Webology (ISSN: 1735-188X)
Volume 18, Number 2, 2021

Resource Efficient Memory Aware Scheduler For Hadoop Mapreduce Framework

JAGADEVI BAKKA[1] , SANJEEV C LINGAREDDY[2]

[1]Research Scholar CSE, Sri Venketeshwara College of Engineering, Bangalore-562157

[2]Prof. & HOD, Dept of CSE , Sri Venketeshwara College Of Engineering, Bangalore-562157

Abstract—MapReduce (MR) has been one of the popular computing framework for analyzing semi-structured and unstructured BigData and processing application in last decade; further Hadoop MR (HMR) framework is an open source platform which is the widely used MR framework. Moreover, existing HMR scheduling design faces major issues like I/O overhead and memory overhead. In this research work, we focus on developing resource efficient memory aware scheduler for HMR namely REMAS-HMR for efficient utilization of system resources and data processing in real time. REMAS-HMR is developed for analyzing the Global Memory Management; thus minimizing the Disk I/O seek. Further, modelled makespan model for mitigating intermediate task failures in REMAS-HMR. The REMAS method are evaluated on the Microsoft Azure HDInsight cloud platform in consideration with text mining and clustering applications, also comparative analysis with the existing model is carried out. Further, comparative analysis shows that REMAS model outperforms existing model in terms of makespan, computational cost, memory utilization, and core-resource utilization.

Keywords—Cloud computing, I/O optimization, Iterative model, MapReduce, Memory Aware, Performance modelling, Resource utilization, Task scheduling.

I. INTRODUCTION

Several organization such as educational institution, government and industry gathers huge volume of unstructured data through various sources like WWW, bioinformatics, social network, sensor network and so on for different purpose. Moreover analyzing these unstructured data has become one of the desired work for various organization; however state-of-art approach fails to perform considering the real time scenario on the stream data. In case of real time scenario, data based platform like google have designed the parallel computational approach named MR (MapReduce) framework [1]; this particular framework offers parallel execution in distributed manner. HMR(Hadoop MapReduce) [2] is one of the popular and widely adopted tool in comparison with other tools like Phoenix [3], Mars[4] and Dryad [5]; as HMR is open source [6]. HMR model comprises various phases which includes Setup, Mapping, shuffling and Reduce; these are shown in figure1; moreover HMR have computing nodes cluster and master node. Further, Jobs assigned to Hadoop are shared into Mapping and Reducing tasks; in setup phase, input data are divided into particular volume known as chunks for Map nodes. Furthermore, Hadoop parts MR (MapReduce) jobs into various task set where each chunk are processed through MapWorker; in general Map phase accepts the input in certain form as $(key_1, value_1)$ key/value and creates further intermediate pair of key/value $(key_2, value_2)$ as an output. Shuffle phase starts after Map phase completion where intermediate key and value pair are gathered from Map Task; sorting is carried out on the intermediate pair of key/value. In general sorting and shuffling are combined in shuffling phase, also reduce phase process the data in accordance with UDF (User Defined Function). At last reduce phase output is written and stored in HDFS aka Hadoop distributed-FS (File system).

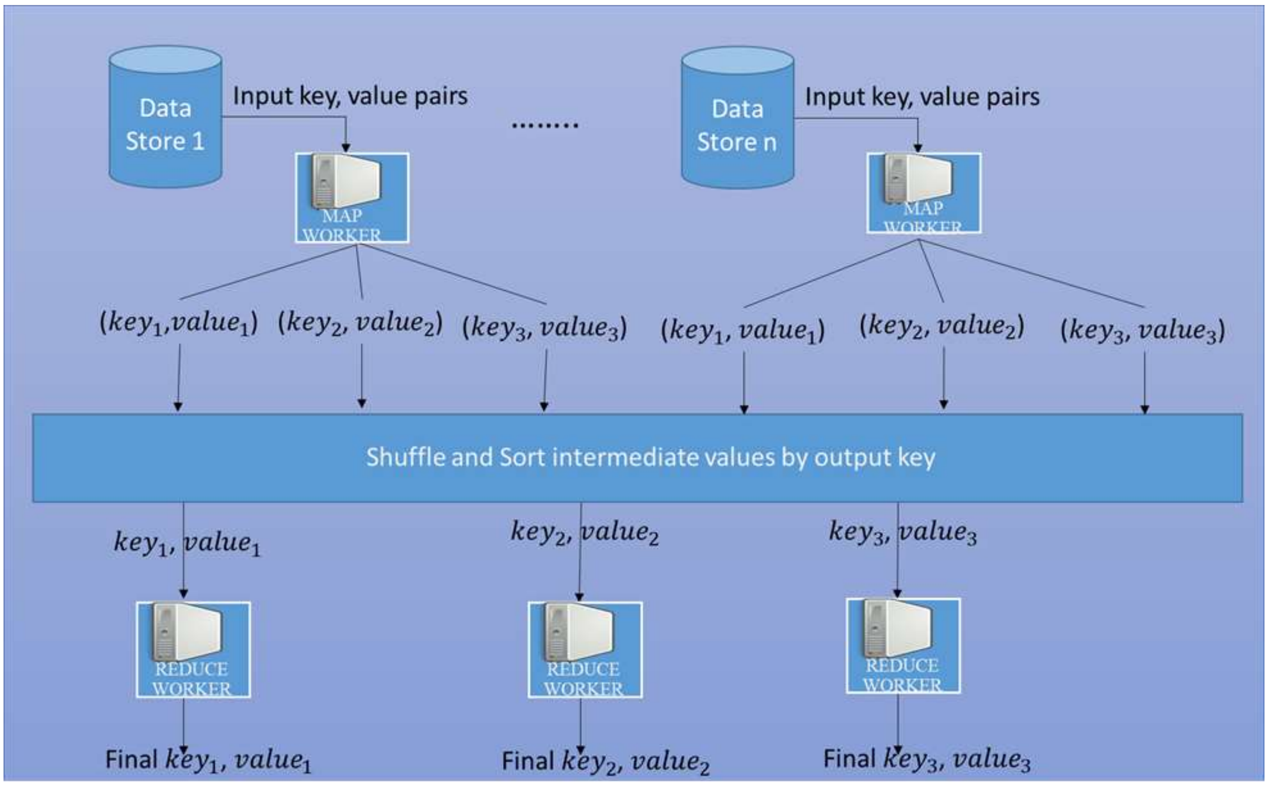

Fig. 1. Hadoop MapReduce (HMR) Model

Azure HDInsight platform provides flexibility to achieve the desired performance; for instance, user sets up and imply the Hadoop application for huge scale cluster. Azure HDInsight provides user for configuring the resource amount needed for performing particular task. Moreover, Azure HDInsight does not provide flexibility and fails to support job with deadline requirement; further, onus is on side of client for computing the resource requirement computation for meeting the task deadline as it is considered as one of the challenging task. Furthermore, Hadoop makespan is very essential model while computing the resource required to maintain the task deadline, also it is observed that Hadoop jobs involves various processing stage which is composed of three integrated phase namely Mapping, Shuffling and Reduction. In here, each phase requires I/O and memory operation; further, in shuffle phase first sub phase are processed in parallel along with Mapping phase (also considered as overlapping phase) and other sub-phase of shuffle phase is processes after completion of Mapping phase (also considered as non-overlapping phase). Moreover, to utilize the cloud resources in more efficient manner various makespan approach were developed in [7] and [8]; these approach does show the potential but fails in terms of efficiency and possesses high computation overhead as these mechanism do not consider overlapping and non-overlapping scenario of shuffle phase.

In past few years, Hadoop application has seen enormous growth and performance enhancement has been observed as well; there are various model of Hadoop some of the effective methods are presented in [9]-[14],[17] and [18]; in [9] developed starfish model that gathers Hadoop task profile for satisfactory granularity. In [10] developed scheduling mechanism named Elasticiser which was based on VM considered (as in starfish model) for resource allocation problem; however it leads to over-predicted task makespan and large overhead while gathering the active task profile (Hadoop task). Further, considering this drawback, [11]-[13] utilizes overlapping and non-overlapping phenomena and to predict the task, conventional LR (Linear regression) is adopted. Moreover these methods also predicts the amount of resources for different task with deadline as constraint. In [14] developed a mechanism named CRESP [28] for task execution prediction and further allows resource allocation based on MR slots; however, in here reduced jobs effect are often discarded. Moreover in [13] and [14] reduced jobs are kept as constant which leads the I/O and memory overhead and thus fails to provide the awareness regarding data locality. Hence, to address such issue and challenges, [17] designed centralized caching machine and further enhancement and distribution is shown in [18] and [19] with distributed caching and in [20] with shared caching. In [21] developed a model that provides lockless FIFO which integrates HMR and external applications; these models were designed for an architecture which is homogeneous in nature and hence they perform unsatisfactory in heterogeneous architecture [21] as it requires memory and I/O optimization approach.

The major factor affecting makespan for executing task in HMR framework is scheduling mechanism during shuffle phase [22], [25], and [26]. In [23] presented joint scheduling scheme considering overlapping of Mapping and shuffling phase for reducing job execution time. In [24] for reducing I/O overhead topology presented aware hierarchical MR scheduler. The model aid in reducing overall I/O traffic and thus aid in reducing job execution makespan. In [25] presented a scheduling scheme namely shadow for increasing speed of shuffling phase execution. The model brings a good tradeoffs among Map task execution locality awareness and balancing load during shuffle phase. In [26] showed the existing doesn’t consider task dependency modelling for executing heterogeneous task; thus failed to achieve effective resource utilization. Thus, they presented an improved YARN scheduling design that reduce makespan of different induvial task. However, the makespan model presented in [23]-[26] doesn’t consider intermediate task failure; affecting makespan performance especially considering complex iterative applications. Thus, makes designing effective scheduling design for HMR framework even more a challenging task.

This research work designs resource efficient memory aware scheduling for HMR framework; REMAS-HMR is very much similar to work carried in [17]; further a thread based execution is considered for optimal memory utilization and minimization of I/O overhead, also this research work focuses on developing a dynamic memory distribution among the task throughout thread in one VM. Furthermore, this research work develops I/O model to improvise memory management for CPU and cross I/O, also REMAS-HMR helps in avoiding the re-reading the data before transmission which minimizes the task through caching final outcome of job in memory. Lastly, present a makespan model with data dependencies for executing simple and complex iterative task.

The significance of REMAS-HMR work:

- The REMAS-HMR is designed using thread based execution realizing global memory management.
- Presented task failure aware makespan model for executing simple and complex application.
- No prior work has considered scheduling considering reducing I/O and memory overhead with task-failure aware makespan model together. As existing model such as [23], [24] addressed I/O overhead and [25], [26] addressed memory utilization overhead.
- REMAS-HMR reduce makespan and computation cost for simple and complex workload; thus improves memory and processing core utilization when compared with existing scheduling design for HMR framework [23]-[26].

II. LITTERATURE SURVEY

This section conduct survey of various existing scheduling design for Hadoop-MR framework proposed in recent times. In [9] designed location-aware HMR scheduler. The scheduling design relies mainly on the distance among the processing nodes and input information. Moreover, this particular technique tries to overcome the several issues such as requirement of storage capacity, high overhead and cost in the real time scenario. Although it performs reasonably well; nonetheless, induces scheduling delay affecting overall system performance. In [10] developed an optimization framework which was designed using cloud computing environment for data locality accomplishment and meet application task deadline prerequisite; in here heuristic approach were developed for provisioning the SLA requirement for cloud user. Further this approach developed an optimization mechanism for reducing the node size needed for processing the task; meanwhile single node failure issue was solved and tradeoff among the locality constraint and deadline reduction is presented. Moreover it was capable of minimizing the makespan and storage requirement; however task deadline constraint were completely ignored in consideration with scheduling in data intensive application.

In [11] using metadata information of correlated tasks modelled improved HMR framework; in here NameNode was designed for finding the block that are preset in given cluster for storing the particular information. This method was proved to be better than the traditional approach of HMR model, in order to evaluate the presented model, bioinformatics application are considered and it achieved some makespan time minimization and I/O cost reduction, but they did not consider more complex application and performed only for lower sensitive bioinformatics application. In [12] presented a scheduling design for HMR framework for minimizing the delay as well as contention in the given network for performance enhancement; it helps in reducing the synchronization delay and scheduling various task at once, also theoretical evaluation were designed which shows the fair accuracy on text mining application such as wordcount applications. Although the methods were able reduce makespan, they experiment is not conducted using cloud computing platform and how model will perform when used for executing complex iterative application is not shown experimentally nor mathematically.

In [13] developed an application named AffordHadoop for minimizing the cost, scheduling the task and allocating the data to enhance the efficiency; however AffordHadoop faces NP-hard problem when scheduling diverse task. For addressing such issue mathematical programming model named integer programming, heuristic, and optimization approach were adoptedfor enabling realistic performance. Moreover performance evaluation were carried out on different applications such asTera-sort and Wordcount; the AffordHadoop achieved fair cost minimization, however fails to deliver for other metrics such as memory and computational core resource utilization. In [14], author develop a prediction based model for predicting the run time of tasks and resource allocation for accomplishing the task in given assigned time; this causes to satisfy the deadline constraint. Further it uses different sub-phase of shuffle phase; upon evaluating the model on Tera-sort and Wordcount applications, it is able to achieve fair performance in terms of cost optimization and accuracy; however it possesses high overhead to complete the task for complex iterative applications such as bioinformatics and clustering applications. In [15] adopted PSO (Particle Swarm Optimization) based scheduling design for HMR framework; PSO mechanism helps in finding the optimal parameters in HMR framework for given specific task; however experiment is conducted on simple application on local cluster. Similarly [16] adopted a scheduling strategy for executing relative large data intensive applications for cost minimization using the geo-distributed data centers; this particular mechanism helps in deciding the parameter for choosing the datacenter; in here a framework is developed for analyzing the efficient information exchange and resource allocation, however task deadline constraint is not considered.

In [23], it is observed that three phases of MapReduce includes mapping, shuffling and reducing; map phase is considered to be CPU sensitive whereas I/O intensive and these phases are performed parallely. Further, author performed joint scheduling for overlapping mapping and shuffling to optimize the makespan; moreover novel concept of weak and strong pair was introduced and pair is said to be strong if map and shuffle workloads of one job is equivalent to other and it is said to be weak, if total map workload of particular job is equivalent to the shuffle workloads. Moreover based on strong and weak pair the jobs are scheduled. Similarly [24] adopted mechanism which was based on the dynamic scheduling for minimizing the shuffle traffic as several existing methodology failed to consider the impact of data centers. In here, Hit(Hierarchical topology) aware MR (MapReduce) was proposed for reducing the overall traffic cost which in terms reduces the execution time; further TAA(Topology Aware Assignment) were developed in consideration with dynamic communication and computation resources in given cloud with hierarchical architecture. Later, synergistic strategy were designed to solve the TAA issue through stable matching mechanism that ensures in house preference of hosting and individual task machines. At last, scheduler is used as pluggable module on YARN and performance were evaluated on same.

In [25] performed an extensive survey and found that slow shuffling is main reason for any degradation in MR job execution and only considerable amount of work has been carried out for shuffle phase speed optimization; hence a novel mechanism was presented for balancing the network loads on various cross rack links in shuffling phase for different sampling applications where random processing generates efficient results. Further, it also introduces several other tasks which offers various choice for selection of task while shuffling; however these schemes were designed for sampling based application only and they were not convenient for general application where whole data is processed. Meanwhile they observed that considering the high map locality, in shuffling phase networks are saturated but in map phase, it is fairly free. Thus it was concluded that little sacrifice in Map locality causes fair shuffling; hence they developed a mechanism named shadow only for the general application that are shuffle constrained and strikes tradeoff among shuffling load balance and map locality. In here, mechanism chooses original map task in iterative manner from the loaded rack and further generates duplicate task on light loaded rack; moreover while processing shadow chooses option among replicated and original version through pre-approximation of job makespan.

In[26] also performed fair amount of survey and introduces YARN mechanism integrated with resource management for scheduling of jobs and they made a point that fairness and efficiency are that major concern in resource management since resources shared by the various applications. Moreover, current scheduling mechanism in YARN does not provide the optimal resource management, hence this framework omits the dependency among the defined which is one of the major concern for resource utilization and heterogeneous characteristics in real time scenario. Further, animproved YARN scheduler was introduced to minimize the makespan time through leveraging requested information of dependency, resource capacities among tasks. Moreover it was noted that scheduler can be extended through job iteration information for scheduling.From survey it is seen how effective handling of memory resource aid in reducing makespan and overall cost of processing data intensive application on parallel computational framework adopting cloud computing environment. From survey it can be state there is need to develop a scheduling methodologies that reduce I/O, memory overhead, and minimize makespan considering intermediate task failure for HMR framework. Thus, this paper present such framework in next section below.

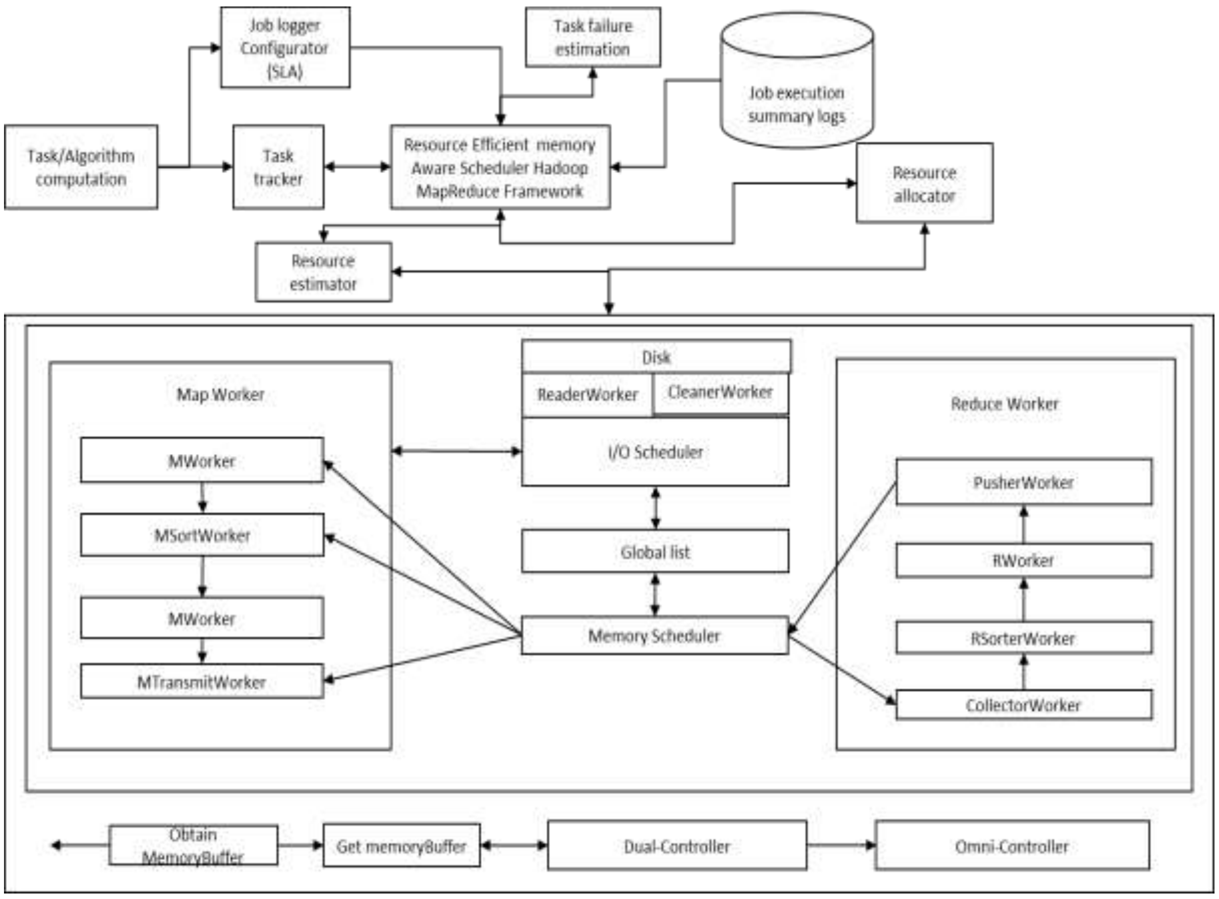

Figure 2. Resource efficient memory aware scheduler for HMR Framework.

III. Resource Efficient Memory Aware Schduler for Hadoop MapReduce Framework

In this section, we present a new framework namely, Resource efficient and memory aware scheduler for HMR Framework as shown in Figure 2.First, discusses the system model of REMAS-HMR. Second, present I/O optimization technique adopted in REMAS-HMR framework to reduce disk seek time. Third, present Memory scheduling technique for realizing global memory management. Lastly, present a makespan model for mitigating intermediate task failure affecting overall scheduling performance.

a) System model of REMAS-HMR framework:

This section present the system model of REMAS-HMR. In standard Hadoop-MR framework, the tasks are executed on different nodes individually. However, in REMAS-HMR framework, the task will be executed using REMAS through MemorySchedular. The MemorySchedular is responsible for allocation and deallocation of memory resources. Here different worker will have different memory level and these information about memory resource capacity can be collected from GlobalList. The I/O scheduler pings ReadWorker for collecting information from disk and CleanWorker cleans information from the GlobalList. The REMAS-HMR realizes global memory management through GlobalList by adopting such data structure mechanism. In GlobalList, the intermediate data of different task are sorted and kept. For reading and cleaning data from the disk the I/O scheduler uses Multiple-buffers. In this way memory resource are utilized more efficiently aiding in reduction of makespan.

b) I/O Scheduler for REMAS-HMR framework:

Here the I/O scheduler is designed in such a way that it will attain better parallelization performance with minimal disk seek time by optimizing the MergeSort operations. Using standard I/O scheduling mechanism induces high I/O latency as jobs are separately in respective computing node without communicating with each other. Thus, this paper introduce a sequential I/O execution combining both CPU and Disk I/O together. The I/O scheduler is composed of ReaderWorker for carrying out reading operation and CleanerWorker for carrying out writing operations. Both ReaderWorker and CleanerWorker is composed of multiple buffers with different priorities which can be utilized to sort the read/write operations. In this work, we have static I/O and dynamic I/O where dynamic I/O has been given high priority; this is because dynamic I/O requires timely memory allocations. On the other side, the static I/O is given lesser priorities when compared with dynamic I/O. The static I/O is started only when intermediate data can't be stored into the buffer and disk are cleaned temporally. The dynamic I/O function on buffer with higher priority composed of high read priority and low clean priority.

c) Memory Scheduler for REMAS-HMR:

The Memory scheduler for REMAS-HMR is designed considering following assumption. First, buffers size will be of varied size; thus for allocating memory resource to these buffer and effective optimization design must be modelled. Second, different MapReduce tasks will have different memory requirement; thus, dynamic memory allocation design is needed. The total size U_T of different buffers is estimated using CacheList as follows

$$u_T = T^{\uparrow} - E_{list_T} - N_{D_T} \tag{1}$$

where $T^{\uparrow}$ represent memory size maximal limit for storing intermediate data, E_{list_T} represent the overall size of DataPairList, and N_{D_T} depicts I/O Scheduler overall memory usage.

In similar manner, the MapController uses memory of size MC_T for executing Map task is computed using following equation

$$MC_T = \min\left(P_{D_T} + Qtrnsm_{D_T}, U_T\right) \tag{2}$$

where P_{D_T} represent MSort buffer size and $Qtrnsm_{D_T}$ defiens I/O buffer size. The MSort buffer size is computed using following equation

$$P_{D_T} = \begin{cases} NP^{\uparrow} * N_o & NP^{\uparrow} \neq 0 \\ QP_{D_T} & NP^{\uparrow} = 0 \end{cases} \tag{3}$$

where $NP^{\uparrow}$ represent MSort maximal size for executing each task, QP_{D_T} defines current MSort buffer size and M_n describes the total Map task current being processed. Then, the ReduceController memory size RC_T for executing task is computed using following equation

$$RC_T = \mathcal{T}_S - MMC_S \tag{4}$$

The REMAS design keep enough memory in reserve for executing task; thus, avoid frequent recycling of memory and I/O resource. When certain task are completed, it logs or deregister from MapController; in this way N_o can be obtained. MergeSorter serves caching information for SortBuffer from MapController and MTransmitWorkerunreserve the caching information which are being shuffled in TranmitBuffer and reallocate it to MapController; and evaluates P_{D_T} and$Qtrnsm_{D_T}$. Every Map tasks estimate their $NP^{\uparrow}$ in dynamic manner prior to completing the task and send the information of $NP^{\uparrow}$to MapController. Then, when any changes in memory information is established the MapController send these information to CentralController and CentralController decides the heap size for MemoryController and ReduceController.

In similar manner to Map task, when a reduce task is started, it logs or unregister itself with ReduceController. Then, ReduceController segment the RC_T in uniform manner between runnable Reduce tasks. CollectorWorker retrieve memory resource for CollectionMemory from ReduceController; simultaneously, PusherWorker free its memory resource from CollectorMemory and ReduceController retain the resource. ReduceController updates the CentralController about its memory levels, which can be reutilized for executing Map task during memory constraints. During entire of process of job execution, memory usage pattern of CentralController will be in varying pattern; thus, memory usage can be optimized in dynamic manner. For example, when adjusted heap size of MapController is higher than previous iteration, in such case there is no memory constraint for executing tasks. Or else, there is no adequate resource and memory has to be freed from the disk as early as possible. In this work we define a threshold of MapControllerMC_T as described in Eq. (2). When certain jobs execution memory requirement reaches beyond MC_T, certain memory resource are freed.

d) Task failure aware Makespan model for REMAS-HMR framework:

This section presents a makespan model for mitigating task failure in REMAS-HMR. The makespan $\mathcal{C}$ of for executing job can be computed using following equation

$$\mathcal{C} = \mathcal{C}_{\mathcal{T}} + \mathcal{C}_{\mathbb{M}} + \mathcal{C}_{\mathbb{R}}. \tag{5}$$

where $\mathcal{C}_{\mathcal{T}}$ define makespan for initialization worker, $\mathcal{C}_{\mathbb{M}}$ depicts map job execution makespan, and $\mathcal{C}_{\mathbb{R}}$ define reduce job execution makespan. Let consider that each worker q is composed n number of core/thread with memory size of x; then the average makespan for executing task can be computed using following equation

$$\mathcal{C}_{\mathbb{M}} = \frac{\sum_{a=1}^{q} \mathcal{C}_{a\_\mathbb{M}}}{q} \tag{6}$$

Similarly, for reduce task average makespan can be computed as

$$\mathcal{C}_{\mathbb{R}} = \frac{\sum_{a=1}^{q} \mathcal{C}_{a\_\mathbb{R}}}{q}. \tag{7}$$

Using Eq. (6) and (7), the total makespan of REMAS can be computed as

$$\mathcal{C} = \mathcal{C}_{\mathcal{T}} + \frac{\sum_{a=1}^{q} (\mathcal{C}_{a\_\mathbb{M}} + \mathcal{C}_{a\_\mathbb{R}})}{q}. \tag{8}$$

The job execution of different phase in REMAS-HMR varies. Thus, we bound these makespan into upper and lower bound for executing job $\mathbb{K}$.The total makespan of $\mathbb{K}^{th}$ job considering best case circumstance in REMAS-HMR framework is computed using following equation

$$\mathcal{U}_{\mathbb{K}}^{\mathbb{L}_{lim}} = \mathcal{U}_{\mathbb{M}}^{\mathbb{L}_{lim}} + \mathcal{U}_{\mathbb{R}}^{\mathbb{L}_{lim}} - (\mathcal{U}_{\mathbb{M}}^{\mathbb{U}_{lim}} - \mathcal{U}_{\mathbb{M}}^{\mathbb{L}_{lim}}) \tag{9}$$

where $\mathcal{U}_{\mathbb{K}}^{\mathbb{L}_{lim}}$ defines minimal makespan time for executing job $\mathbb{K}$, $\mathcal{U}_{\mathbb{M}}^{\mathbb{L}_{lim}}$ represent minimum time for executing map task, $\mathcal{U}_{\mathbb{R}}^{\mathbb{L}_{lim}}$, $\mathcal{U}_{\mathbb{M}}^{\mathbb{L}_{lim}}$ represent minimum time for executing reduce task,$\mathcal{U}_{\mathbb{M}}^{\mathbb{U}_{lim}}$ represent maximum time for executing map task. Similarly, the total makespan of $\mathbb{K}^{th}$ job considering worst case circumstance in REMAS-HMR framework is computed using following equation

$$\mathcal{U}_{\mathbb{K}}^{\mathbb{U}_{lim}} = \mathcal{U}_{\mathbb{M}}^{\mathbb{U}_{lim}} + \mathcal{U}_{\mathbb{R}}^{\mathbb{U}_{lim}} - (\mathcal{U}_{\mathbb{M}}^{\mathbb{U}_{lim}} - \mathcal{U}_{\mathbb{M}}^{\mathbb{L}_{lim}}) \tag{10}$$

The total makespan of job $\mathbb{K}$ on REMAS-HMR platform can be obtained using following equation

$$\vec{\mathcal{U}}_{\mathbb{K}} = \frac{(\mathcal{U}_{\mathbb{K}}^{\mathbb{U}_{lim}} + \mathcal{U}_{\mathbb{K}}^{\mathbb{L}_{lim}})}{2} \tag{11}$$

Using Eq. (9) and (10), the total makespan $\vec{\mathcal{U}}_{\mathbb{K}}$ for executing job$\mathbb{K}$ is estimated using following equation

$$\vec{\mathcal{U}}_{\mathbb{K}} = \frac{\left(\left(\mathcal{U}_{\mathbb{M}}^{\mathbb{U}_{lim}} + \mathcal{U}_{\mathbb{R}}^{\mathbb{U}_{lim}} - (\mathcal{U}_{\mathbb{M}}^{\mathbb{U}_{lim}} - \mathcal{U}_{\mathbb{M}}^{\mathbb{L}_{lim}})\right) + \left(\mathcal{U}_{\mathbb{M}}^{\mathbb{L}_{lim}} + \mathcal{U}_{\mathbb{R}}^{\mathbb{L}_{lim}} - (\mathcal{U}_{\mathbb{M}}^{\mathbb{U}_{lim}} - \mathcal{U}_{\mathbb{M}}^{\mathbb{L}_{lim}})\right)\right)}{2} \tag{12}$$

The simplified representation of total makespan is described below

$$\vec{\mathcal{U}}_{\mathbb{K}} = \frac{(3\mathcal{U}_{\mathbb{M}}^{\mathbb{L}_{lim}} + \mathcal{U}_{\mathbb{R}}^{\mathbb{L}_{lim}} + \mathcal{U}_{\mathbb{R}}^{\mathbb{U}_{lim}} - \mathcal{U}_{\mathbb{M}}^{\mathbb{U}_{lim}})}{2} \tag{11}$$

This work uses similar approach presented in [14], [27], and [28] to model data dependency using linear regression. The REMAS-HMR design minimize makespan and reduce cost for executing text mining and iterative application when compared with existing HMR scheduling methodologies by effective utilization of memory and processing core resource which is experimentally shown below.

IV. RESULT AND DISCUSSION

Here experiment is conducted to evaluate the performance of REMAS-HMR over HaSTE [26]. The system parameter used for experiment analysis is Ubuntu 16 operating system configured with 8GB RAM and two cores. Hadoop cluster with one master and two slave node of identical configuration is used similar to HDInsight Azure A2_v2 instance [29]. Experiment is conducted on simple Wikipedia dataset of size varied from 250 MB to 1 GB [26], [30]. Further, experiment is conducted using complex sensor data of size varied from 100MB to 400 MB [31]. Outcome is measured in terms of makespan and computational cost for executing above workload using respective scheduling mechanism.

a) Simple Wordcount applications:

The makespan outcome achieved for executing simple workload of varied size by HaSTE and REMAS-HMR is shown in Fig. 3. REMAS-HMR reduces makespan by 4.52%, 4.912%, 6.163%, and 7.21% when compared with HaSTE when workload size is 200MB, 400MB, 600MB and 1600MB, respectively. From result obtained it can be stated that REMAS-HMR improves makespan performance by 5.7% on an average when compared with HaSTE. The computational cost induced for executing simple workload of varied size by HaSTE and REMAS-HMR is shown in Fig. 4. REMAS-HMR reduces computational cost by 4.89%, 5.272%, 6.52%, and 7.56% when compared with HaSTE when workload size is 200MB, 400MB, 600MB and 1600MB, respectively. From result obtained it can be state that REMAS-HMR reduce computation cost by 6.06% on an average when compared with HaSTE under varied workload scenarios.

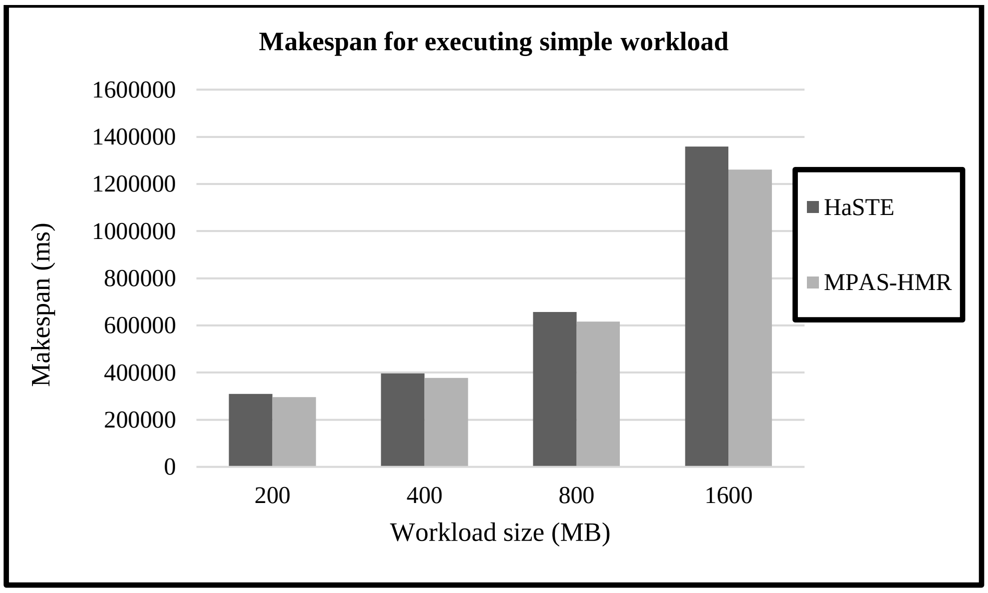

Fig. 3. Makespan performance for executing simple workload.

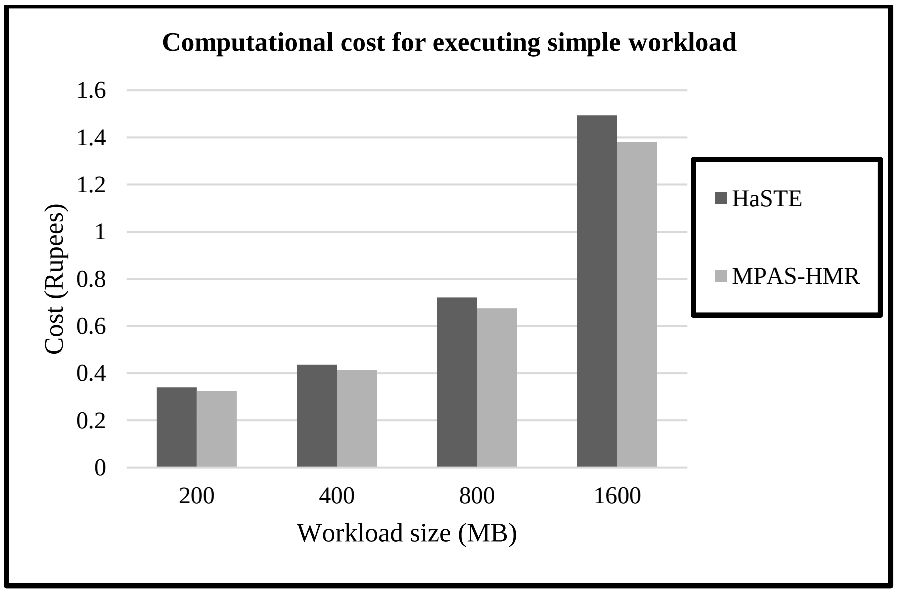

Fig. 4. Computational cost for executing simple workload.

b) Complex Kmeans applications:

The makespan outcome achieved for executing complex workload of varied size by HaSTE and REMAS-HMR is shown in Fig. 5. REMAS-HMR reduces makespan by 2.94%, 4.53%, 5.91%, and 6.7% when compared with HaSTE when workload size is 50MB, 100MB, 250MB and 500MB, respectively. From result obtained it can be state that REMAS-HMR improves makespan performance by 5.01% on an average when compared with HaSTE. The computational cost induced for executing complex workload of varied size by HaSTE and REMAS-HMR is shown in Fig. 6. REMAS-HMR reduces computational cost by 3.31%, 4.892%, 6.27%, and 7.05% when compared with HaSTE when workload size is 50MB, 100MB, 250MB and 500MB, respectively. From result obtained it can be state that REMAS-HMR reduce computation cost by 3.38% on an average when compared with HaSTE under varied workload scenarios.

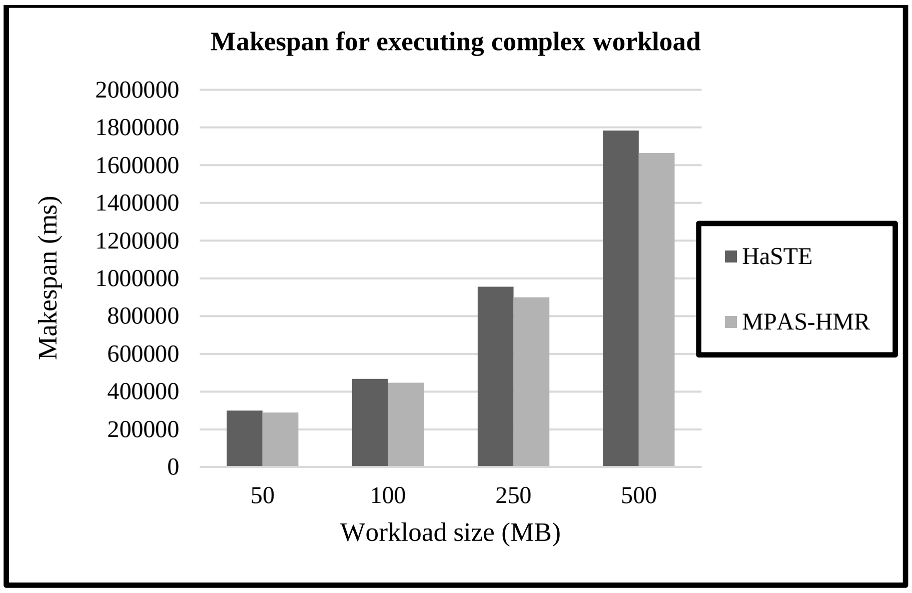

Fig. 5. Makespan performance for executing complex workload.

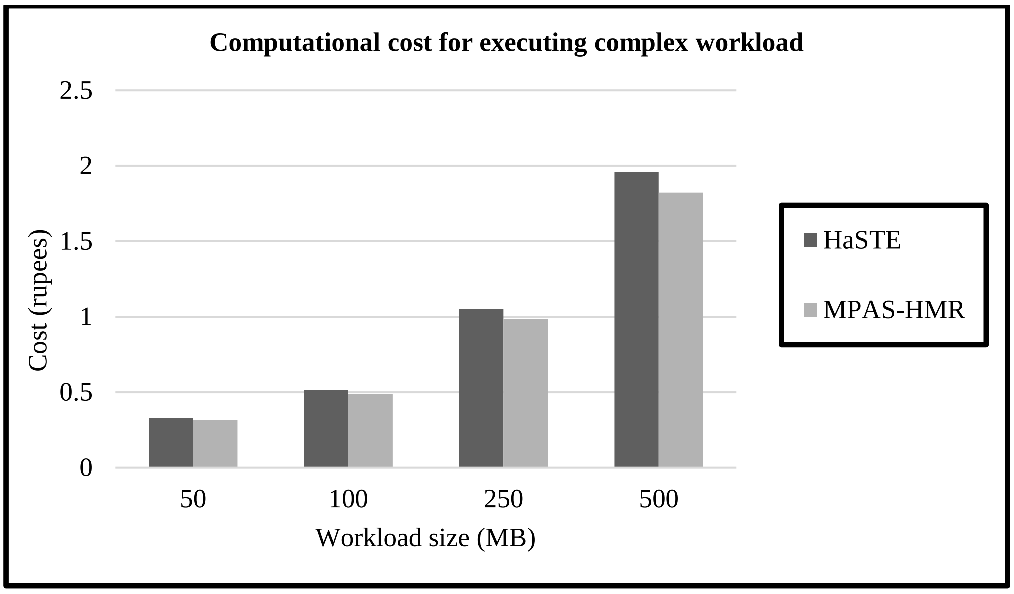

Fig. 6. Computational cost for executing complex workload.

V. CONCLUSION

Managing memory resource is a challenging task. Since different phases of MapReduce job are executed concurrently. This paper presented resource efficient memory aware scheduling adopting dynamic memory management technique and thread based task execution. Further, designed task-failure aware makespan model with dependencies for REMAS-HMR; thus, uses memory resource and multi-core processing resource more efficiently when compared with existing HMR scheduler. Experiments are conducted using simple and complex workload. From result achieved it can be seen the REMAS-HMR reduce makespan and cost by 5.6% and 6.06% when compared with HaSTE for simple workload, respectively. Similarly, REMAS-HMR reduce makespan and cost by 5.01% and 3.38% when compared with HaSTE for complex workload, respectively.

Thus, REMAS-HMR is efficient for running simple and complex iterative task. Though the REMAS-HMR achieves good result; still it is important to test the outcome considering heterogeneous workload. Further, it is important to identify job pattern for improving scheduling mechanism to improve resource utilization.

REFERENCES

1. J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” ACM Commun., vol. 51, no. 1, pp. 107–113, Jan. 2008.
2. “Apache Hadoop.” [Online]. Available: http://hadoop.apache.org/. [Accessed: 21-Oct-2017].
3. K. Taura, T. Endo, K. Kaneda, and A. Yonezawa, “Phoenix: a parallel programming model for accommodating dynamically joining/leaving resources,” in SIGPLAN Not., 2003, vol. 38, no. 10, pp. 216–229.
4. B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, “Mars: a MapReduce framework on graphics processors,” in Proceedings of the 17th international conference on Parallel architectures and compilation techniques - PACT ’08, 2008, p. 260.
5. M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, “Dryad: distributed data-parallel programs from sequential building blocks,” ACM SIGOPS Oper. Syst. Rev., vol. 41, no. 3, pp. 59–72, Mar. 2007.
6. U. Kang, C. E. Tsourakakis, and C. Faloutsos, “PEGASUS: Mining Peta-scale Graphs,” Knowl. Inf. Syst., vol. 27, no. 2, pp. 303–325, May 2011.
7. X. Lin, Z. Meng, C. Xu, and M. Wang, “A Practical Performance Model for Hadoop MapReduce,” in Cluster Computing Workshops (CLUSTER WORKSHOPS), 2012 IEEE International Conference on, pp. 231–239, 2012.
8. X. Cui, X. Lin, C. Hu, R. Zhang, and C. Wang, “Modeling the Performance of MapReduce under Resource Contentions and Task Failures,” in Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on, vol. 1, pp. 158–163, 2013.
9. M. Khan, Y. Liu and M. Li, "Data locality in Hadoop cluster systems," 2014 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), Xiamen, pp. 720-724, 2014.
10. M. Xu, S. Alamro, T. Lan and S. Subramaniam, "CRED: Cloud Right-Sizing with Execution Deadlines and Data Locality," in IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 12, pp. 3389-3400, 2017.
11. H. Alshammari, J. Lee and H. Bajwa, "H2Hadoop: Improving Hadoop Performance using the Metadata of Related Jobs," in IEEE Transactions on Cloud Computing, vol. PP, no. 99, pp. 1-1, 2016.
12. Daria Glushkova, Petar Jovanovic, Alberto Abelló, “MapReduce Performance Models for Hadoop 2.x”, in Workshop Proceedings of the EDBT/ICDT 2017 Joint Conference, ISSN 1613-0073, 2017.
13. M. Ehsan, K. Chandrasekaran, Y. Chen and R. Sion, "Cost-Efficient Tasks and Data Co-Scheduling with AffordHadoop," in IEEE Transactions on Cloud Computing, vol. PP, no. 99, pp. 1-1, 2017.
14. M. Khan, Y. Jin, M. Li, Y. Xiang and C. Jiang, "Hadoop Performance Modeling for Job Estimation and Resource Provisioning," in IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 2, pp. 441-454, 2016.
15. Khan, M., Huang, Z., Li, M., Taylor, GA., – Optimizing Hadoop parameter settings with gene expression programming guided PSO. Concurrency Computation: Practice and Experience, DOI: 10.1002/cpe.3786, 2016.
16. W. Xiao, W. Bao, X. Zhu and L. Liu, "Cost-Aware Big Data Processing Across Geo-Distributed Datacenters," in IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 11, pp. 3114-3127, 2017.
17. Apache, Centralized Cache Management in HDFS. Update date 2014.

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," 2016 IEEE International Conference on Big Data (Big Data), Washington, DC, pp. 214-222, 2016.
19. Zhang, J., et al., A Distributed Cache for Hadoop Distributed File System in Real-Time Cloud Services, in Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing. 2012, IEEE Computer Society. p. 12-21.
20. Longbin, L., et al. ShmStreaming: A Shared Memory Approach for Improving Hadoop Streaming Performance. in Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on. 2013.
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," in IEEE Transactions on Computers, vol. 67, no. 4, pp. 589-595, April 1 2018.
22. N. Zhang, M. Wang, Z. Duan and C. Tian, "Verifying Properties of MapReduce-Based Big Data Processing," in IEEE Transactions on Reliability, doi: 10.1109/TR.2020.2999441, 2020.
23. Zheng, Huanyang & Wu, Jie. (2018). Joint Scheduling of Overlapping MapReduce Phases: Pair Jobs for Optimization. IEEE Transactions on Services Computing. PP. 1-1. 10.1109/TSC.2018.2875698, 2018.
24. 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, doi: 10.1109/TCC.2019.2894779, 2019.
25. 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, doi: 10.1109/TBDATA.2019.2943473, 2019.
26. 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, doi: 10.1109/TCC.2019.2961653, 2019.
27. Z. Zhang, L. Cherkasova and B. T. Loo, "Optimizing cost and performance trade-offs for MapReduce job processing in the cloud," 2014 IEEE Network Operations and Management Symposium (NOMS), Krakow, 2014, pp. 1-8.
28. K. Chen, J. Powers, S. Guo and F. Tian, "CRESP: Towards Optimal Resource Provisioning for MapReduce Computing in Public Clouds," in IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 6, pp. 1403-1412, June 2014.
29. HDInsight cluster pricing https://azure.microsoft.com/en-in/pricing/details/hdinsight/. Last accessed on October 30.
30. Kajdanowicz, T.; Indyk, W.; Kazienko, P.; Kukul, J., "Comparison of the Efficiency of MapReduce and Bulk Synchronous Parallel Approaches to Large Network Processing," Data Mining Workshops (ICDMW), 2012 IEEE 12th International Conference on , vol., no., pp.218,225, 10-10 Dec. 2012.
31. Ramon Huerta, Thiago Mosqueiro, Jordi Fonollosa, Nikolai Rulkov, Irene Rodriguez-Lujan. Online Decorrelation of Humidity and Temperature in Chemical Sensors for Continuous Monitoring. Chemometrics and Intelligent Laboratory Systems 2016.