheduler was developed with the following requirements in mind. The allocation of memory resources to buffers of varying sizes necessitates the development of an optimized design to effectively manage this task. Furthermore, it is important to note that various Map-Reduce tasks may exhibit varying memory requirements. Consequently, it becomes imperative to incorporate a structure that facilitates flexible memory allocation. The estimation of the overall size U_T of various buffers is conducted utilizing the Cache-List methodology.

$$u_T = T^\dagger - E_{list_T} - N_{DT} \quad (1)$$

The variable T^\dagger denotes the upper bound on the memory size allocated for archiving intermediate information. E_{list_T} represents the total size of the Data-Pair-List, while N_{DT} represents the total memory consumption of the I/O Scheduler. Identically, the Map-Controller utilizes a memory allocation denoted as MC_T to carry out the mapping task. The computation of this memory allocation is determined by the subsequent as:

$$MC_T = (P_{DT} + Qtrnsm_{DT}, U_T) \quad (2)$$

The variable P_{DT} denotes the size of the Map-Sort buffer, while $Qtrnsm_{DT}$ represents the I/O buffer-size. The computation of the Map-Sort buffer capacity is determined by (3). The user inquiries about the location of NP^\dagger . The Map-Sort maximum capacity for carrying out every task is denoted as $MSort_{max}$. The present MSort buffer size is represented by QP_{DT} , while M_n signifies the overall number of Map tasks presently handled. The computation of the memory capacity RC_T for carrying out a given task is determined by (4).

$$P_{DT} = \{NP^\dagger * N_o \quad NP^\dagger \neq 0 \quad QP_{DT} \quad NP^\dagger = 0 \quad (3)$$

$$RC_T = T_S - MMC_S \quad (4)$$

The MIOOS approach incorporates a memory management strategy that ensures a sufficient amount of memory is reserved for carrying out tasks. This approach effectively mitigates the need for frequently recycling and reusing memory and I/O resources. Upon the successful execution of specific tasks, the system initiates the process of either logging or deregistering from the Map-Controller component. This action facilitates the acquisition of the value denoted as N_o . The Merge-Sorter component is responsible for providing caching data to be stored in the Sort-Buffer, which is obtained through the Map-Controller. On the other hand, the M-Transmit-Worker component is responsible for releasing the cached data that is currently

being shuffled within the Transmit-Buffer and reallocating it back to the Map-Controller. Additionally, it assesses the performance metrics P_{DT} and $Qtrnsm_{DT}$. Each task employs an adaptive strategy for determining its non-parametric upper bound (NP^\dagger) before its completion. The estimated NP^\dagger data is then transmitted through the Map-Controller. Upon the establishment of any modifications to in-memory data, the Map-Controller proceeds to transmit said data through the central controller. Subsequently, the central-controller assumes the responsibility of determining the appropriate heap length for both the Reduce-Controller and Memory-Controller.

In an approach analogous to the Map-task, the reduce-task initiates the process by either logging or unregistering themselves through the Reduce-Controller. The Reduce-Controller is responsible for segmenting the Reduce-Controller-Task (RC_T) within an even way among the running Reduce-tasks. The Collector-Worker is responsible for retrieving the memory resources associated with the Collection-Memory through the Reduce-Controller. At the same time, the Pusher-Worker releases the storage resources coming from the Collector-Memory, while the Reduce-Controller maintains possession of the resource. The Reduce-Controller component communicates with the Central-Controller component to provide improvements on its stored memory stages. These memory stages are capable of being employed for the execution of map tasks in situations where there are limitations on available memory resources. Throughout every stage of task execution, the memory consumption structure of the central controller exhibits variability, indicating the potential for continuous optimization of memory utilization. In instances where the modified heap length of Map-Controller exceeds that of the previous cycle, it can be inferred that no memory limitations are impeding the execution of tasks. Alternatively, in the absence of sufficient resources, it becomes necessary to promptly release memory from its disk. In the present study, we establish a specific threshold value, denoted as Map-Controller MC_T , which is precisely defined in (2). When the memory demand of specific task executions exceeds the threshold value MC_T , specific memory resources are released.

2.5. Makespan model for memory and I/O optimization scheduler

A comprehensive makespan approach is presented to address the issue of improving task execution efficiency in the MIOOS framework. The computation of the makespan C for carrying out a task is easily achieved by utilizing the equation shown in (5). The variable C_T represents the makespan for the beginning worker considering both I/O and memory optimization, while C_M denotes the makespan for carrying out map tasks, and C_R represents the makespan for carrying out reduced tasks. The MIOOS framework has been observed to exhibit superior performance in terms of minimizing makespan and reducing costs for the execution of text and data mining and continuous applications, in comparison with current HMR scheduling techniques. This is achieved through the effective allocation of processing core and memory resources, as illustrated by the experimental findings presented in result section.

$$C = C_T + C_M + C_R. \quad (5)$$

3. RESULTS AND DISCUSSION

The following part focuses on the evaluation of performance metrics, namely makespan, and memory efficiency, in the context of the suggested MIOOS framework compared to the existing reliable workflow scheduling (RWS) approach [24]. MIOOS and RWS have been developed and deployed through the utilization of the Java programming language. The CloudSimSDN-NFV simulator [25] has been employed as the underlying platform for the implementation of these systems. The Cybershake scientific workflow is used for verifying the proposed MIOOS framework. The mathematical complexity of the tasks at hand involves a significant amount of computational processing and input/output operations. The Cybershake structure is known for its computationally demanding nature, particularly concerning CPU and memory usage.

3.1. Makespan performance

The following part addresses the analysis regarding the makespan required to complete the execution of Cybershake workflows with sizes ranging from 30, 50, 100, and 1000. Figure 3 illustrates a visual representation of the makespan achieved over the execution of the Cybershake application employing the MIOOS and RWS scheduling algorithms while considering a diverse range of workflow scenarios such as 30, 50, and 100. Similarly, for a Cybershake workflow size of 1000, the visual representation of the makespan is given in Figure 4. The utilization of MIOOS, as compared to RWS, results in a notable enhancement in average makespan efficiency, with an observed increase of 78.84%.



Figure 3. Makespan efficiency with different Cybershake workflow sizes

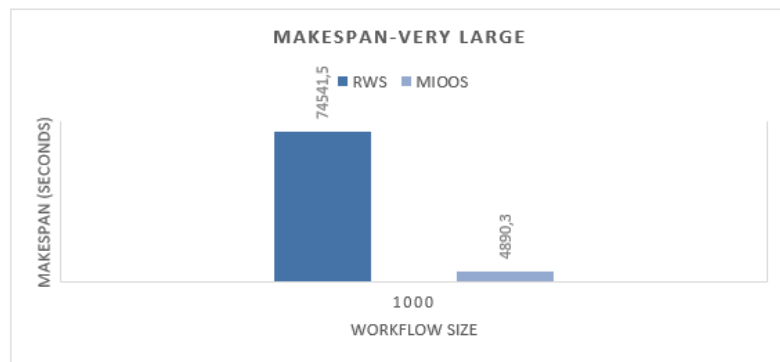


Figure 4. Makespan efficiency with Cybershake workflow size of 1000

3.2. Memory usage

The following part addresses the analysis regarding the memory needed to complete the execution of Cybershake workflows with sizes ranging from 30, 50, 100, and 1000. Figure 5 illustrates a visual representation of the memory efficiency achieved over the execution of the Cybershake application employing the MIOOS and RWS scheduling algorithms while considering a diverse range of workflow scenarios such as 30, 50, and 100. Similarly, for a Cybershake workflow size of 1000, the visual representation of memory usage is given in Figure 6. The utilization of MIOOS, as compared to RWS, results in a notable enhancement in average memory efficiency, with an observed increase of 83.12%.

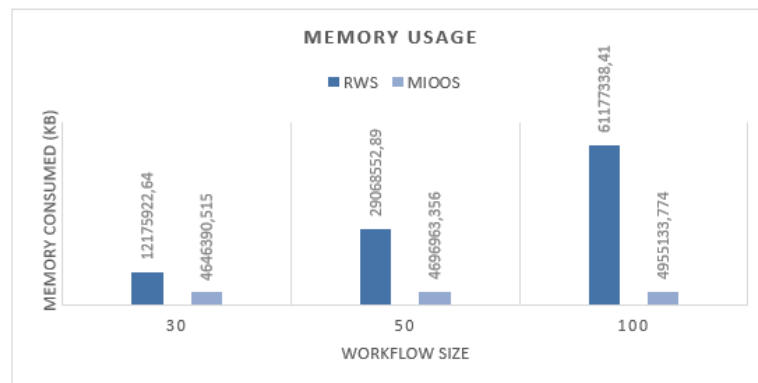


Figure 5. Memory usage with different Cybershake workflow sizes

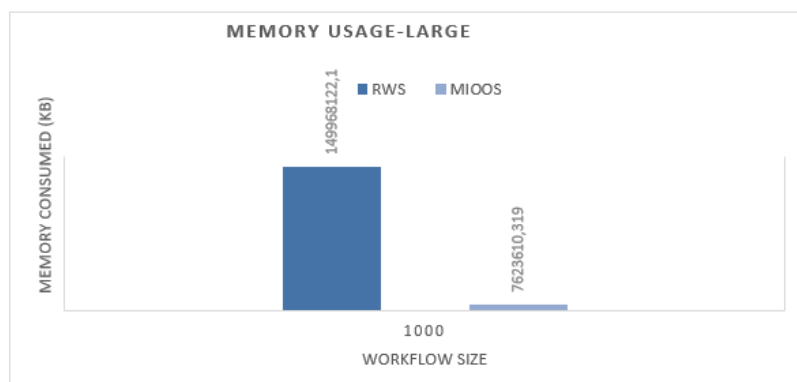


Figure 6. Memory usage with Cybershake workflow size of 1000

4. CONCLUSION

The current identifies the limitation of the current scheduling strategy to meet the strict QoS and SLA requirements of modern scientific workflows. The work addressed major issues such as the I/O scheduler and memory scheduler; the work developed a novel global cache management mechanism to improve workflow execution efficiency. A makespan model considering initializing both I/O and memory scheduler using a thread-based execution model to develop a cloud-based MR model for the execution of scientific workflows. The experimental results demonstrate that the MIOOS system exhibits a high level of makespan and energy efficiency for the execution of Cybershake workflows. Specifically, when compared to the RWS system, MIOOS showcases an impressive increase of 78.84% and 83.12% for energy and memory efficiency, respectively. In the future, it is anticipated that the suggested scheduling approach will undergo testing using a broader dataset of workloads. In addition, it is worth exploring the potential benefits of utilizing an edge-cloud structure to potentially achieve cost reduction and minimize execution delays.

ACKNOWLEDGEMENTS

We would like to express our sincere gratitude to all those who have supported and contributed to this research project. Primarily, we extend our heartfelt thanks to our guide for his unwavering guidance, invaluable insights, and encouragement throughout the research process.





REFERENCES

- [1] G. Papadimitriou *et al.*, "End-to-end online performance data capture and analysis for scientific workflows," *Future generation computer systems*, vol. 117, pp. 387–400, 2021, doi: 10.1016/j.future.2020.11.024.
- [2] R. Ferreira, H. Casanova, Anne-Cécile Orgerie, R. Tanaka, E. Deelman, and F. Suter, "Characterizing, modeling, and accurately simulating power and energy consumption of I/O-intensive scientific workflows," *Journal of Computational Science*, vol. 44, pp. 101157–101157, 2020, doi: 10.1016/j.jocs.2020.101157.
- [3] S. Hedayati, N. Maleki, T. Olsson, F. Ahlgren, M. Seyednezhad, and K. Berahmand, "MapReduce scheduling algorithms in Hadoop: a systematic study," *Journal of Cloud Computing*, vol. 143, 2023, doi: 10.1186/s13677-023-00520-9.
- [4] J. Kim, H. Roh and S. Park, "Selective I/O bypass and load balancing method for write-through SSD caching in big data analytics," *IEEE Transactions on Computers*, vol. 67, no. 4, pp. 589-595, 2018, doi: 10.1109/TC.2017.2771491.
- [5] N. Zhang, M. Wang, Z. Duan and C. Tian, "Verifying properties of mapreduce-based big data processing," in *IEEE Transactions on Reliability*, vol. 71, no. 1, pp. 321-338, 2022, doi: 10.1109/TR.2020.2999441.
- [6] H. Zheng and J. Wu, "Joint scheduling of overlapping mapreduce phases: pair jobs for optimization," in *IEEE Transactions on Services Computing*, vol. 14, no. 5, pp. 1453-1463, 1 2021, doi: 10.1109/TSC.2018.2875698.
- [7] Y. Yao, H. Gao, J. Wang, B. Sheng and N. Mi, "New scheduling algorithms for improving performance and resource utilization in Hadoop YARN clusters," in *IEEE Transactions on Cloud Computing*, vol. 9, no. 3, pp. 1158-1171, 2021, doi: 10.1109/TCC.2019.2894779
- [8] S. Wu, H. Chen, H. Jin and S. Ibrahim, "Shadow: exploiting the power of choice for efficient shuffling in MapReduce," in *IEEE Transactions on Big Data*, vol. 8, no. 1, pp. 253-267, 2022, doi: 10.1109/TBDATA.2019.2943473.
- [9] D. Yang, D. Cheng, W. Rang and Y. Wang, "Joint optimization of MapReduce scheduling and network policy in hierarchical data centers," in *IEEE Transactions on Cloud Computing*, vol. 10, no. 1, pp. 461-473, 2022, doi: 10.1109/TCC.2019.2961653.
- [10] A. Kumar, N. Varshney, S. Bhatiya and K. U. Singh, "Replication-based query management for resource allocation using Hadoop and MapReduce over big data," in *Big Data Mining and Analytics*, vol. 6, no. 4, pp. 465-477, 2023, doi: 10.26599/BDMA.2022.9020026.
- [11] X. Li, F. Chen, R. Ruiz and J. Zhu, "MapReduce task scheduling in Heterogeneous Geo-Distributed data centers," in *IEEE Transactions on Services Computing*, vol. 15, no. 6, pp. 3317-3329, 1 Nov.-Dec. 2022, doi: 10.1109/TSC.2021.3092563.
- [12] R. Jeyaraj and A. Paul, "Optimizing MapReduce task scheduling on virtualized Heterogeneous environments using ant colony optimization," in *IEEE Access*, vol. 10, pp. 55842-55855, 2022, doi: 10.1109/ACCESS.2022.3176729.





- [13] J. Anselmi and N. Walton, "Stability and optimization of speculative queueing networks," in *IEEE/ACM Transactions on Networking*, vol. 30, no. 2, pp. 911-922, 2022, doi: 10.1109/TNET.2021.3128778.
- [14] J. Perez-Valero, A. Banchs, P. Serrano, J. Ortín, J. Garcia-Reinoso, and X. Costa-Pérez, "Energy-aware adaptive scaling of server farms for NFV with reliability requirements," in *IEEE Transactions on Mobile Computing*, vol. 23, no. 5, pp. 4273-4284, 2024, doi: 10.1109/TMC.2023.3288604.
- [15] B. Hu, Z. Cao and M. Zhou, "Energy-minimized scheduling of real-time parallel workflows on Heterogeneous Distributed computing systems," in *IEEE Transactions on Services Computing*, vol. 15, no. 5, pp. 2766-2779, 2022, doi: 10.1109/TSC.2021.3054754.
- [16] B. Mahjani, S. Toor, C. Nettelblad and S. Holmgren, "A flexible computational framework using R and MapReduce for permutation tests of massive genetic analysis of complex traits," in *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 14, no. 2, pp. 381-392, 2017, doi: 10.1109/TCBB.2016.2527639.
- [17] X. Fei and S. Lu, "A dataflow-based scientific workflow composition framework," in *IEEE Transactions on Services Computing*, vol. 5, no. 1, pp. 45-58, 2012, doi: 10.1109/TSC.2010.58.
- [18] S. Mofrad, I. Ahmed, F. Zhang, S. Lu, P. Yang and H. Cui, "Securing big data scientific workflows via trusted Heterogeneous environments," in *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 6, pp. 4187-4203, 2022, doi: 10.1109/TDSC.2021.3123640.
- [19] W. Chen, Y. Yaguchi, K. Naruse, Y. Watanobe and K. Nakamura, "QoS-Aware robotic streaming workflow allocation in cloud robotics systems," in *IEEE Transactions on Services Computing*, vol. 14, no. 2, pp. 544-558, 2021, doi: 10.1109/TSC.2018.2803826.
- [20] S. Hong, J. Choi and W. -K. Jeong, "Distributed interactive visualization using GPU-optimized spark," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 9, pp. 3670-3684, 2021, doi: 10.1109/TVCG.2020.2990894.
- [21] S. Zhang, C. Wang and A. Y. Zomaya, "Robustness analysis and enhancement of deep reinforcement learning-based schedulers," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 1, pp. 346-357, 2023, doi: 10.1109/TPDS.2022.3218649.
- [22] D. Lungu, J. Gerrand, L. Yang, C. Layton and R. Stewart, "Apache spark accelerated deep learning inference for large scale satellite image analytics," in *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 13, pp. 271-283, 2020, doi: 10.1109/JSTARS.2019.2959707.
- [23] S. -G. Lee, E. Kim, J. S. Bae, J. H. Kim and S. Yoon, "Robust end-to-end focal liver lesion detection using unregistered multiphase computed tomography images," in *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 7, no. 2, pp. 319-329, 2023, doi: 10.1109/TETCI.2021.3132382.
- [24] X. Tang, "Reliability-aware cost-efficient scientific workflows scheduling strategy on multi-cloud systems," in *IEEE Transactions on Cloud Computing*, vol. 10, no. 4, pp. 2909-2919, 2022, doi: 10.1109/TCC.2021.3057422.
- [25] J. Son, T. He, and R. Buyya, "CloudSimSDN-NFV: Modeling and simulation of network function virtualization and service function chaining in edge computing environments," *Software: Practice and Experience*, vol. 49, no. 6, 2019, doi: 10.1002/spe.2755.

BIOGRAPHIES OF AUTHOR



Farha Naaz     learned her B.E degree in CSE from K.B.N College of engineering under VTU University, Belagavi in 2012 and master degree in M. Tech. in CSE from K.B.N College of Engineering under VTU University in 2015. Currently she is research scholar at K.B.N University doing her Ph.D. in CSE. Her areas of interest are big data analytics, cloud computing and computer networks. She can be contacted at email farhanaaz_12@rediffmail.com.



Dr. Sameena Banu     is an associate professor in the department of computer science and engineering, faculty of engineering and technology, Khaja Banda Nawaz University, Kalaburgi. She is qualified in bachelor and master degree in computer science and engineering and Ph.D. in computer science and engineering in the area of digital image processing. She is having 21 years of teaching experience. her areas of interest are image processing, machine learning artificial intelligence and internet of things. She can be contacted at this email: sameenabanu271@gmail.com.