'National Conference on Recent Advances of Computational Intelligence Techniques in Science, Engineering and Technology'
International Journal of Scientific Research in Computer Science, Engineering and Information Technology | ISSN : 2456-3307 (www.ijsrcseit.com)

Hadoop MapReduce Framework for Performance Aware Scheduling

Jagadevi Bakka*, Sanjeev C Lingareddy, Manimegalai A, Rashmi T V
Department of CSE, East Point College of Engineering and technology, Bangalore, Karnataka, India

ABSTRACT
MapReduce (MR) has been one of the popular computing framework for BigData analysis and processing application in last decade; further Hadoop is an open source platform which is widely used for MR framework. Moreover, existing HMR aka Hadoop-MR model faces major issues like I/O overhead and memory overhead. In this research work, we focus on developing memory and performance awarescheduler design named as MPA-HMR for efficient utilization of system resources and data processing in real time. MPAS-HMR is developed for analyzing the Global Memory Management; thus minimizing the Disk I/O seek. Moreover, MPAS method are evaluated on the Microsoft Azure HDInsight cloud platform in consideration with text mining applications, also comparative analysis with the existing model is carried out. Further, comparative analysis shows that our model outperforms existing model in terms of computation time and computing cost.
Keywords: Cloud computing, MapReduce, Performance modelling, Resource utilization, Task scheduling.

I. INTRODUCTION

Several organizations such as educational institution, government and industry gathers huge 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/continuous data. In case of real time scenario, data-based platform like google have designed the parallel computational approach named MR (MapReduce) framework [Dean and Ghemawat (2008)]; this particular framework offers parallel execution in distributed manner. HMR (Hadoop MapReduce) is one of the popular and widely adopted tool in comparison with other tools like Phoenix [Taura et al. (2003)], Mars [He et al. (2008)] and Dryad [Isard et al. (2007)]; as HMR is open source [Kang et al. (2011)].
HMR model comprises various phases which includes Setup, Mapping, shuffling and reduce; these are shown in Fig. 1; 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 Map Worker; in general Map phase accepts the input in certain form as (K_1,V_1) key/value and creates further pair of key/value 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).

In past few years, Hadoop application has seen enormous growth and performance enhancement has been observed as well [Lin et al. (2012)], [Cui et al. (2013)]; there are various model of Hadoop some of the effective methods are presented in [Khan et al. (2014)] developed starfish model that gathers Hadoop task profile for satisfactory granularity. In [Xu et al. (2017)] developed mechanism named Elasticiser which was based on VM considered (as in starfish model) for resource allocation problem; however, it leads to over-predicted task run time and large overhead while gathering the active task profile (Hadoop task). Further, considering this drawback, [Glushkova et al. (2017), Ehsan et al. (2017), Khan (2016)] 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 [Wu et al. (2017)] found that slow shuffling is main reason for any degradation in MapReduce and only considerable amount of work has been carried out for shuffle phase speed optimization; hence they presented a new mechanism for balancing the network loads on various cross rack links while sampling and shuffling for application where random processing generates efficient results. However, these schemes were designed for sampling-based application only and they were not convenient for general application where whole data is processed.

In [Yao et al. (2019)] 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 concerns for resource utilization and heterogeneous characteristics in real time scenario. In [Zheng et al. (2018)], it is observed map phase is considered to be CPU sensitive whereas I/O intensive and these phases are performed parallelly.

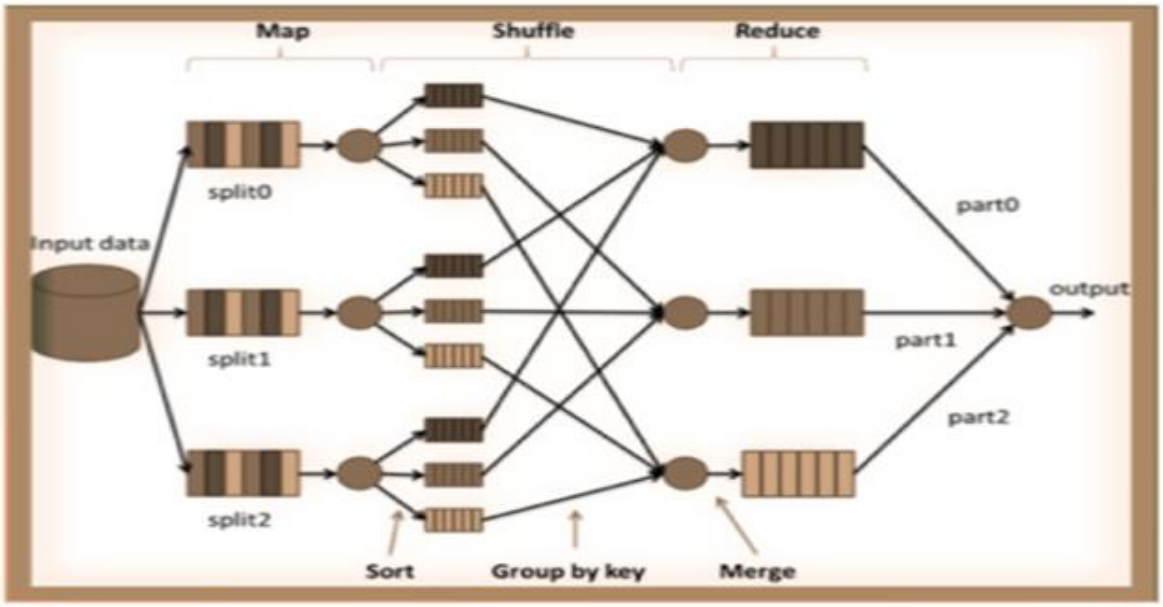

Figure 1. Architecture of Standard Hadoop MapReduce Framework.

Further, author performed joint scheduling for overlapping mapping and shuffling to optimize the makespan. Similarly [Yang et al. (2019)] 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. However, these model do not utilize memory efficiently as jobs are executed at system level rather than thread level.

Moreover to overcome these challenges this research work designs and develops memory constraint aware scheduler for HMR framework namely MPAS-HMR; MPAS-HMR is very much similar to work carried in [Apache (2014)]; further a thread based execution is considered for optimal memory utilization and minimization of I/O overhead [Zhang, J. et al.(2012), Longbin, L., et al.(2013), Kim et al. (2018)], 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 MPAS-HMR helps in avoiding the re-reading the data before transmission which minimizes the task through caching final outcome of job in memory.

Research Contribution are as follows:

- Presented memory and performance aware scheduling design for HMR for executing text mining and iterative application.
- The proposed MPAS design reduces makespan and computational cost for executing text mining and iterative applications when compared with existing scheduling design for HMR [Yao et al. (2019)].

The rest of the paper is organized as follows. In section II the memory and performance aware scheduling design for HMR framework is presented. In penultimate section experimental study is carried out. The conclusion and future work is described in last section.

II. METHODS AND MATERIAL

In this section, we present a new framework namely, Memory and performance aware scheduler (MPAS) design for Hadoop MapReduce Framework as shown in Figure 2.

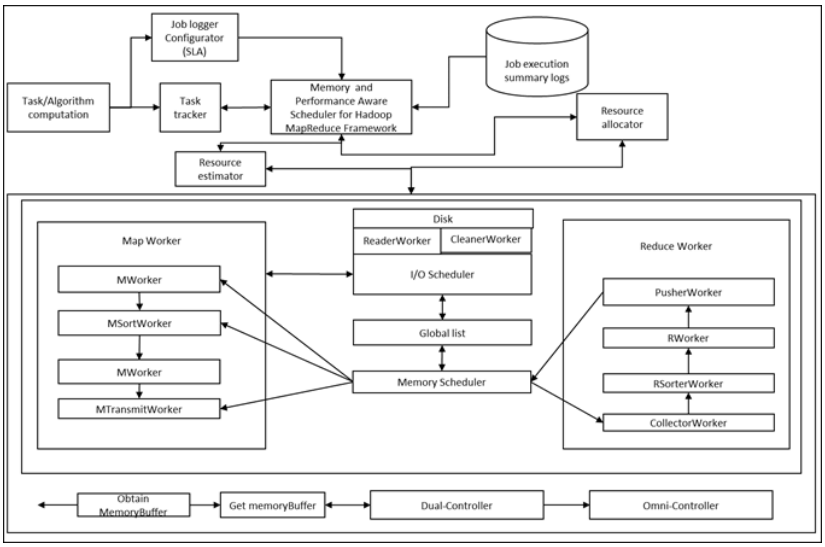

Figure 2. Architecture of Memory and performance Aware Scheduling Design for Hadoop MapReduce Framework.

a) System model:

In standard Hadoop-MR framework, the tasks are executed on different nodes individually. However, in MPAS-HMR framework, the task will be executed 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 MPAS-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) Memory and Performance Aware Scheduling design:

The memory and performance aware scheduler is designed considering following assumption. First, buffers size will be of varied size; thus for allocating memory resource to these buffers an 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^↑-〖E_(〖list〗_T)-N〗_(D_T) (1)

where T^↑ 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⁡(〖P_D〗_T+〖〖Qtrnsm〗_D〗_T,U_T) (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={(NP^↑*N_o NP^↑≠0@〖〖QP〗_D〗_T NP^↑=0)┤ (3)

where NP^↑ 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 Reduce Controller memory size 〖RC〗_T for executing task is computed using following equation

〖RC〗_T=T_S-〖MMC〗_S (4)

The MPAS design keep enough memory in reserve for executing task; thus, avoid frequent recycling of memory and I/O resource aiding in reduction of makespan. The makespan C of for executing job can be computed using following equation

C=C_T+C_M+C_R. (5)

where C_T define makespan for initialization worker, C_M depicts map job execution makespan, and C_R define reduce job execution makespan. Let consider that each workerq 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

C_M=(∑_(a=1)^q▒C_(a_M))/q. (6)

Similarly, for reduce task average makespan can be computed as

C_R=(∑_(a=1)^q▒C_(a_R))/q. (7)

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

$C=C_T+(\sum_{a=1}^{q}(C_{a_M}+C_{a_R}))/q. \quad (8)$

The MPAS design minimize makespan and reduce cost for executing text mining and iterative application when compared with existing HMR scheduling methodologies which is experimentally shown below.

III. RESULTS AND DISCUSSION

Here experiment is conducted to evaluate the performance of MPAS-HMR over HaSTE [21]. 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 [24]. Experiment is conducted on simple Wikipedia dataset of size varied from 250 MB to 1 GB. Further, experiment is conducted using complex sensor data of size varied from 100MB to 400 MB. Outcome is measured in terms of make span and computational cost for executing above workload using respective scheduling mechanism.

A. The makespan outcome achieved for executing simple workload of varied size by HaSTE and MPAS-HMR is shown in Fig. 3. MPAS-HMR reduces makespan by 3.714%, 6.66%, and 8.52% when compared with HaSTE when workload size is 100MB, 200MB, and 1000MB, respectively. From result obtained it can be state that MPAS-HMR improves makespan performance by 6.3% on an average when compared with HaSTE.

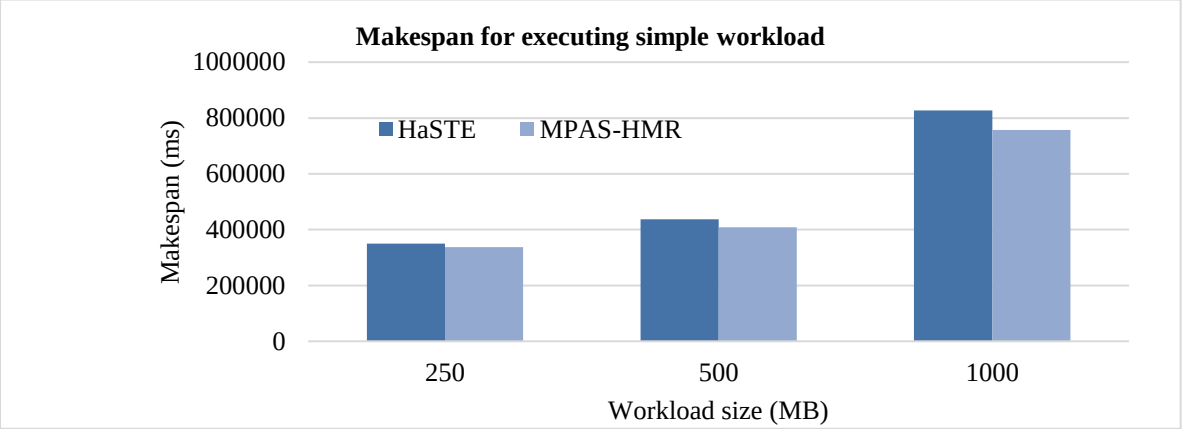

Figure 3. Makespan performance for executing simple workload.

The computational cost induced for executing simple workload of varied size by HaSTE and MPAS-HMR is shown in Fig. 4. MPAS-HMR reduces computational cost by 4.079%, 7.015%, and 8.87% when compared with HaSTE when workload size is 100MB, 200MB, and 1000MB, respectively. From result obtained it can be state that MPAS-HMR reduce computation cost by 6.654% on an average when compared with HaSTE under varied workload scenarios.

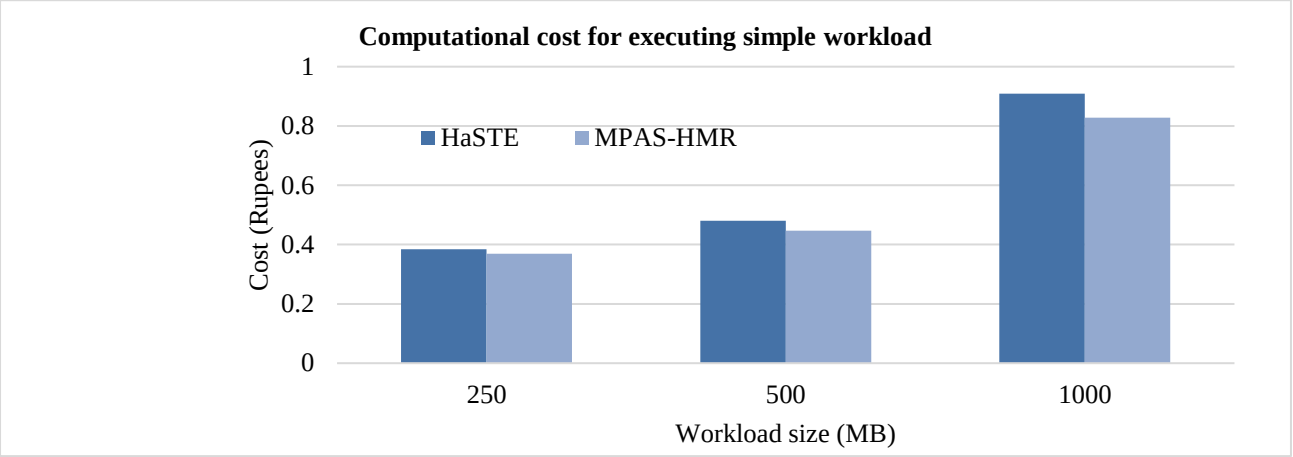

Figure 4. Computational cost for executing simple workload.

The makespan outcome achieved for executing complex workload of varied size by HaSTE and MPAS-HMR is shown in Fig. 5. MPAS-HMR reduces makespan by 1.833%, 2.323%, and 6.226% when compared with HaSTE when workload size is 100MB, 200MB, and 400MB, respectively. From result obtained it can be state that MPAS-HMR improves makespan performance by 3.46% on an average when compared with HaSTE.

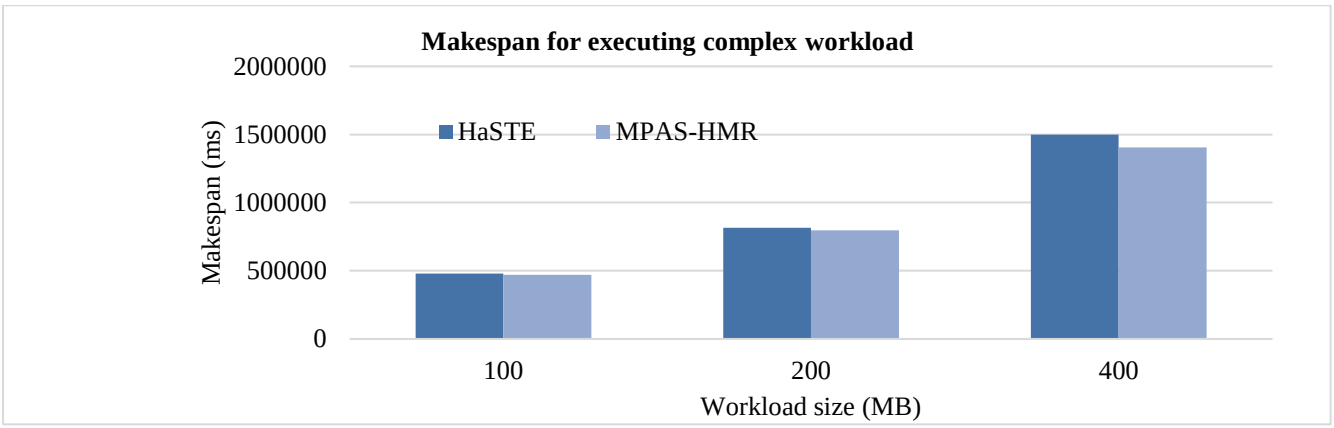

Figure 5. Makespan performance for executing complex workload.

The computational cost induced for executing complex workload of varied size by HaSTE and MPAS-HMR is shown in Fig. 6. MPAS-HMR reduces computational cost by 2.206%, 2.69%, and 6.58% when compared with HaSTE when workload size is 100MB, 200MB, and 400MB, respectively. From result obtained it can be state that MPAS-HMR reduce computation cost by 3.83% on an average when compared with HaSTE under varied workload scenarios.

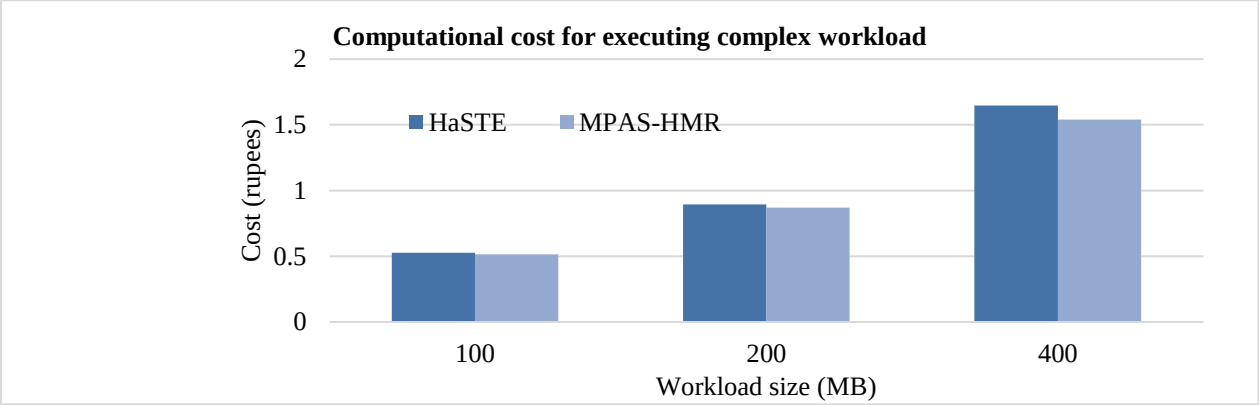

Fig. 6. Computational cost for executing complex workload.

IV. CONCLUSION

Managing memory resource is a challenging task. Since different phases of MapReduce job are executed concurrently. This paper presented memory and performance aware task scheduling adopting dynamic memory management technique and thread based task execution. 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 MPAS-HMR reduce makespan and cost by 6.3% and 6.654% when compared with HaSTE for simple workload, respectively. Similarly, MPAS-HMR reduce makespan and cost by 3.46% and 3.83% when compared with HaSTE for complex workload, respectively. Thus, MPAS-HMR is efficient for running simple and complex iterative task. Though the MPAS-HMR achieves good result; still it is important to test the outcome considering heterogeneous workload. Further, need to evaluate how intermedia task failure affects makespan of scheduling model for HMR.

V. REFERENCES

[1]. Dean and Ghemawat (2008) MapReduce: Simplified Data Processing on Large Clusters, ACM Commun., vol. 51, no. 1, pp. 107–113.

[2]. Taura et al. (2003) Phoenix: a parallel programming model for accommodating dynamically joining/leaving resources, in SIGPLAN Not., , vol. 38, no. 10, pp. 216–229.

[3]. He et al. (2008) Mars: a MapReduce framework on graphics processors, in Proceedings of the 17th international conference on Parallel architectures and compilation techniques - PACT '08, p. 260.

[4]. Isard et al. (2007) Dryad: distributed data-parallel programs from sequential building blocks, ACM SIGOPS Oper. Syst. Rev., vol. 41, no. 3, pp. 59–72.

[5]. Kang et al. (2011) PEGASUS: Mining Peta-scale Graphs, Knowl. Inf. Syst., vol. 27, no. 2, pp. 303–325.

[6]. Lin et al. (2012) A Practical Performance Model for Hadoop MapReduce, in Cluster Computing Workshops (CLUSTER WORKSHOPS), 2012 IEEE International Conference on, pp. 231–239.

[7]. Cui et al. (2013) 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.

[8]. Khan et al. (2014) Data locality in Hadoop cluster systems, 2014 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), Xiamen, pp. 720-724.

[9]. Xu et al. (2017) 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.

[10]. Alshammari et al. (2016) H2Hadoop: Improving Hadoop Performance using the Metadata of Related Jobs, in IEEE Transactions on Cloud Computing, vol. PP, no. 99, pp. 1-1.

[11]. Glushkova et al. (2017) MapReduce Performance Models for Hadoop 2.x, in Workshop Proceedings of the EDBT/ICDT 2017 Joint Conference, ISSN 1613-0073.

[12]. Ehsan et al. (2017) Cost-Efficient Tasks and Data Co-Scheduling with AffordHadoop, in IEEE Transactions on Cloud Computing, vol. PP, no. 99, pp.1-1.

[13]. Khan (2016), Hadoop Performance Modeling for Job Estimation and Resource Provisioning, in IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 2, pp. 441-454.

[14]. Apache (2014), Centralized Cache Management in HDFS.

[15]. Huang et al. (2016) 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.

[16]. Zhang, J. et al.(2012), 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.

[17]. Longbin, L., et al.(2013) ShmStreaming: A Shared Memory Approach for Improving Hadoop Streaming Performance. in Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on. 2013.

[18]. Kim et al. (2018) 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.

[19]. Wu et al. (2017) Shadow: Exploiting the Power of Choice for Efficient Shuffling in MapReduce, IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS), Shenzhen, 2017, pp. 553-560, doi: 10.1109/ICPADS.2017.00078.

[20]. Yao et al. (2019) 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.

[21]. Zheng et al. (2018) Joint Scheduling of Overlapping MapReduce Phases: Pair Jobs for Optimization. IEEE Transactions on Services Computing. PP. 1-1. 10.1109/TSC.2018.2875698.

[22]. Yang et al. (2019) Joint Optimization of MapReduce Scheduling and Network Policy in Hierarchical Data Centers, in IEEE Transactions on Cloud Computing, doi: 10.1109/TCC.2019.2961653.