ISSN 2063-5346

MAKESPAN AWARE OPTIMIZED HYBRID-HADOOP MAPREDUCE MODEL IN CLOUD COMPUTING ENVIRONMENT

Vaishali Sontakke [1], Dr. Chandrakala B M*[2]

Article History: Received: 10.05.2023 Revised: 29.05.2023 Accepted: 09.06.2023

Abstract

The use of high-performance computing (HPC) infrastructure in a cloud computing environment is an effective approach for running data-intensive applications. The MapReduce (MR) framework is a parallel computing solution that is commonly used for high-performance applications that involve the analysis of BigData, scientific research, and data-intensive tasks. Hadoop is a parallel computing framework based on MapReduce that enjoys widespread adoption among diverse organizations. The Apache foundation offers an open source framework that can be obtained at no cost. The current makespan methodology that employs Hadoop MapReduce (HMR) results in memory and I/O overhead, which has an adverse effect on the makespan performance. The proposed publication presents a model called Hybrid HMR (HHMR) makespan, which aims to effectively tackle research problems and obstacles. The Hybrid Hadoop MapReduce (HHMR) technique utilizes virtual computing workers to execute tasks in parallel, resulting in reduced makespan times. This is achieved through the implementation of a cloud computing framework. The HHMR model offers an efficient memory management framework suitable for virtual computing environments. The aforementioned architecture minimizes the amount of memory allocation and transmission overheads. Performance evaluation of the HHMR in comparison to previous models is carried out through experiments on the Azure HDInsight cloud platform, within a public cloud environment. Various applications, including bioinformatics, text mining, stream applications, and non-stream applications, are under consideration. According to the findings, the HHMR model demonstrates better performance than the previous model in reducing makespan time and enhancing the correlation between actual and theoretical makespan values.

[1] Research scholar, Dayananda Sagar College of Engineering, kumaraswamy layout, Bengaluru, India.

[2] Associate Professor, Department of Information Science & Engineering, Dayananda Sagar College of Engineering, Kumaraswamy layout, Bengaluru, India.

*Corresponding Author: chandrakalabm-ise@dayanandasagar.edu

DOI:10.48047/ecb/2023.12.9.15

1 Introduction

Computations for applications that are data intensive as well as scientific use cloud computing as an essential function. Distributed architectural patterns are embraced by cloud computing that can process huge amount of data gathered by different sources. For instance, social network platforms, sensory networks, genome technology, etc. Measurable computations are performed on data which is huge, raw and unstructured being suited to various organisations. The final model includes Phoenix, Mars, Dryad and Spark [1-2] that are inefficient for stream data analysis. Google introduced a parallel computational framework termed MapReduce that examines information intense as well as scientific approaches. Considering MapReduce [3], it is the most wide-scale utilized framework because of it being open source, can be deployed easily and is also scalable. MapReduce proposed by Google was popularized using Hadoop Apache, transformed distributed information processing through simultaneous execution for a group of machines. The use of MapReduce [4] has increased throughput, scalability as well as fault tolerance. Although, efficient use of data is critical for performance optimization of MapReduce. inadequate management of memory could result to higher input/output disk, excessive transfer of data as well as errors that are out of memory, leading to task completion taking longer than normal. Hence, there exists a requirement for an alongside Hadoop MapReduce architecture that aims on efficient memory to focus on these tasks.

The programming model architecture of MapReduce [5] streamlines the distributed information processing by splitting huge amount of information into small clusters that are processed simultaneously across various machines. In the map stage, alongside computations for input information, pairs of key-value are produced. The reduction stage [6] along with intermediate outcomes having the similar key that results in the concluding output. Considering the process, memory has a critical part in intermediate information storage, aiding information transfer as well as management of executional flow.

The Reduce and Map Stage [7] is used to make up the MapReduce Hadoop framework. This is also combined with a shuffle, Setup as well as Sort stage. The MapReduce Hadoop architecture is given in the figure described above. The MapReduce Hadoop cluster is made up of a collection of virtual machines that is linked to master computational virtual machines. The tasks are given to the MapReduce Hadoop Cluster by the user. The master user divides the tasks and assigns them to Reduce and Map user. Considering the Setup stage, the input information from the cloud storage requires processing using Map machines that are divided logically into identical groups.

Considering the Map stage, the master machine allots tasks to the Map machines. The map machines assume input to be the key and the value pairs to be (j_1, u_1) and develop a list of (j_2, u_2) key and valued pairs resulting in [8, 9]. Once the Map stage is complete, initialization of the shuffle stage begins. The shuffle stage gathers the value pairs as well as the intermediary keys from the Map machine. After which a sorting function is implemented to gather the intermediary information of the Map stage. For

convenience, the process of sort as well as shuffle are combined and termed as the Shuffle stage. Finally, considering the user specified operation Reduce stage is started. The result of this stage is stored in the cloud location.

The utilization of memory [10] is crucial for performance optimization of MapReduce tasks. Inadequate management of memory could result in excessive input/output disk, higher transfer of information along the network, resource limitations, that finally lead to impacts on makespan tasks. Hence, efficient memory management is a essential factor for distribution of computational scenarios. Over the years, various researchers have studied efficient memory management for parallel MapReduce Hadoop scenarios and introduced different methodologies to resolve the challenge. Strategies of task schedules are one such means of resolving the problem. Although prior researches have added value to the insights of efficient memory and resource management for parallel MapReduce Hadoop scenarios [11] leaving room for future studies. Recently, advancements in parallel MapReduce Hadoop scenarios aim at improvising efficient resource and memory management. Different techniques are introduced and developed by researchers for optimization of memory usage, scheduling tasks as well as equipping resources dynamically. These advancements involve memory aware information locating techniques, memory management architectures that are cost efficient, efficient methods of memory usage for iterative computational methods. Also, strategies for managing information [12] and reducing its influence on memory utilization are studied. The recent developments add to enhancing efficient memory, decreasing completion time of tasks and improving the total performance for parallel MapReduce Hadoop scenarios.

Considering the analysis mentioned above, it is obvious that there are more limitations of resource utilization in Hadoop [13] as well as accuracy of task modelling for makespan model. Hence, reducing time of makespan and using resources effectively with reduced expenses is required of the cloud computing model.

The makespan model [14] that is proposed in this paper focuses to develop on the prior researches as well as give a complete framework that includes scheduling of tasks, analysed execution, optimization of memory and memory management to improve memory usage as well as decrease task complete time taken.

1.1 Motivation and contribution

Memory utilization and management of resources effectively are critical challenges that are faced by MapReduce Hadoop scenarios. These challenges are to be resolved with the motivation of performance optimization as well as scalable information processing for huge distributed systems. Memory management that in inadequate results in overhead transfer of information, higher input/output disk, effective the complete efficiency of system as well as performance depletion. Also, sub-optimization of resource allocation as well as scheduling of techniques result in underuse of resources and increased time for task completion. The attention given to efficient memory and resource management, is focused at reduced time for task completion, improved

memory usage as well as enhancement of complete performance for distributed information processing. This study is aimed at developing real-time solutions that allow increased efficiency and effectiveness for processing as well as analysing information on large-scale, being advantageous to different domains and industries that require scientific study, financial research, information-intensive applications, etc.

- This paper introduces a novel makespan model specifically designed for Hybrid Hadoop MapReduce jobs. This model takes into account memory utilization as a crucial factor in determining the overall makespan of the job execution.
- By considering memory efficiency, the proposed model aims to optimize resource allocation and reduce the overall execution time and optimizing memory usage and improving overall system performance.
- A memory-aware scheduling strategy that leverages the makespan model to allocate resources efficiently is designed by considering memory utilization as a key factor, the scheduling strategy aims to minimize memory-related issues and enhance overall performance.

2 Related Work

The Hadoop MapReduce model predominantly consist of following phases, Setup, Map, Shuffle, Sort and Reduce. The Hadoop framework consists of a master node and a cluster of computing nodes. Jobs submitted to Hadoop are further distributed into Map and Reduce tasks. The Hadoop MapReduce platform suffers from a number of drawbacks. The preconfigured memory allocator for Hadoop jobs leads to issues of buffer concurrency amongst jobs and heavy disk read seeks. The memory allocator issues result in increasing makespan time and induces high Input/output (I/O) [15].

Assumed homogenous map execution times and serial execution strategy put forth utilized map workers (and their resources) who have completed their tasks and are waiting for the other map workers to complete [16]. In cloud environments where organizations / users are charged according to (storage, compute and communication) resources utilized these issues burden the costs apart from effecting performance [17].

Hadoop platforms do not support flexible pricing [18]. Scalability is an issue owing to the cluster based nature of Hadoop platforms. Processing of streaming data is also an issue with Hadoop . To overcome these drawbacks researchers have adopted various techniques.

In [19], as size of HPC algorithm surges, four noteworthy difficulties comprising programmability, heterogeneity, energy proficiency and fault tolerance have emerged in the basic of distributed computing environment. To handle with every one of them without giving (compromising) up execution, customary methodologies in utilizing resource, programming model and job scheduling are ought to be re-examined. As HMR has dealt with scientific and data-intensive HPC algorithms well in CC platforms, GPU has shown its speeding up viability for calculation of serious scientific and data-intensive HPC algorithms.

In [20], the cutting edge genome sequencing issue with short and long genomic sequences is a developing field in various Big-data and scientific areas. Nevertheless, information sizes and straightforward entry for research scientists are developing and most current philosophies depend on one speeding up methodology thus can't meet the prerequisites forced by massive information scales and complexities. Here, they resented accelerator model by combining FPGA with MR computing model on various hardware. The mixture of MR computing model and hardware accelerator could incredibly quicken the job of performing short-read genomic sequence alignment operation to a known database genomic sequence.

In Hadoop task [21] are executed based on the available CPU cores and memory is allocated based on preset configuration which lead do memory bottleneck due to buffer concurrency and heavy disk seeks resulting I/O wait occupancy which further increases the makespan time. In [22] they addressed the issues related to Hadoop memory management by adopting a global memory management technique. They proposed a prioritization model of memory allocation and revocation by adopting a rule based heuristic approach.

HMR is the most mainstream open-source execution [23-24] of the MR programming framework. In HMR, input records are segmented into numerous small data chunks and these chunks are distributed to number of virtual computing machine a computing clusters. To proficiently process the information chunks, HMR ought to give an effective scheduling design to upgrading the execution of the framework in a shared computing platform. In HMR scheduling predominantly induced by information location awareness problems because of constrained system data transfer capacity.

[25] This study focuses on enhancing Hadoop MapReduce task scheduling to boost efficiency and speed up job completion. It proposes a load balancing and data locality-aware adaptive task scheduling technique.

[26] The following document introduces the Spark framework, which expands the capabilities of the MapReduce model by enabling in-memory processing and iterative algorithms. The technology under discussion presents RDD (Resilient Distributed Datasets) as a highly effective distributed memory abstraction for the purpose of data processing.

[27] The present study introduces a performance model for Hadoop MapReduce, which assists in approximating the duration of task execution and the requisite resources. The model takes into account various factors such as the size of data, job scheduling, and system configuration.

3 Proposed Methodology

This paper introduces a model for optimizing the utilization of unused slots through Hybrid Hadoop MapReduce (HHMR), this model is designed to improve the overall makespan. The initial step involves the description of a system model. Next, it provides a description of the sequential Hadoop MapReduce makespan model. Finally, the Hybrid Hadoop MapReduce makespan model that has been proposed is presented.

3.1 System Architecture

The cloud computing platform is considered as a base in this research, its components are, the master computing node/worker, map and reduce computing nodes make up the Hybrid Hadoop MapReduce (HHMR) makespan framework. Initialization of s maps and virtual computing nodes is done by the master computing node. P computer cores make up each virtual computing node. Let $W_g$represent the amount of time it takes to set up the virtual computing environment. Let W_o be the s map computing workers' average makespan time. Let W_t be the typical amount of time that it has taken for s reduction computing workers to complete shuffle, sort, and reduce computation. Upon assumption that an execution operation i will be assessed in a cloud environment utilizing the MR framework in a heterogeneous computing environment. Equation (1) explains the evaluation by considering U,where the input parameter is shown by d.

$U = i(d)$	(1)

The Map and Reduce framework utilizes virtual computing nodes, also known as workers, to perform computational tasks for the given activity. The execution of g on the cloud computing architecture incurs overhead or costs. The variables R_f and W denote the overhead resulting from input/output operations and computation, respectively. The expense is determined by the makespan that has been accumulated. The acronym W_U^R denotes the total or aggregated makespan.

In the context of a cloud computing environment denoted by y, an input/output (I/O) event is currently occurrs. The function $M(a, p, y)$ denotes the retrieval of p bytes of data from a blob container location in the cloud and storing it in a virtual storage location a on a local machine. The function $MO(y)$ is a representation of the observer for data transfer makespan overhead. The function $DE(b, p)$ is used to save the results of an execution process to a Hadoop distributed storage location on a cloud computing platform. The variable z denotes the virtual node that is presently operational at the given time u. The R_l denotes the initialization of makespan for z virtual computing nodes. The transfer of e bytes of data from the cloud computing platform to the virtual computing worker's memory is supposed to have R_w^e. The function $R_w^e(z)$ represents the I/O makespan from the viewpoint of different virtual computing workers.

3.2 Makespan model of HMR framework

This study implies the utilization of the HMR framework to assess the operation i. The virtual computing nodes for Map and Reduce have been initialized using a cloud computing configuration. The virtual computing node count, denoted as z , is initialized. The dataset i is partitioned into g' segments. Each segment has a size of $g^{"} * e$ bytes. The Map virtual nodes for computing retrieve the input array of data from another spot container location and store it in a memory cache represented by the symbol E. The expression for the makespan method for production of varying values of z is as follows.

$R^f(g') = \left(R_L + (R_w^e * e * g')\right)$	(2)

u is an application computation used for memory-caching to exchange information to get the resultant outcome operation to store the remote spot container location through virtual computation nodes. The information for the dataset is executed using g' for virtual map content expressed through the following equation.

$W(g') = (w_e * g'' * g')$	(3)

Upon completion of the Map execution, the intermediary outcome information is stored in a remote spot container location. Due to variations in makespan across virtual computing environments and the inherent challenges of performing synchronization, achieving optimal performance can be a complex task. The initialization of Reduce virtual computing nodes occurs after the completion of Map task execution. The Reduce virtual computing node executes a reduction operation on intermediary data that is stored remotely. The resulting array U is obtained as the output of the reduce jobs. The equation below expresses the makespan time induced by the Reduce virtual computing node for performing reducing operations.

$\frac{g}{g''} - 1$	(4)

The HMR algorithm operates sequentially, resulting in a significant initial makespan. This is necessary to segment information and transmit it through a remote spot container ($R^F(g')$) before executing the algorithm ($W(g')$). The variable g'_r denotes the duration of the session where the computation of segmentation and application occurs, while considering the transmission makespan, this results in equal values for the utilization function W and the processing demand function QD (i.e., $W(g') = R^F(g')$). The makespan's length is represented as a set $[1 \dots g'_r \dots \widehat{g'}]$. The initial makespan is expressed using the equation $g' < g'_r$.

$W_U^R(g') = \left(\frac{g}{g''} + 1\right) R^F(g'), \forall g' < g'_r$	(5)

The computation of makespan time $W_U^R(g')$ can be performed in the following manner.

$W_U^R(g') \cong \left(\frac{g.R_L}{g''}\right) + (g * r_w^e * e * g'), \forall g' < g'_r$	(6)

The expression for the makespan time, taking into account for the time period when $\forall g' > g'_r$, is as follows.

$W_U^R(g') = \left(z * R^F(g')\right) + (g) * w_e, \forall g' > g'_r$	(7)

The makespan time $W_U^R(g')$ is approximated as follows:

$W_U^R(g') \cong (r_w^e * e * g'') + (g * w_e * r_w^e * g'), \forall g' < g'_r$	(8)

3.3 Hybrid HMR model

The earlier section's HMR model employs a serial execution approach. The Map computing node is responsible for the management and utilization of the virtual

computing slots allocated in the HMR. Upon completion of the Map operation, the Reduce computing node proceeds to utilize the available open slots. Consequently, there are several locations that remain unutilized. The increase in algorithm runtime leads to a higher makespan time. The allocation of Map and Reduce slots in the Hybrid HMR proposal is optimized to achieve maximum slot utilization. The Map virtual computing node retrieves the dataset information in segments from a remote spot container location and store them in its virtual computing memory cache. The HHMR makespan divides the data collected from the dataset into parallel chunks g'', allowing for execution on a multicore system. The makespan triggered by this procedure across z workers is expressed as follows.

$R^f(g'', z) = \left(R_L + (R_w^e(z) * g'' * e)\right)$	(9)

The parallelization of each element of g' is accomplished by utilizing a multi-core environment that employs virtual computing workers. In order to minimize storage and makespan overhead, the virtual computing worker's in-memory cache is cleared of redundant and previously computed data. As the makespan increases, it can be deduced that the availability of in-memory caching from the Map virtual computing nodes decreases. The decrease in usage of in-memory caching is in direct correlation with the quantity of computational operations utilized. The equation for calculating makespan takes into account the parallelized chunk g', utilized in the execution of applications u by the Map virtual computing node. The equation is as follows:

$W(g'') = (w_e * g'')$	(10)

The Map virtual computing nodes store the processed output in remote spot containers. The Reduce virtual computing node utilizes these outcomes to perform the reduce operation. The Reduce virtual computing node is represented by the following equation for makespan time.

$\left(\frac{g}{g''} - 1\right)$	(11)

The makespan here is introduced by the notation $W_U^R(g'')$, whereas the input dataset information vector is shown using the notation $D[a]$ and the algorithm consists of the vector shown as $U[b]$. However $\left[1,2, \ldots g_r'' \ldots \widehat{g''}\right]$ shows the makespan time by considering $g'' > g_r''$ shown as below:

$W_U^R(g'') = \left(\frac{g}{g'' * z} + 1\right) R^F(g'', z), \forall g'' < g_r''$	(12)

During the time-interval $g'' > g_r''$ the makespan time is expressed as shown below:

$W_U^R(g'') = \left(z * R^F(g'', z)\right) + \left(\frac{g}{z * w_e}\right), g'' > g_r''$	(13)

By utilizing the z virtual computing nodes is efficient via the optimization function denoted as :

$R^F(g_r''(z), z) = W_U^R(g_r''(z))$	(14)

The equation $R^F(g_r'', z) = W(g'')$ is used to represent a relationship between the time period e_p'' and the array $[1 \dots \widehat{g''}]$, where g_r'' is the time period. The proposed parallelized model could potentially enhance makespan performance by utilizing an enhanced slot scheduling technique for virtual computing workers. The utilization of in-memory cache is effective in achieving higher makespan speed, as demonstrated above.

3.4 Parallel optimal HMR

In this section, the HMR framework has been optimized for I/O to enhance parallel I/O performance and minimize disk search times. Furthermore, the HMR MergeSort algorithm has been optimized, concurrent input/output (I/O) activities within Heterogeneous Memory Management (HMR) may lead to considerable expenses to disk search. The reason for jobs being executed in distinct virtual machines without any interaction is due to their isolated nature. The I/O performance is decreased as a consequence. The study implemented a technique that involves sequential input/output (I/O) and layered central processing unit (CPU) execution and disk I/O in order to enhance I/O performance. The proposed I/O scheduler utilizes the ReadWorker (RW) and CleanWorker (CW) components to manage read and write operations, respectively. RW and CW have request capacity pools, which are also called multicapacity or multi-buffer capacities. Each pool is linked to 76 I/O requests. The read/write operations are reordered by RW and CW based on the selectivity of each capacity pool.

The proposed model categorizes the input/output (I/O) operation into two distinct types: active and passive I/O. To ensure fault tolerance, an active input/output (I/O) mechanism writes both the final output and intermediate output from Map tasks to the Hadoop distributed file system. The system is capable of reading input from the Hadoop distributed file system. Active I/O is a more selective option when compared to passive I/O, performing active I/O operations as quickly is essential for completing tasks efficiently. Passive I/O is a term used to describe the process of initiating I/O when intermediate data cannot be accommodated in the buffer and needs to be temporarily sent to the disk. Passive I/O is characterized by low clean selectivity and high read selectivity. This is achieved through the use of buffers that possess high allocation selectivity.

Upon submission of a write request to the I/O scheduler, as detailed in Algorithm 3, the scheduler will allocate a selectivity to the request based on its class P (Line 3 to 11). Subsequently, a $pool$ will be designated and appended to the pool list in $CleanWorker$. The process by which $CleanWorker$ performs data purification across multiple pools is shown in lines 14 through 22. Requests with high selectivity are prioritized for fulfilment. CleanWorker utilizes a round-robin approach to write one block per instance for requests with the same selectivity by polling their pools. This process is referred to as Interleaved-I/O in this study. ReadWorker performs the same function as CleanWorker in handling a read request.

Step 1	start
Step 2	Determine the required level of selection to fulfil the task requirement.
Step 3	If ($\mathcal{P} = \beta$) then
Step 4	$Selection <- B$
Step 5	else if (P=p)
Step 6	$Selection <- K$
Step 7	else if (P=no of turns)
Step 8	$Selection <- L$
Step 9	else if (P= 0)
Step 10	$Selection <- M$
Step 11	end if
Step 12	$Pool <- initializePool(selection)$
Step 13	Include a new pool in the cleaner's pool inventory.
Step 14	Data cleaning through the $buffer\_pool$
Step 15	When the $poolist$ is not null
Step 16	$T <- Peak$ selection of pools
Step 17	$Total\_Pool \in poollist$ do
Step 18	If ($t = pool\_selection$) then
Step 19	$Clean(pool[0])$
Step 20	end if
Step 21	end for
Step 22	end while
Step 23	end

The operation utilized in this study is commonly known as "in-memory merge-sort" as it is predominantly employed for processing data that has been stored in the memory. In the context of HMR, the sorting process is commonly known as an external sort. The sorting algorithm is applied to the data that is stored on drives, whereas the execution of CPU-bound sort operations and disk I/O incurs one after the other as a result. The central processing unit (CPU) is required to enter a blocking state and remain idle until the buffer is completely filled with the sorted data. Non-blocking, I/O can be achieved through various capabilities, the capacity pools of $CleanWorker$ are expanded by incorporating a cache of data elements. Once the cache components have been written back to the disk, $CacheList$ will immediately retrieve them.

4 Result Analysis`

This section presents an empirical evaluation of the HHMR's performance related to the previous model across various applications. HMR is the recommended MR framework for running data-intensive and scientific applications on a cloud computing infrastructure. The Hadoop cluster consists of a single master node and four slave nodes; it has been deployed on the Microsoft Azure public cloud platform. This study utilizes Hadoop 2.7, which includes a cloud spot container capacity of 120 GB, 7 GB of RAM, and 4 cores for each worker, encompassing both the master and slave. Identical configurations are utilized for executing various applications on both HMR and HHMR. The evaluation of HHMR versus HMR performance in memory is based on an assessment that considers applications with a high CPU to I/O workload. To assess the resilience of a parallel computing framework, it is essential to utilize the three case studies. The study examined various applications, including stream data analysis, non-stream data analysis, and gene sequence analysis. The process of analyzing gene sequences is currently resource-intensive in terms of memory, CPU, and I/O due to the vast amount of genetic data that is accessible. HMR models exhibit careful handling of tasks that necessitate elevated CPU utilization. Therefore, it is essential to

assess its efficiency when subjected to memory and I/O intensive workloads. The evaluation of the HHMR model's performance is crucial in the context of diverse genomic data sizes, which may comprise short and long reads. The HHMR model is designed to tackle the challenges related to Memory and Input/Output, while simultaneously maintaining the CPU capabilities of HMR. This section presents an in-depth examination of the genetic information that is employed. Both stream and non-stream applications are considered for analysis. The analysis of data for non-stream e-commerce data is considered. CPU-intensive applications frequently require minimal input/output (I/O), here the Twitter dataset is considered for non-stream applications. The performance of these applications relies heavily on the CPU and I/O operations. It is vital to evaluate the performance of HHMR and HMR in these particular applications. The following section provides a detailed analysis of the dataset.

4.1 Performance Evaluation for gene sequenced data comparing HHMR vs HMR

This section evaluates the efficacy of gene sequence analysis through the application of the HHMR and HMR frameworks. Sequence alignment methods are essential for analyzing applications related to cancer research, genetic illness detection, and reproductive health. This study utilized query sequences from the genomic databases of baker's yeast, the influenza virus, and the Homo sapiens chromosome (NC_000015.10). The query details and geneomic reference utilised for the experimental analysis are shown in table I and the result is graphically plotted in fig 2. The outcome of the experiment shows that

The current investigation concerns the calculation of the theoretical optimization of HHMR makespan time, as outlined in Equation (14). The acquired numerical result is subsequently compared to the empirical value, taking into account different sizes of genomic data. The output obtained is presented in Figure 3. The calculation of makespan, as determined through both empirical data and mathematical analysis, displays a slight variance. It is common for there to be a high degree of agreement between the makespan time derived from experimental and computational approaches. The results of the experiment suggest that incorporating gene sequence analysis into the proposed HHMR framework generates better results when compared to the HMR framework. The demonstration is achieved via the utilization of bioinformatics analysis. The mathematical model for job makespan in HHMR is validated by means of correlation measurements, which conform to its accuracy and validity.

Table 1 Gene sequence considered for experiment analysis

Reference genome sequence	**Sequence length**	**Query genome sequence**	**Sequence length**
NC_000015.10	101991189 bp	NC 026141.2	4988 base pair
NC_000015.10	101991189 bp	NC 010955.1	10207 base pair
Saccharomyces cerevisiae S288c chromosome XII	1001933 bp	Saccharomyces cerevisiae S288c chromosome V_BK006939.2	576874 base pair
Saccharomyces cerevisiae S288c chromosome XII	1001933 bp	Saccharomyces cerevisiae S288c chromosome XVI_BK006949.2	948066 base pair

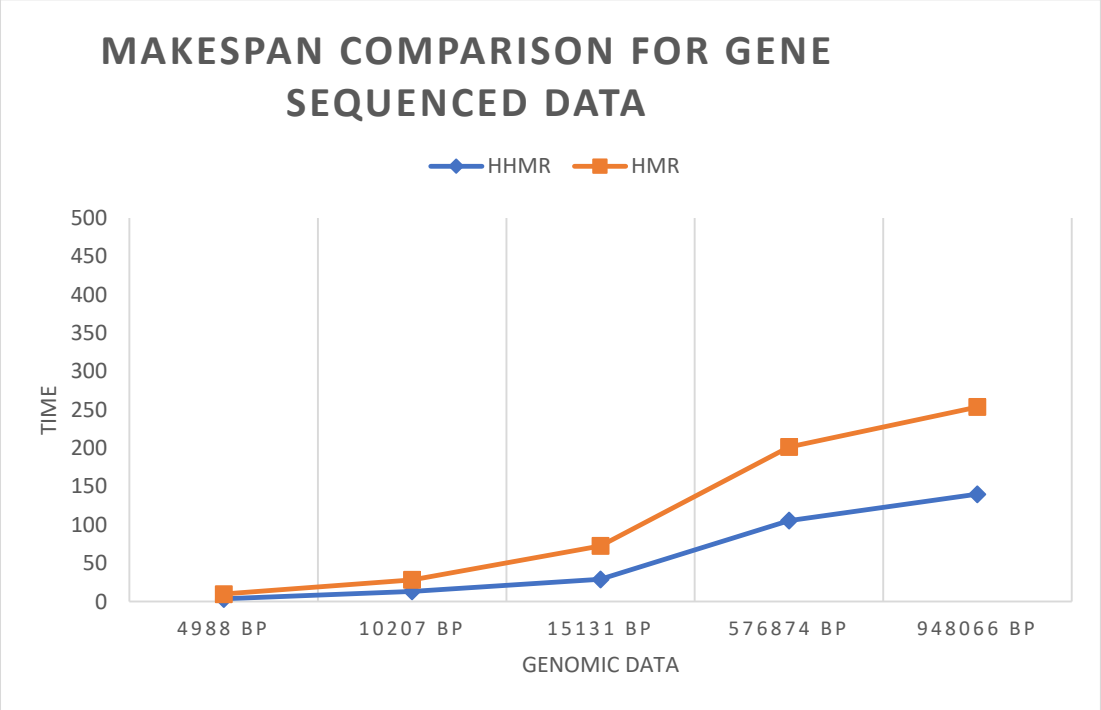

Figure 1 Makespan comparison of gene sequenced data of existing system and proposed system

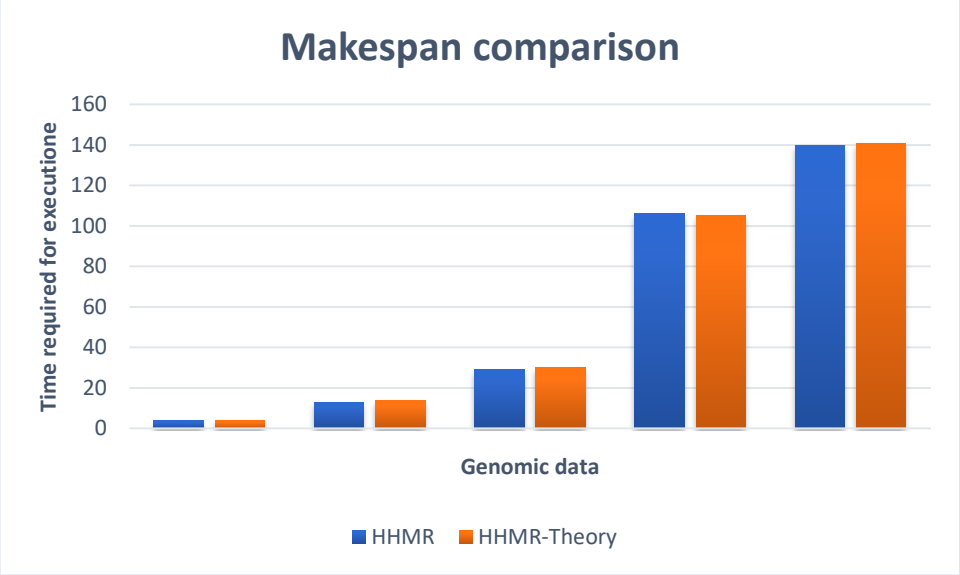

Figure 2 Theoretical and actual execution times on varied datasets and the magnitude of the genomic data using the HHMR framework are correlated.

4.2 Performance Evaluation of e-commerce data comparing HHMR vs HMR

This section discusses the findings of a performance assessment conducted on HHMR and HMR for e-commerce data. The purpose of the assessment was to compare the performance of HHMR and HMR in handling e-commerce data. The investigation was carried out using established methodologies and statistical techniques. The results of the assessment are depicted in this section. The performance of the HDPC and HMR framework is assessed in this section with respect to its application in the analysis of e-commerce data. The program "Wordcount" (also known as "Text computation/mining") is utilized for the purpose of conducting experimental analysis. The experiment instance illustrated in Table II is considered in this study. Figure 4 presents a visual representation of the outcomes obtained from the wordcount analysis conducted on the Amazon review dataset.

The present study estimates and compares the experimental value of HHMR's makespan time with the theoretically optimized value. The analysis takes into account various Amazon review data sizes. Figure 5 depicts the resulting outcome. The experimental results and calculations related to makespan computation exhibit a minor deviation. Frequently, there exists a significant concurrence between the makespan times obtained from experiments and those calculated. The experimental findings indicate that utilizing Wordcount analysis, a technique in text computing/mining, on the proposed HHMR framework produces superior outcomes in comparison to utilizing the HMR framework. The level of precision and accuracy of the makespan model utilized in the HHMR is demonstrated through correlation measurements.

Table 2 E-commerce data considered for experiment analysis

Experiment ID	**Dataset**	**Number of reviews**	**Number of product**
1	Sports and outdoors	3,268,695	532,197
2	Cellphones and accessories	3,447,249	346,793
3	Clothing shoes and jewelry	5,748,920	1,503,384

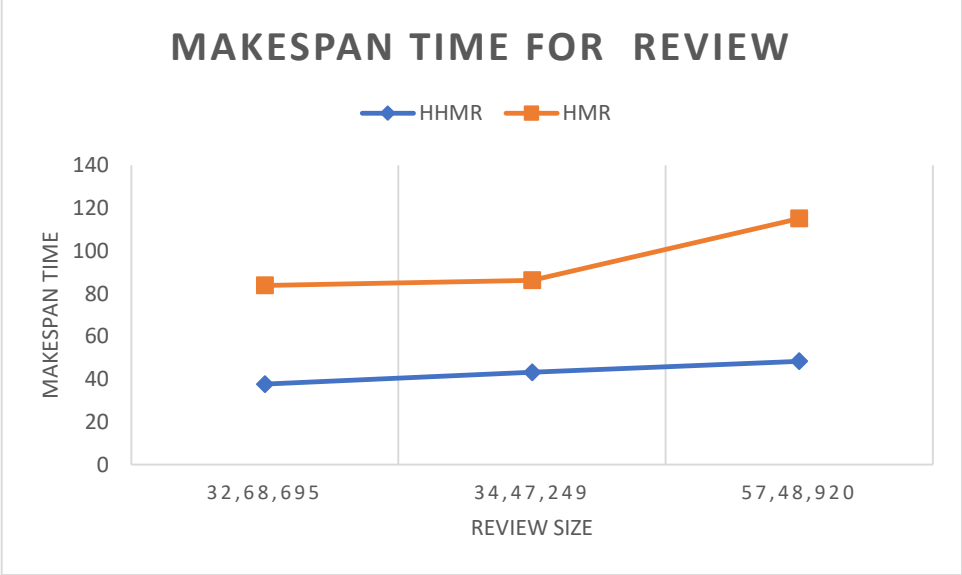

Figure 3 total makespan time required to perform E-commerce analysis on datasets of varying sizes and review sizes using both the HHMR and HMR frameworks.

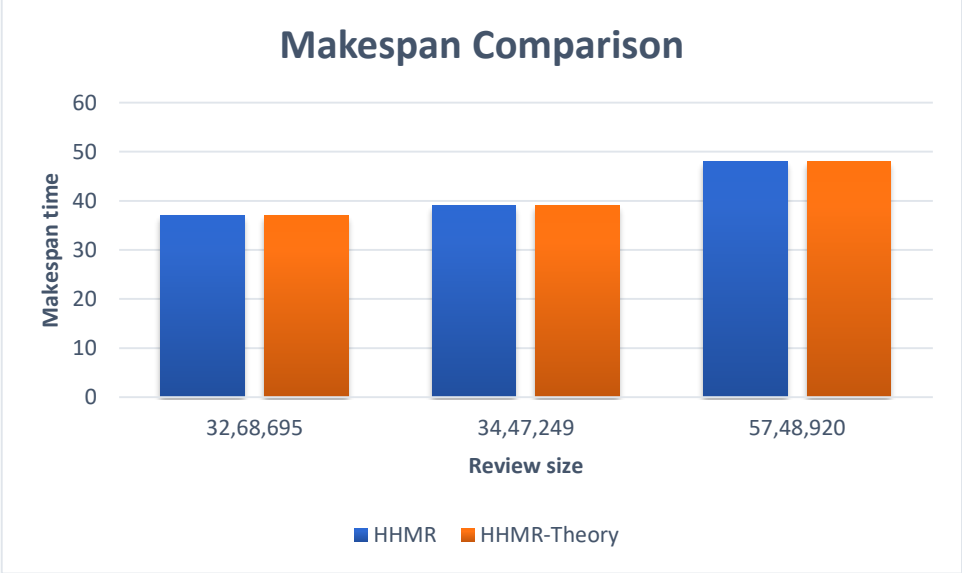

Figure 4 Theoretical and actual execution times on varied datasets and the magnitude of the review done using the HHMR framework are correlated.

4.3 Performance assessment of HHMR vs HMR using stream data analysis

This section evaluates the performance of the stream application on both HHMR and HMR frameworks. The hot word detection application is being taken into consideration. Python is the programming language utilized for application development. In order to evaluate the effectiveness of a system or process, it is necessary to conduct a performance assessment. The "Movietweetings" dataset has been taken into consideration. The present task is stored in a container of Azure blob storage and employs stream tweet sizes of 25K, 50K, and 100K for experimental evaluation. The hot word detection test is performed on the stored data of both HMR and HHMR. The outcomes of this test are recorded as illustrated in Figure 6.

The makespan time of HHMR is theoretically optimized and compared to experimental values obtained from varying tweet data sizes. Figure 7 presents the resulting output. The computation of makespan based on experimental results and mathematical analysis exhibits a minor deviation. The experimental and calculated makespan times frequently exhibit significant concurrence. The experimental findings indicate that the HHMR framework's stream data analysis outperforms the HMR framework in terms of performance. The accuracy and validity of the makespan model utilized in the HHMR is demonstrated through correlation measurements.

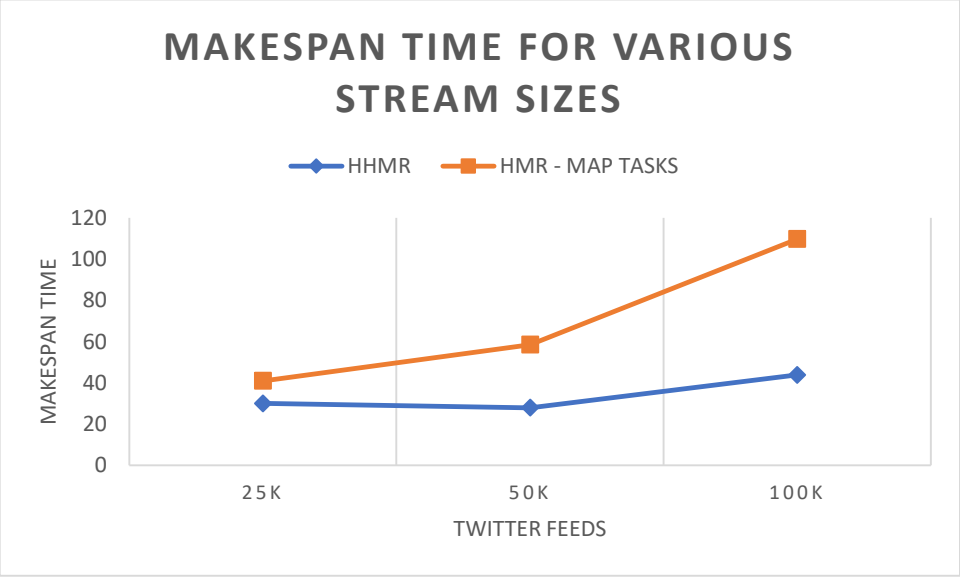

Figure 5 Makespan time for various stream sizes

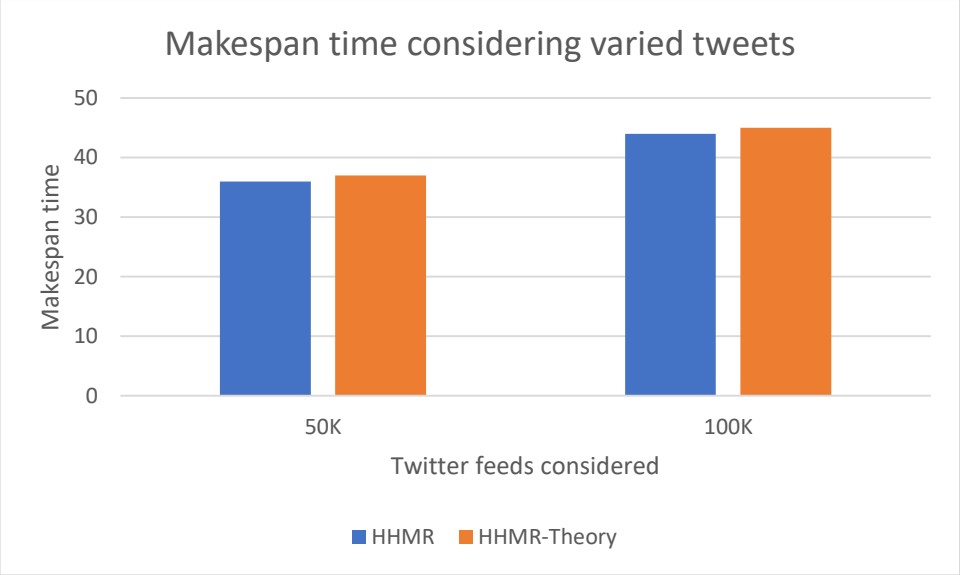

Figure 6 Makespan time considering varied tweets

4.4 Comparison result

This section discusses the advantages of HHMR over HMR and other Hadoop-based MR models for a variety of applications, including hot word detection, non-stream applications for analysing e-commerce reviews, and non-stream applications for genomic sequence analysis. The findings presented here demonstrate that the suggested makespan model, which takes into account memory optimization and a parallel execution technique to eliminate empty/null slots, results in the HHMR framework minimizing the overall makespan. In table 3 the comparative analysis for gene sequence data is carried out for 4988 bp, 10207 bp, 15131 bp, 576846 bp, 948066 bp the improvisation is 124.896%, 82.2939%, 98.4811%, 65.1441%, 66.8069%. In table 4 the comparative analysis is carried out for review data for reviews 32,68,695, 34,47,249, 57,48,920 the improvisation is 75.9679%, 68.4801%, 81.6689%. In table 5 the comparative analysis is carried out for twitter feeds data for 25k,50k and 100k the improvisation is 101.767%, 70.6656%, 85.7474%.

Table 3 Comparative Analysis for gene sequenced data

Gene sequenced data	**ES**	**PS**	**Improvisation**
4988 bp	9.72	2.2469518	124.896%
10207 bp	28.45	11.86267896	82.2939%
15131 bp	72.56	24.67899	98.4811%
576846 bp	201.66	102.567	65.1441%
948066 bp	253.75	126.675	66.8069%

Table 4 Comparative Analysis for review data

Review	ES	PS	Improvisation
32,68,695	83.857	37.689	75.9679%
34,47,249	86.215	42.234	68.4801%
57,48,920	115.085	48.348	81.6689%

Table 5 Comparative Analysis for twitter feeds data

Twitter feeds data	ES	PS	Improvisation
25k	40.998	13.3459	101.767%
50k	58.57	27.987	70.6656%
100k	109.787	43.897	85.7474%

Conclusion:

The objective of this study was to examine the constraints of the Hadoop MapReduce framework. This section discusses the importance of memory and input/output (I/O) requirements in the development of an efficient HMR (Hybrid Mobile Application) framework. The aim of this project was to reduce the time taken for makespan and enhance the efficiency of resource utilization in a cloud computing environment. This article presents a model for achieving a Hybrid Hadoop MapReduce makespan. The HHMR makespan model has been developed with the objective of minimizing the unutilized or null slot of a virtual compute node. The proposed approach involves the development of an optimization model to manage the memory of local workers. Our developed paradigm enables efficient parallel processing while minimizing memory overhead. The segmented data is divided into smaller units to enable concurrent processing by virtual machines utilized in MapReduce computing. Efficient utilization of resources (slots) is made possible by the use of parallel execution technique and minimal memory usage, which reduces memory I/O overhead. Microsoft Azure HDInsight, a cloud-based platform for the public, is currently in the testing phase for its usage. The performance of the HHMR framework has undergone testing in comparison to the latest parallel computing technology. The validation process encompassed various applications, including bioinformatics, text mining, stream, and non-stream applications.

REFERENCES

[1] 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, p. 260, 2008.

[2] 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.

[3] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica,“Spark: Cluster Computing with Working Sets,” in Proceedings of the

2nd USENIX Conference on Hot topics in Cloud Computing, (Boston,MA), June 2010.

[4] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," ACM Commun., vol. 51, no. 1, pp. 107–113, Jan. 2008.

[5] "Apache Hadoop." [Online]. Available: http://hadoop.apache.org/. [Accessed: 21-july-2018].

[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. Shi et al., "Mammoth: Gearing Hadoop Towards Memory-Intensive MapReduce Applications," in IEEE Transactions on Parallel and Distributed Systems, vol. 26, no. 8, pp. 2300-2315, Aug. 1 2015.

[8] J. Zhu, J. Li, E. Hardesty, H. Jiang and K. C. Li, "GPU-in-Hadoop: Enabling MapReduce across distributed heterogeneous platforms," Computer and Information Science (ICIS), 2014 IEEE/ACIS 13th International Conference on, Taiyuan, pp. 321-326, 2014.

[9] D. Dahiphale et al., "An Advanced MapReduce: Cloud MapReduce, Enhancements and Applications," in IEEE Transactions on Network and Service Management, vol. 11, no. 1, pp. 101-115, March 2014.

[10] E. Deelman, G. Singh, M. Livny, B. Berriman and J. Good, "The cost of doing science on the cloud: The Montage example," 2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis, Austin, TX, pp. 1-12, 2008.

[11] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz, "See spot run: using spot instances for mapreduce workflows," in Proc. 2010 USENIX Conference on Hot Topics in Cloud Computing, ser. HotCloud'10. USENIX Association, pp. 7–7, 2010.

[12] 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.

[13] 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.

[14] Liu Y, Maskell DL, Schmidt B. CUDASW + + : optimizing Smith–Waterman sequence database searches for CUDA-enabled graphics processing units. BMC Res. Notes. 2009; 2:73. [PubMed: 19416548].

[15] Nawrocki EP, Kolbe DL, Eddy SR. Infernal 1.0: inference of RNA alignments. Bioinformatics. 2009; 25:1335–1337. [PubM ed: 19307242] One of the first GPU-based RNA sequence aligners.

[16] L. D. Stein, "The case for cloud computing in genome informatics," Genome Biology, vol. 11, no. 5, article 207, 2010.

[17] R. S. Thakur, R. Bandopadhyay, B. Chaudhary, and S. Chatterjee, "Now and next-generation sequencing techniques: future of sequence analysis

using cloud computing," Front Genetics, vol. 3, article 280, 2012.

[18] J. Chen, F. Qian, W. Yan, and B. Shen, "Translational biomedical informatics in the cloud: present and future," BioMed Research International, vol. 2013, Article ID 658925, 8 pages, 2013.

[19] T. Nguyen, W. Shi, and D. Ruden, "CloudAligner: a fast and fullfeatured MapReduce based tool for sequence mapping," BMC Research Notes, vol. 4, article 171, 2011.

[20] John Gantz and David Reinsel. THE DIGITAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East. Tech. rep. Internet Data Center (IDC), 2012. URL: http://www.emc.com/collateral/analyst-reports/idc-the-digitaluniverse-in-2020.pdf.

[21] M. Sahli, E. Mansour, T. Alturkestani and P. Kalnis, "Automatic tuning of bag-of-tasks applications," 2015 IEEE 31st International Conference on Data Engineering, Seoul, pp. 843-854, 2015

[22] J. Arram, T. Kaplan, W. Luk and P. Jiang, "Leveraging FPGAs for Accelerating Short Read Alignment," in IEEE/ACM Transactions on Computational Biology and Bioinformatics, vol. 14, no. 3, pp. 668-677, May-June 1 2017.

[23] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz, "See spot run: using spot instances for mapreduce workflows," in Proc. 2010 USENIX Conference on Hot Topics in Cloud Computing, ser. HotCloud'10. USENIX Association, 2010, pp. 7–7.

[24] Babak Alipanahi, Andrew Delong, Matthew T Weirauch and Brendan J Frey1, "Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning," in nature biotechnology, vol. 33, no 8, 2015.

[25] 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. [54] K. Mahadik, S. Chaterji, B. Zhou, M. Kulkarni and S. Bagchi, "Orion: Scaling Genomic Sequence Matching with Fine-Grained Parallelization," SC14: International Conference for High Performance Computing, Networking, Storage and Analysis, New Orleans, LA, 2014, pp. 449-460.

[26] 6]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. [107]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.

[27] 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.

[28] 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.

[29] 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.

[30] 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