Memory aware optimized Hadoop MapReduce model in cloud computing environment

Vaishali Sontakke[1,2], Dayananda R. B.[3]
[1]Vishveshwarya Technological University, Karnataka, India
[2]Department of Information Science and Engineering, East Point College of Engineering and Technology, Bangalore, Karnataka, India
[3]Department of Computer science and Engineering, MSRIT Bangalore, VTU Karnataka, India

Article Info

Article history:

Received Dec 15, 2021
Revised Feb 3, 2022
Accepted Feb 11, 2022

Keywords:

Cloud computing
Hadoop
MapReduce
Parallel computing

ABSTRACT

In the last decade, data analysis has become one of the popular tasks due to enormous growth in data every minute through different applications and instruments. MapReduce is the most popular programming model for data processing. Hadoop constitutes two basic models i.e., Hadoop file system (HDFS) and MapReduce, Hadoop is used for processing a huge amount of data whereas MapReduce is used for data processing. Hadoop MapReduce is one of the best platforms for processing huge data in an efficient manner such as processing web logs data. However, existing model This research work proposes memory aware optimized Hadoop MapReduce (MA-OHMR). MA-OHMR is developed considering memory as the constraint and prioritizes memory allocation and revocation in mapping, shuffling, and reducing, this further enhances the job of mapping and reducing. Optimal memory management and I/O operation are carried out to use the resource inefficiently manner. The model utilizes the global memory management to avoid garbage collection and MA-OHMR is optimized on the makespan front to reduce the same. MA-OHMR is evaluated considering two datasets i.e., simple workload of Wikipedia dataset and complex workload of sensor dataset considering makespan and cost as an evaluation parameter.

Corresponding Author:

Vaishali Sontakke
Department of Information Science and Engineering, East Point College of Engineering and Technology
Bangalore, Karnataka, India
Email: vaishalisontakke12345678@rediffmail.com

1. INTRODUCTION

Recent growth in the deployment of scientific applications, web application and sensor networks tend to generate a huge amount of data. Also, other organizations like industry, government and educational institutions generate a large amount of unstructured data. Hence, analyzing the unstructured data is one of the popular and eminent tasks. Moreover, the state-of-art technique fails to structure in a proper manner considering the real-time scenario enormous challenge.

Google introduced a parallel computational framework named MapReduce [1] which offers the parallel execution through distributed approach. Moreover, Hadoop-MR (HMR) [2] is one of the popular and efficient frameworks adopted for parallel computing in comparison with another framework like Dryad [3], Mars [4], Phoenix [5] where Hadoop-MR is an open-access tool and comprises different steps as initialization, map, shuffle and reduce, further work model is presented in Figure 1. Hadoop-MR comprises two unique phenomena i.e., master node and computing nodes, jobs that are assigned Hadoop are distributed into map step and reduce step. Furthermore, in the initialization part, input data are fragmented into chunks for map nodes. Hadoop divides MapReduce jobs into different task sets where chunk data are processed through map worker.

The mapping step takes input in form of key and value as a pair and output is generated as (key, value) pair. Shuffle step gets initiated after map step completion where (key, value) generated from the map. The sorting task is carried out on the intermediate pair. Moreover, shuffling and sorting are integrated into shuffling. Further, reduce step is processed following defined function. At last, the result of the reduced step is stored in distributed file system (Hadoop-DFS).

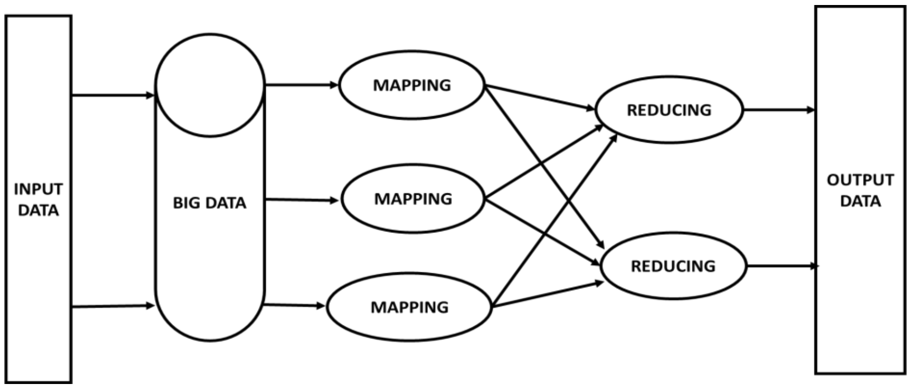

Figure 1. Typical Hadoop MapReduce architecture

Figure 1 shows the Hadoop MapReduce architecture, here the map function uses the pairs of (key, value) to generate the intermediate pairs (key, value). This pair act as the input to Reduce function for producing the final output. MapReduce constitutes two steps: MAP and Reduce. Many platforms provide the flexibility to perform parallel computation, Azure HDInsight is one of them, here user initiates and implies the Hadoop based application to scale the cluster. Moreover, Azure-HD Insights provides the resource required to perform the particular task. However, it fails to provide flexibility and simultaneously also fails to support deadline-based jobs. Moreover, an onus is utilized for the client-side to compute the resource required to meet the task deadline as it is the most challenging task [6]. Hadoop makespan becomes an eminent model in the computation of resources required to meet the task deadline; moreover, Hadoop involves different processing stages that comprise the three-step i.e., map, shuffle and reduces. In the case of the mapping phase, memory and Input/output operation is carried out whereas shuffle comprises sub-phases; first, sub-phase is performed with parallel operation of mapping phase and the second sub phase is carried out after completion Map Phase completion. Thus, it is required to utilize the cloud resource inefficient manner [7], [8]. However, existing approaches fail despite showing potential and fails to cope with efficiency and large overhead (computation) is observed. Moreover, large computation overhead is mainly due to not being able to differentiate between non-overlapping and overlapping.

In the last few years, Hadoop application has observed performance enhancement, few effective methods are developed [9], [10], like scheduling technique Elastciser based on virtual machine for resource allocation, however, this leads to over-prediction in makespan and huge overhead is observed. Developed starfish model which gathers Hadoop task profile [11], [12] uses non-overlapping and overlapping phenomena for task prediction. The designs a mechanism CRESP for task execution and it allows the resource allocation on slots use by [13]. Researchers used educed jobs kept constant that leads memory overhead and I/O which further lead to failed data locality [14]–[16]. Moreover, Soualhia *et al*. [17] used centralized caching and the improvisation is observed [18]–[20]. In developed a mechanism named CRESP for task execution prediction and further allows resource allocation based on MR slots. However, here reduced jobs effects are often discarded. Moreover, reduced jobs are kept constant which leads to the I/O and memory overhead and thus fails to provide awareness regarding data locality [21], [22]. Hence, to address such issues and challenges, Yao *et al*. [23] designed a centralized caching machine and further enhancement and distribution is shown in [24], [25] with distributed caching and shared caching. Wu *et al*. [26] developed a model that provides lockless First in First Out which integrates HMR and external applications; these models were designed for an architecture that is homogeneous and hence they perform unsatisfactorily in heterogeneous architecture as it requires memory and I/O optimization approach.

In traditional HMR, the task is scheduled considering the CPU cores and ignoring resources like memory as it acts as the bridge among the CPU and Input/output device; also memory size available for different phases like mapping and reducing is set in the static configuration, this causes the buffers not to use for reduce task even if they are completed with another phase, also in different phases optimization can be carried out to reduce the total execution time and cost, hence motivated by the application of Hadoop architecture and problem identification. This research work proposed MA-optimized HMR and further contribution of research work is highlighted below.

i. This research work proposes MA-optimized HMR, in here memory is taken into consideration as a resource and HMR is designed where memory allocation is carried out to enhance the MR jobs.
ii. Memory aware optimized Hadoop MapReduce (MA-OHMR) utilized global memory management which avoids the issue of garbage collection.
iii. Further, optimization is carried out in each phase considering memory as the constraint in each phase i.e., mapping, reducing, and shuffling. This optimality helps in efficiently sharing resources and the optimal make span model is introduced to minimize the total execution time.
iv. MA-optimized HMR is evaluated considering make span and computational cost parameter on two distinctive datasets of simple and complex, further evaluation is carried out by varying the workload size.

This particular research is designed as the first section discusses the issue of big data and its processing, further the Hadoop framework is discussed along with its three phases. Furthermore, recent development and challenges are discussed. The first section ends with motivation and contribution of research work. In the second section, different HPMR architecture is discussed along with their shortcomings, the third section proposes MA- optimized Hadoop-P MapReduce along with their architecture and mathematical model. MA-optimized HPMR is evaluated in the fourth section of the research work.

2. RELATED WORK

The last decade has observed evolution in data processing and a variety of approaches is developed and adopted by different researchers, hence this section discusses multiple available scheduling designs for HMR framework, as proposed recently. In an optimization framework has been developed which is designed with cloud computing system for accomplishing data locality and to meet application requirements within deadline prerequisite [9]. Overall, a heuristic way of approach is developed to provide the service level agreements (SLA) tasks for cloud users. Apart from this, an optimization mechanism to reduce the node size required to process the task is developed; at the same time, a single node failure issue is fixed and the tradeoff between the locality constraint and deadline reduction is provided. It is also capable of reducing the makespan and the storage space, but the task deadline constraint is ignored considering the scheduling in data-based applications. The HMR scheduler, location-aware is designed in [10]. The scheduling design depends upon the distance between the processing nodes and the input information which is provided. However, this particular system overcomes several issues like the demand of storage capacity, keeping high overhead and minimum cost in real-time. Although this system works reasonably well but may induce scheduling delay which affects the overall system performance.

Glushkova *et al.* [11] developed a scheduling design for Hadoop-MR framework to reduce the delay and the contention in the provided network to increase the performance; it helps to minimize the synchronization lags and scheduling multiple tasks at once, moreover, theoretical tests were designed to show the fair efficiency over a text-mining application like word-count applications. However, the system is capable to minimize the makespan. The test is not conducted using a cloud-computing environment and how the model will be performing when it is used to execute the complex iterative application is not known experimentally or mathematically. Alshammari *et al.* [12] with metadata information of interrelated tasks modelled improvised HMR framework, in this part, name node is designed to search the block which is preset to the given group to store the specific data. This system is proven to be more efficient than the traditional way of Hadoop-MR model, to test the given model, bioinformatics applications are taken into consideration, and it provides some makespan time reduction and I/O cost minimization, but it will not consider the more complex application and perform only with lower sensitive bioinformatics applications. The narrator developed a prediction intensive model for analyzing the run time of tasks as well as resource allocation to accomplish the task within provided time [13], this helps to achieve the deadline constraint. Also, it takes various sub-phases of the shuffle phase; based on evaluation of the model on Tera-sort as well as Wordcount applications, it is capable of achieving fair efficiency in terms of cost optimization and performance accuracy; at the same time, it possesses high overhead to accomplish the task for complex iterative applications like bioinformatics as well as clustering applications.

Ehsan *et al.* [14] author designed an application called Afford-Hadoop to reduce the effective cost, schedule the task and allocation the data to increase the performance and accuracy; Furthermore, Afford-Hadoop goes with the NP-hard problem of scheduling diversified tasks. For referring such kind of issue mathematical programming model called integer programming, heuristic, and optimization approach is taken into consideration to enable realistic performance. However, performance tests were done on various applications like Tera-sort and Word count; Afford-Hadoop has shown fair minimization of cost, but it failed to provide for other metrics like memory as well as computational core resource utilization. Xiao *et al.* [15] author developed a scheduling strategy to execute relative large-scale data-based applications for minimization of cost with the help of geo-distributed data centers. Moreover, this particular method helps in identifying the parameter to choose the data-center; in this part, a framework is presented to analyses the accuracy information exchange as well as resource allocation, at the same time, task deadline constraint is not taken into consideration. Particle swarm optimization (PSO) based scheduling design is adopted for the HMR framework [16]; the PSO method helps to find the optimal parameters in the HMR framework for the provided task. However, the test is conducted over simple application on the local cluster. It has been seen that 3-phases of MapReduce inclusive of mapping, shuffling, and reducing [23]. Map phase is known to be CPU sensitive whereas I/O intensive and all these phases are performed at the same time. Also, Narrator has tested joint scheduling for overlapping mapping and shuffling for optimization of the makespan. Well, the novel concept of weak and strong pair is introduced. The pair is called strong if the map and shuffle workloads of one job are of equal value to the other. Similarly, it is called weak, if the overall map workload of a given job is of equal value to the shuffle workloads. However, based on strong and weak pairs the jobs are scheduled.

The adopted method is related to the dynamic scheduling for lowering the shuffle traffic as many existing methods failed to include the effect of data centers [24]. In this section, hierarchical topology (HIT) aware MR is introduced to reduce the overall traffic cost which also reduces the execution time taken. Along with that, topology aware assignment (TAA) is adopted considering dynamic communication as well as computation resources in particular cloud with hierarchical architecture. Thereafter, a synergistic strategy is designed to fix the TAA problem through a stable matching mechanism that takes care of in-house preference of hosting as well as individual task machines. Lastly, the use of scheduler as a pluggable module on Y-ARN and performance are tested at the same. Similarly carried out a fair number of surveys and presented the YARN mechanism combined with resource management for scheduling of jobs and it is observed that fairness and efficiency are the main concerns in resource management because resources are shared by various applications [27]. Parallelly, the present scheduling mechanism from YARN will not provide the optimal resource management, therefore, this framework excludes the dependency among the defined that was one of the main matters of concern for resource utilization as well as heterogeneous properties in real-time cases. Apart from this, an improved YARN scheduler is introduced for reducing the makespan time by extra-marginalizing requested information of dependency and resource capacities within tasks. Moreover, it is observed that the scheduler could be extended by job iteration information for scheduling.

Through the survey, it is noticeable how effective handling of memory resource help in lowering makespan and the overall cost of processing data-based application on similar computational framework adopting cloud computing mechanism. From the survey, it is concluded that it is necessary to develop a scheduling technique that minimizes I/O, memory overhead, and reduce makespan with consideration of intermediate task failure for the HMR framework. Extensive survey is conducted where it was observed that slow shuffling is a major reason for any degradation in MR Job execution and only a notable amount of work has been performed for shuffle phase speed optimization [28]. Therefore, a novel procedure was presented for stabilizing the network loads on multiple cross rack links in shuffling phase for various sampling applications and there random processing generates accurate results. It also presents multiple other tasks which provide various choices to select a task while shuffling; At the same time these schemes are developed specifically for sampling-based applications, and they are not properly suitable for general applications where overall data is being processed. Side by side, it is recorded that to consider the high map locality, over shuffling phase networks are saturated, however, in the map phase, it is free. Hence, the conclusion is taken that small sacrifice in Map locality helps in fair shuffling. Thus, a mechanism is designed named shadow only for the general application, which is shuffle constrained and strikes tradeoff between shuffling load balance as well as map locality. In this part, the mechanism selects the original map task in the iterative method from the loaded rack and generates duplicate tasks on the light loaded rack. At the same time, while processing shadow selects option between replicated and the original version by pre-approximation of job makespan.

3. PROPOSED METHODOLOGY

In this section of research, the improvised Hadoop model is designed and developed. Moreover, the execution process is carried out through a virtual machine. Figure 2 represents the architecture of improvised-Hadoop. The proposed Hadoop framework processes the task inside the execution process; further like any other Hadoop framework it has integrated task map, shuffle and reduce (MSR). Proposed architecture retains upper cluster management (CM) i.e., $JObMOnitors$ which acts as the master manages the $TaskMOnitors$. Moreover, when the $TaskMonitors$ receives a new task, when it is assigned to the engine through the procedure call. Moreover, in the execution engine task assigned interacts with cache scheduler which is mainly responsible for memory allocation and coordination of memory. Moreover, scheduler 1 is utilized for fetching the data from the disk and distributing the data to the disk. The proposed Hadoop model utilizes the global memory management using the public pool that contains the basic data structure which helps in enabling efficient memory management. Furthermore, it comprises an element pool that has elements that further consists of a structure of $Input$, $Output$ and a pair of keys and values. Elements store unsorted pair of key and value that is collected by the map function whereas the cache pool holds the sorted intermediate data which is generated by other components in the MR-task. Furthermore, multiBuffer is used by the I/O scheduler where data can be read.

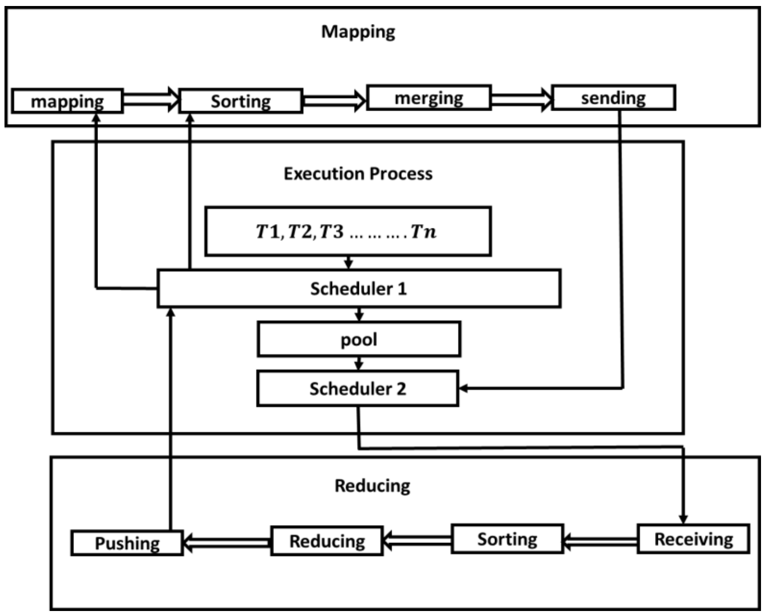

Figure 2. Proposed architecture model

3.1. Optimal mapping phase

In the mapping phase, a component named reader is introduced that reads the input from the Hadoop file structure and fed to a multi-buffer. The mapper demands an empty cell from the pool reads the data from the multi-buffer and gathers the pair. In case, if the element is full then the pair gets sorted, elements that hold the sorted data are proceeded in the queue and thus queue increases when more elements are added. Further, $Sorter$ is used to request the numbers from the pool and data is stored in cache-based units, this is known as a cache file. In the mapping phase, the Reader component reads the input from the Hadoop file structure to the buffer and mapper requests for the M_{output} is the computed average output data size (DS) of the map task, M_{input} is the size of input data(Map Function), $R_{i/p}^{i/p}$ is the defined ratio of reduced output to the reduced input.

$$M_{output} = M_{input} \text{ X } R_{i/p}^{i/p} \tag{1}$$

$$M_{TE} = \frac{M_{ATE} X\ M_{task}}{M_{slot}} \tag{2}$$

i. In the proposed model of Hadoop, the mapper sends a request from the pool reads the given data and gathers the key and value to the new element.
ii. Here the reader reads the input and parts into the various buffer and Mapper sends the particular request for the data and reads from the buffer and stores the information such as key value in the new process, when this is overloaded, it starts sorting once it reaches its thresholds and then the data are sorted in the cache units.
iii. Once the Mapping and Sorting are done then it merges all the files and sends them to the buffer.

3.2. Optimal shuffling

In the existing model data generated through the map, the process gets written in the local disk.S_{avg} is the average data size of shuffle phase, M_{task} and R_{task} are the number of tasks in the map phase and reduce phase task, respectively. S_{TE} is the total execution time, R_{TE} is the total execution time reduced and R_{slot} is the number of the slot at reduce. The main difference between the traditional shuffle phase and MA-optimized HMR is that in the traditional approach data generated in the earlier phase are written to local disk and data is pulled by reducing phase from the local disk. However, in MA-OHMR, further data generated by the map are intentionally pushed to reduce the phase. Thus, when there is insufficient memory, the buffer holds intermediate data.

$$S_{avg} = \frac{S_{output} \, X \, M_{task}}{R_{task}} \tag{3}$$

$$S_{TE} = \frac{R_{TE} \, X \, R_{task}}{R_{slot}} \tag{4}$$

3.3. Optimal reducing

In a typical Hadoop model, all the data to be reduced exist in memory and formulated as: R_{output} is the computed average output DS of map function, R_{input} is input data (reduce function), $R_{i/p}^{i/p}$ is the defined ratio of reduced output to the reduced input.

$$R_{output} = R_{input} \text{ X } R_{i/p}^{i/p} \tag{5}$$

$$R_{TE} = \frac{R_{ATE} X \, R_{task}}{R_{slot}} \tag{6}$$

R_{TE} is the total time taken to execute the map function, R_{ATE} is the total average execution time, R_{slot} is the total number of configured slots. In the MA-optimized HPMR Model, the intermediate data are sorted and then the key, value uses the Reduce Function to process the aggregation and the final output is pushed to Hadoop file system (HDFS).

3.4. Optimal memory allocation

In the Hadoop model, the memory allocation is distributed dynamically since the demand of memory may vary in various stages of the process and an optimized HPMR model provides these strategies to tackle the memory: i) To share the memory dynamically terminology is developed which analyzes the memory use over time and ii) (Key, value) follows the flow of sorting, sending and receiving through each buffer this forms stream hence the priority is set. Moreover, the memory scheduler of MA-OHMR is designed under the condition that MR will have different requirements and buffers size will be varied in size, which aims to design the dynamic memory allocation. Buffers size is computed through cache list with $V^{\uparrow}$ representing the maximum threshold of data, G_{list_V} indicating the list size and P_{F_V} presenting the memory usage through an input-output based scheduler,

$$w_V = V^{\uparrow} - G_{list_V} - \partial \tag{7}$$

Further, in mapping MpC i.e., map controller utilizes memory size to perform mapping with $\wp$ as buffer size of input-output,

$$MpC = \min(\partial + \wp, w_V) \tag{8}$$

Similarly, we formulate the buffer size of the sorting process given with $\mathcal{H}^{\uparrow}$ indicating the maximal size requires for task execution and $\mathcal{E}^{*}$ buffer size of the sorting process,

$$= \begin{cases} \mathcal{H}^{\uparrow} * P_o & \mathcal{H}^{\uparrow} \; is \; not \; equatl \; to \; 0 \\ \mathcal{E}^{*} & \mathcal{H}^{\uparrow} \; is \; equalt \; to \; 0 \end{cases} \tag{9}$$

At last, we compute the reduce controller denoted as $\aleph$,

$$\aleph = \mathcal{T}_S - MMC_S \tag{10}$$

MA-OHMR is designed in a way that model keeps memory available for task execution, which avoids the frequency cycling of input-output resources, and memory.

3.5. Optimization of input/output terminology in HPMR model

In the MA-optimized HPMR model, the input/output follows dual-mode operation i.e., two types of operation are performed Indirect and direct operation. In Indirect, operation occurs when intermediate data cannot be placed in any buffer. The second operation is known as direct operation i.e., when the input is read from the HDFS, and output is written in HDFS. Direct operation is given more priority than indirect one as direct operation runs jobs that are more important. In the case of indirect operation buffer that has higher allocation is given higher read and lower spill priority. For instance, if the sort of buffer is given higher priority than the send buffer, then send buffer has a high spill for memory generation.

3.6. Optimization of makespan model

Further, we design an optimal way of total execution time for mitigation of task failure in the proposed model, let us consider any parameter E indicating total execution time (TET) and job execution is formulated using the (11) where, $E_{initial}$ indicates work initialization, E_{mf} indicates mapping job and E_{rf} indicates reduction job. Consider another parameter q which indicates the worker that comprises p core with z memory size. Average makespan is formulated as:

$$E = E_{initial} + E_{mf} + E_{rf} \tag{11}$$

$$E_{mf} = \sum_{c=1}^{s} E_{avgmf}\ (s)^{-1} \tag{12}$$

In the case of the reducing phase, the average makespan is formulated as:

$$E_{rf} = \sum_{c=1}^{c} E_{avgrf}\ (s)^{-1} \tag{13}$$

Thus, using the above two equations, the total execution time can be formulated as:

$$E = E_{initial} + \sum_{c=1}^{s}\left(E_{avgmf} + E_{avgrf}\right)(s)^{-1} \tag{14}$$

Also, job execution of different phases varies in the proposed model. Hence upper bound and lower bound is computed for job M. The total execution time of job M for best-case scenario is formulated through (15). In the (15) and (16), the least execution time required for performing on job M is denoted through $W_{map}^{N_{lim}}$, further minimum execution time required for map and reduce task are denoted as:

$$W_{M}^{N_{lim}} = W_{map}^{N_{lim}} + W_{reduce}^{N_{lim}} - \left(W_{map}^{w_{lim}} - W_{map}^{N_{lim}}\right) \tag{15}$$

$$W_{M}^{N_{lim}} = W_{map}^{W_{lim}} + W_{reduce}^{W_{lim}} - \left(W_{map}^{W_{lim}} - W_{map}^{N_{lim}}\right) \tag{16}$$

Thus, the total execution time of job M in the proposed model is computed using by (17).

$$\overrightarrow{\mathrm{W}}_{\mathrm{M}} = 0.5\left(W_{M}^{w_{lim}} + W_{M}^{N_{lim}}\right) \tag{17}$$

Further, using best-case and worst-case scenarios, total execution time is formulated:

$$\overrightarrow{\mathrm{W}}_{\mathrm{M}} = \mathrm{o.}\,5\begin{pmatrix}\left(W_{map}^{W_{lim}} + W_{reduce}^{W_{lim}} - \left(W_{map}^{W_{lim}} - W_{map}^{N_{lim}}\right)\right) + \\ \left(W_{map}^{N_{lim}} + W_{reduce}^{N_{lim}} - \left(W_{reduce}^{W_{lim}} - W_{map}^{N_{lim}}\right)\right)\end{pmatrix} \tag{18}$$

The above equation (18) can be simplified and formulated as mention below in equation (19):

$$\overrightarrow{\mathrm{W}}_{\mathrm{M}} = 0.5\left(3W_{map}^{N_{lim}} + W_{reduce}^{N_{lim}} + W_{reduce}^{W_{lim}} - W_{reduce}^{W_{lim}}\right) \tag{19}$$

The linear regression approach is used as discussed in Institute of Electrical and Electronics Engineers, for data dependency, proposed model is designed to reduce the cost and total execution time for two distinctive applications i.e., text mining and iterative application [29]. Performance evaluation of both applications considering the existing Hadoop model is carried out in the next section.

4. PERFORMANCE EVALUATION

Hadoop is open-source data storage framework that provides the combination of storage and processing; for storage, HDFS is utilized and MapReduce for processing. It provides flexibility to hold a large chunk of data that can be accessed. Thus, Hadoop MapReduce model is one of the popular models for data processing, hence utilizing the traditional architecture, this research work proposes MA-OHMR. In this section, the proposed method is evaluated considering the different parameters and constraints which is discussed later in the same section, Evaluation system parameter includes system configuration of Ubuntu 16 packed with dual-core 16 GB RAM. Furthermore, the Hadoop cluster along with two slaves and the master node is utilized to HDInsight Azure instance [29]. Performance evaluation is carried out on the standard dataset of Wikipedia, which varied up to 1024 MB, also further evaluation is carried out on complex sensor data up to 400 MB. Moreover, parameters such as execution cost and total execution time, also known as makespan [30], [31].

4.1. Computational cost

Computational cost is nothing, but the cost required to complete the task. Figure 3 shows the computational cost on simple workload and complex workload. Evaluation is carried out on different workloads i.e., 200, 400, 800, and 1600. In the case of the workload of size 200, 400, 800 and 1600 cost of the existing model is 0.3406, 0.4363, 0.7220, and 1.4934 respectively whereas the computational cost of the proposed model is 0.3115, 0.3809, 0.6479, and 1.3805, respectively. Similarly, Figure 4 shows complexed workload, 50, 100, 250, and 500 sizes are considered; in the case of workload size 50, execution cost for this size is 0.3282, 0.5143, 1.0506 and 1.9612 for discussed size through the existing model in a respective manner whereas proposed model execution cost for these workloads are 0.3046, 0.4696, 0.9848, and 1.7500, respectively.

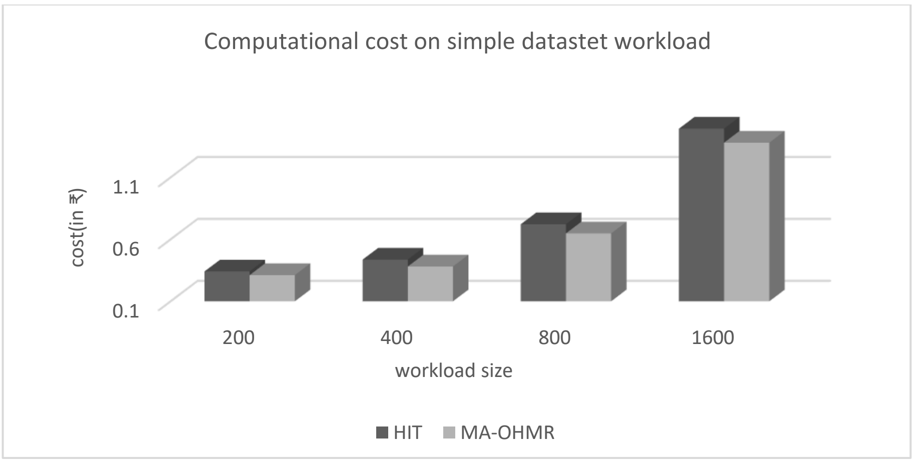

Figure 3. Cost comparison on simple workload

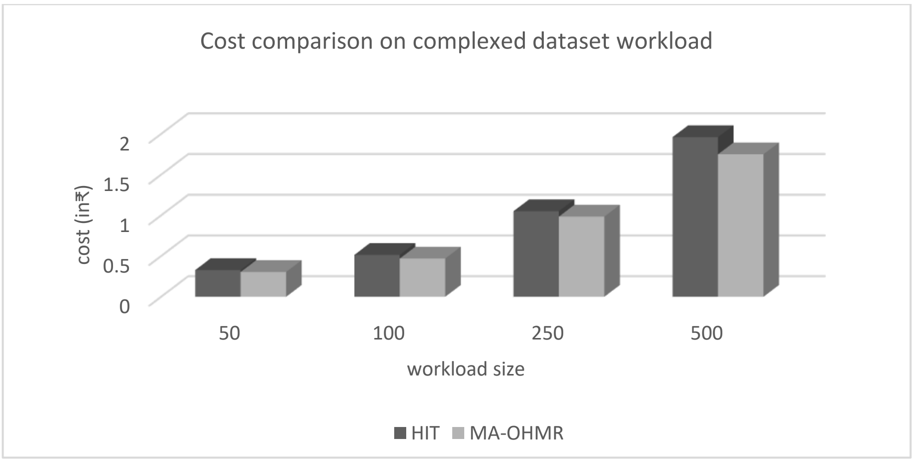

Figure 4. Cost comparison on the complex dataset

4.2. Total execution time

Total execution time is the total time taken to complete the task, less TET is required for the model to be more efficient and optimal for real-time adoption. Considering the TET parameter, we evaluate the simple and complex task, Figures 5 and 6 shows the evaluation of simple and complex workflow, respectively. In Figure 5, various sizes of workload as 200, 400, 800, and 1600 are considered and the existing model observed TET of 310000, 397000, 65700, and 1358900 whereas the proposed model achieves 284520, 347904, 591549.6 and 1260900, respectively. In Figure 6, various sizes of workload as 50, 100, 250 and 500 are considered and the existing model observes TET of 298656, 468000, 956000, and 1784500 whereas the proposed model observes 278283.84, 428928, 899500, and 1598400, respectively.

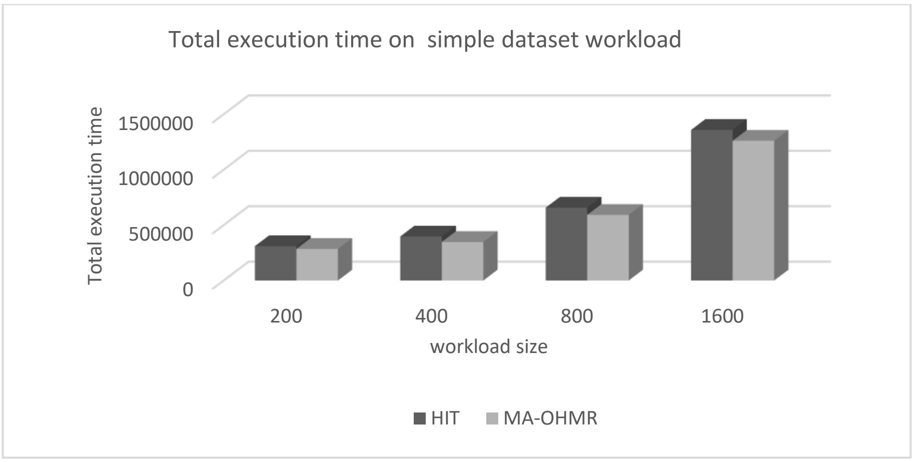

Figure 5. Total execution time (makespan) on simple workload

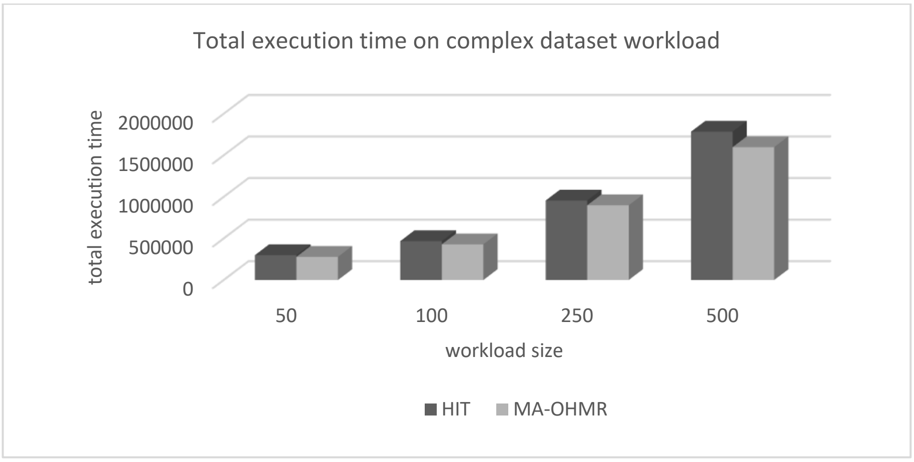

Figure 6. Comparison of makespan on complex workload

4.3. Comparative analysis and discussion

This section discusses improvisation of the proposed model over the existing model through comparative analysis, moreover, the comparison is carried out on two datasets simple and complex considering makespan execution cost as a parameter; in the case of simple workload, the proposed model improvises by 8.21%, 12.36%, 9.91%, and 7.21% for 200, 400, 800, and 1600 respectively for makespan and 8.56%, 12.69%, 10.25%, and 7.56% improvisation observed in terms of execution cost. Similarly, in case of complex dataset for workload size of 50, 100, 250, and 500 proposed model improvises by 7.17%, 8.69%, 6.26%, and 10.76% for execution cost and 6.82%, 8.34%, 5.91%, and 10.42%, respectively.

5. CONCLUSION

A huge quantum of data is continuously generated through various instruments and networks and to address that MapReduce is a commonly used tool and models which is adapted by various datacenters deployed across globe. However still improvisation is needed to optimize its performance even with at most precision. In order to answer this challenge, the author has come up with MA-OHMR architecture for efficient data processing. MA-OHMR utilizes global memory management and memory utilization with at most precision as well optimal make span model is designed considering the memory which reduces the total execution time of jobs which terms minimize the computation cost. To evaluate, simple workload and complex workload are considered as a dataset. Moreover, data processing evaluation shows the marginal improvisation of up to 10% in terms of cost-efficient and total execution time aka make span. MapReduce model architecture has been exploited for quite a long, this research tries to solve memory issues through optimization. In the real scenario, data might be more complicated and in huge quantity, hence different aspect has to be looked including more parameter. The proposed model tries to optimize the overall performance and make the MapReduce model even more competitive.

REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008, doi: 10.1145/1327452.1327492.

[2] C. T. Chen, L. J. Hung, S. Y. Hsieh, R. Buyya, and A. Y. Zomaya, "Heterogeneous job allocation scheduler for Hadoop MapReduce using dynamic grouping integrated neighboring search," *IEEE Transactions on Cloud Computing*, vol. 8, no. 1, pp. 193–206, 2020, doi: 10.1109/TCC.2017.2748586.

[3] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3, pp. 59–72, Jun. 2007, doi: 10.1145/1272998.1273005.

[4] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, "Mars: A MapReduce framework on graphics processors," *Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT*, pp. 260–269, 2008, doi: 10.1145/1454115.1454152.

[5] K. Taura, T. Endo, K. Kaneda, and A. Yonezawa, "Phoenix: a parallel programming model for accommodating dynamically joining/leaving resources," *Architecture d'Aujourd Hui*, no. 447, pp. 6–7, 2022, doi: 10.1145/966049.781533.

[6] U. Kang, C. E. Tsourakakis, and C. Faloutsos, "PEGASUS: Mining peta-scale graphs," *Knowledge and Information Systems*, vol. 27, no. 2, pp. 303–325, 2011, doi: 10.1007/s10115-010-0305-0.

[7] X. Cui, X. Lin, C. Hu, R. Zhang, and C. Wang, "Modeling the performance of MapReduce under resource contentions and task failures," *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, vol. 1, pp. 158–163, 2013, doi: 10.1109/CloudCom.2013.28.

[8] X. Lin, Z. Meng, C. Xu, and M. Wang, "A practical performance model for Hadoop mapreduce," *Proceedings - 2012 IEEE International Conference on Cluster Computing Workshops, Cluster Workshops 2012*, pp. 231–239, 2012, doi: 10.1109/ClusterW.2012.24.

[9] M. Xu, S. Alamro, T. Lan, and S. Subramaniam, "CRED: Cloud right-sizing with execution deadlines and data locality," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 12, pp. 3389–3400, 2017, doi: 10.1109/TPDS.2017.2726071.

[10] M. Khan, Y. Liu, and M. Li, "Data locality in Hadoop cluster systems," *2014 11th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2014*, pp. 720–724, 2014, doi: 10.1109/FSKD.2014.6980924.

[11] D. Glushkova, P. Jovanovic, and A. Abelló, "Mapreduce performance model for Hadoop 2.x," *Information Systems*, vol. 79, pp. 32–43, 2019, doi: 10.1016/j.is.2017.11.006.

[12] H. Alshammari, J. Lee, and H. Bajwa, "H2Hadoop: Improving hadoop performance using the metadata of related jobs," *IEEE Transactions on Cloud Computing*, vol. 6, no. 4, pp. 1031–1040, 2018, doi: 10.1109/TCC.2016.2535261.

[13] M. Khan, Y. Jin, M. Li, Y. Xiang, and C. Jiang, "Hadoop performance modeling for job estimation and resource provisioning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 441–454, 2016, doi: 10.1109/TPDS.2015.2405552.

[14] M. Ehsan, K. Chandrasekaran, Y. Chen, and R. Sion, "Cost-efficient tasks and data co-scheduling with affordHadoop," *IEEE Transactions on Cloud Computing*, vol. 7, no. 3, pp. 719–732, 2019, doi: 10.1109/TCC.2017.2702661.

[15] W. Xiao, W. Bao, X. Zhu, and L. Liu, "Cost-aware big data processing across geo-distributed datacenters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 11, pp. 3114–3127, 2017, doi: 10.1109/TPDS.2017.2708120.

[16] M. Khan, Z. Huang, M. Li, G. A. Taylor, and M. Khan, "Optimizing Hadoop parameter settings with gene expression programming guided PSO," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 3, 2017, doi: 10.1002/cpe.3786.

[17] M. Soualhia, F. Khomh, and S. Tahar, "A dynamic and failure-aware task scheduling framework for Hadoop," *IEEE Transactions on Cloud Computing*, vol. 8, no. 2, pp. 553–569, 2020, doi: 10.1109/TCC.2018.2805812.

[18] Y. Huang, Y. Yesha, M. Halem, Y. Yesha, and S. Zhou, "YinMem: A distributed parallel indexed in-memory computation system for large scale data analytics," *Proceedings - 2016 IEEE International Conference on Big Data, Big Data 2016*, pp. 214–222, 2016, doi: 10.1109/BigData.2016.7840607.

[19] J. Zhang, G. Wu, X. Hu, and X. Wu, "A distributed cache for Hadoop distributed file system in real-time cloud services," *Proceedings - IEEE/ACM International Workshop on Grid Computing*, pp. 12–21, 2012, doi: 10.1109/Grid.2012.17.

[20] L. Lai *et al.*, "ShmStreaming: A shared memory approach for improving Hadoop streaming performance," *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, pp. 137–144, 2013, doi: 10.1109/AINA.2013.90.

[21] 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.

[22] N. Zhang, M. Wang, Z. Duan, and C. Tian, "Verifying properties of MapReduce-based big data processing," *IEEE Transactions on Reliability*, vol. 71, no. 1, pp. 321–338, 2022, doi: 10.1109/TR.2020.2999441.

[23] Y. Yao, H. Gao, J. Wang, B. Sheng, and N. Mi, "New scheduling algorithms for improving performance and resource utilization in Hadoop YARN clusters," *IEEE Transactions on Cloud Computing*, vol. 9, no. 3, pp. 1158–1171, 2021, doi: 10.1109/TCC.2019.2894779.

[24] H. Zheng and J. Wu, "Joint scheduling of overlapping MapReduce phases: Pair jobs for optimization," *IEEE Transactions on Services Computing*, vol. 14, no. 5, pp. 1453–1463, 2021, doi: 10.1109/TSC.2018.2875698.

[25] D. Yang, D. Cheng, W. Rang, and Y. Wang, "Joint optimization of MapReduce scheduling and network policy in hierarchical data centers," *IEEE Transactions on Cloud Computing*, vol. 10, no. 1, pp. 461–473, 2022, doi: 10.1109/TCC.2019.2961653.

[26] S. Wu, H. Chen, H. Jin, and S. Ibrahim, "Shadow: Exploiting the power of choice for efficient shuffling in MapReduce," *IEEE Transactions on Big Data*, vol. 8, no. 1, pp. 253–267, 2022, doi: 10.1109/TBDATA.2019.2943473.

[27] Z. Zhang, L. Cherkasova, and B. T. Loo, "Optimizing cost and performance trade-offs for MapReduce job processing in the cloud," *IEEE/IFIP NOMS 2014 - IEEE/IFIP Network Operations and Management Symposium: Management in a Software Defined World*, 2014, doi: 10.1109/NOMS.2014.6838231.

[28] K. Chen, J. Powers, S. Guo, and F. Tian, "CRESP: Towards optimal resource provisioning for MapReduce computing in public clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1403–1412, 2014, doi: 10.1109/TPDS.2013.297.

[29] Z. Yang, Y. Yao, H. Gao, J. Wang, N. Mi, and B. Sheng, "New YARN non-exclusive resource management scheme through opportunistic idle resource assignment," *IEEE Transactions on Cloud Computing*, vol. 9, no. 2, pp. 696–709, 2021, doi: 10.1109/TCC.2018.2867580.

[30] T. Kajdanowicz, W. Indyk, P. Kazienko, and J. Kukul, "Comparison of the efficiency of MapReduce and Bulk synchronous parallel approaches to large network processing," *Proceedings - 12th IEEE International Conference on Data Mining Workshops, ICDMW 2012*, pp. 218–225, 2012, doi: 10.1109/ICDMW.2012.135.

[31] R. Huerta, T. Mosqueiro, J. Fonollosa, N. F. Rulkov, and I. Rodriguez-Lujan, "Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring," *Chemometrics and Intelligent Laboratory Systems*, vol. 157, no. 1, pp. 169–176, Oct. 2016, doi: 10.1016/j.chemolab.2016.07.004.

BIOGRAPHIES OF AUTHORS

Prof. Vaishali Sontakke is assistance professor in Department of Information Science and Engineering, EPCET, Bengaluru. She has academic experience of more than 15 years. She is a research Scholar doing research in big data memory management and related to her research work one IEEE conference, one UGC approved international journal published, and one patent also filed. She can be contacted at email: vaishalisontakke12345678@rediffmail.com.

Dr. Dayananda R. B is associate professor in Department of Computer Science and Engineering, MSRIT, Bengaluru. He has academic experience of 15+ years and 4 years of Industry experience. He worked as Professor, HOD, IQAC Director, Research center Head and Vice Principal at Prestigious Engineering Institutions. Under his guidance Ph.D. degree is awarded to research scholar from VTU on machine learning in the year 2021. His research mainly focuses on Design and Development of a Cloud Computing Architecture for data security and on machine learning. Research supervisor at VTU guiding 5 students at present. He is reviewer at Springer Journal (Cluster Computing) and also reviewed many papers of Conferences and symposiums. Published more than 25 papers in reputed journals and conferences. He has been felicitated with Governor's Award-a National award for Excellence in Research and Development, SHIKSHA RATAN Award for talented personality's presented on Saturday 24th September 2016 at New Delhi. He can be contacted at email: rbdayanandakit@gmail.com.