IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 29 September 2024, accepted 12 October 2024, date of publication 29 October 2024, date of current version 8 November 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3487936

RESEARCH ARTICLE

GSelf-MapReduce: A Method for Enhancing Mapreduce Performance in Distributed Heterogeneous Data Centers

EMIN SESEN, SERDAR KIRISOGLU, AND RESUL KARA
Department of Computer Engineering, Düzce University, 81620 Düzce, Türkiye
Corresponding author: Emin Sesen (eminsesen@duzce.edu.tr)

ABSTRACT Big data are often stored close to the locations where they are generated, owing to the cost of data transfer. These stored data are moved to a single location for processing or processed at that location. In the literature, it is possible to find different methods for processing data in distributed data centers. In this study, we present a new method for data processing called GSelf-MapReduce. In the proposed method, shuffling is performed among heterogeneous data center (DC) that complete the data-processing process. To calculate the data processing cost of the reduce function of the DCs, a polynomial regression model was created using the data obtained in the test environment, and the coefficients obtained from this model were used in the decision process. The key/value pairs to be shuffled are distributed according to the cost of the DCs, considering their location. In addition, not all DCs are waited to finish their job for shuffling. DCs that complete their job perform shuffling among themselves. Thus, the keys are deduplicated between these DCs. The shuffling volume in the last phase and the total job completion time are reduced. The performance of the proposed method was compared with that of four different distributed data processing methods in the literature. As a result, this work generates 15% less shuffled data than the closest work.

INDEX TERMS Big data, data reduction, geo-distributed mapreduce, heterogenous data centers.

I. INTRODUCTION

Big data has emerged as a new research field in academia and industry [1]. High-volume, rapidly increasing data per unit time are called big data [2]. These include social network interactions, electronic mail data, videos, sensor data, images, search engine queries, health records, data generated from customers, and access records produced by large companies [3], [4], [5]. Because the generated data do not have a specific structure, they are not suitable for storage in a structured database [6]. To give an example of this data, on a daily average in 2022; 720 thousand hours of content was uploaded to Youtube [7], 8.5 billion searches were made on Google [8], 95 million photos were uploaded on Instagram [9], 5.75 billion likes and 4.3 billion messages were sent on Facebook [10]. The data generated by 2025 were estimated to be 180 ZB [7]. It is not possible to store and analyze the data in a structured database.

Hadoop is a scalable and reliable software project that enables large datasets to be stored and computed on distributed nodes. It can consist of one or more servers, each of which performs local data storage and computational computations [11]. Hadoop is preferred for large data storage because of its scalability, high availability, fault tolerance, low cost, and efficiency [12].

Let us imagine that a heavily used web application server can serve different geographical locations. This server generates access information (IP, HTTP status code, browser, file path), access errors, and data-load log data. Because the size of this log data is large, it can be stored in a Hadoop cluster near an application server. It is very difficult to move and process these data in a single data center [13], [14]. This results in the intensive use of network resources around aggregated datasets and increases transmission time. Therefore, instead

The associate editor coordinating the review of this manuscript and approving it for publication was Rahim Rahmani.

© 2024 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

of centralized data aggregation, an ideal method is to aggregate geographically distributed data from multiple sources in multiple-edge DCs and process them in parallel in the DC where they are stored [15]. However, Hadoop does not provide any infrastructure to perform MapReduce processing of unstructured data stored on multiple Hadoop clusters [16], which we will discuss later in this study.

Organizations with multiple DCs may have multiple data processing nodes. Nodes in one data center can be very different from nodes in another data center in terms of CPU frequency, number of cores, cache size, memory size, network bandwidth, and storage capacity. These factors are important for the performance of computations on the datasets. If the dataset to be processed is small and the time to transfer the data is insignificant, the datasets can be transferred to a single data center for processing [16].

Big data also needs to be fault tolerant. In addition, owing to its size, it cannot be stored in a single node. Therefore, the need for distributed storage and analysis has emerged [17].

The MapReduce is a programming model that allows large datasets to be processed simultaneously on different machines. It is offered by Hadoop and was introduced by Google as a method for processing petabyte-sized data on large clusters of low-equipped machines [18]. MapReduce consists of two phases: Map and Reduce. In MapReduce applications, key/value pairs generated from stored blocks are used as inputs in the mapping process. The key/value pair is generated as the output of the map process. The key/value pair generated as the output of the mapping process constitutes the input data for the reduction process. After completion of the reduction operations, the results of the application are obtained using MapReduce.

To use the MapReduce model, the code needed for this purpose is first created. The code generated for this job is automatically distributed to the computing nodes. With the MapReduce model, applications in many areas such as large-scale graphics processing, data mining, text processing, and machine learning can be realized [19].

Big data can be located in different locations. The transportation of data over a network can create a bottleneck in the network and increase the processing time of the data [20]. Therefore, the data must be processed at the cluster where it is stored. The data processing workload is also distributed across the clusters.

In addition to MapReduce for processing big data, there is a data processing engine called Spark, which was developed after MapReduce. MapReduce is fast in batch data processing. Spark is faster in streams. Because while MapReduce processes data on disk, Spark processes data in in-memory [21]. If the data size is larger than Spark's memory, Spark processes the data more slowly [22]. Also, when the memory size is increased, data processing becomes more expensive. For these reasons, we preferred MapReduce.

In this paper, we propose a method for processing data in distributed data centers using MapReduce. In the stages of the proposed model; MapReduce job is started in each DC. It is aimed to perform shuffling and reduce processes between DCs that complete the MapReduce job. The job progress of the ongoing DCs is checked. If there is a DC whose job completion status is greater than a certain threshold value, this DC is included in the shuffling and reduce process. The data processing costs of DCs are determined by the parameters of the reduce function that DCs process per unit time. Keys are grouped according to the DCs to which they belong. The grouped keys are distributed according to the costs. It is ensured that the data processing costs of all DCs are approximately the same. In this study, we propose a methodology to answer the following questions: (i)How to reduce the shuffling volume and data processing time of distributed DCs? (ii)Which DCs perform shuffling and reduce processes among themselves? (iii)How to distribute keys fairly across heretogenous DCs?

The main contributions of this work are summarised as follows:

- DCs that completed the data processing process perform shuffling among themselves and deduplicate the same keys.
- Key allocations are made according to the reduce function processing performance of DCs, so that key/value distributions are balanced in shuffling.
- Since not all DCs are waited to complete their work for shuffling, total data processing time and shuffling traffic are reduced.
- An experimental environment was created to demonstrate the effect of this method, the results obtained were compared with other methods.

Section II presents previous work in this area, Section III presents the methodology, Section IV presents the experimental work, Section V presents the discussion and Section VI presents the conclusion.

II. RELATED STUDIES

Although Hadoop performs well for data processing using MapReduce on clusters of one or more nodes, it does not perform well on geographically distributed data clusters. In the literature, there are two main approaches for data processing on many such clusters: Geo-Hadoop and Hierarchical Hadoop [23].

Iordache et al. [24] designed an API for MapReduce jobs to utilise the resources of multiple different cloud providers simultaneously. Wang et al. [25] proposed a job scheduling framework that considers data location, job completion time and resource availability to save energy in heterogeneous clusters. Luo and Plale [16] proposed two algorithms based on Hierarchical approach for computationally resource-intensive jobs and jobs with large datasets.

In the literature, studies are presented to reduce data migration costs between distributed data centres. Pu et al. [14] reduce network utilisation and improve query time by taking into account the location of the input data of a job, the location of the output of the map function or the location of the clusters where the tasks will run. Quoc et al. [26], crawled web sites on the internet with distributed clusters. With their proposed

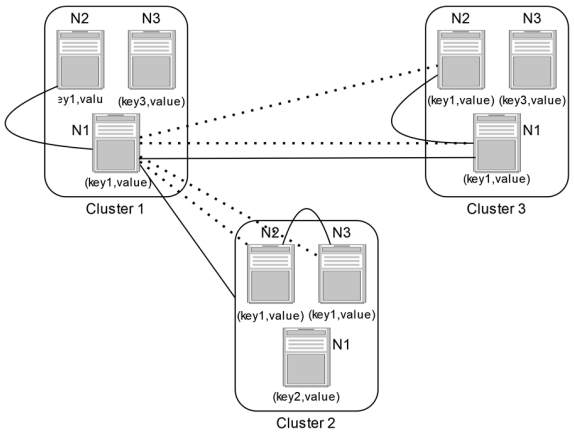


FIGURE 1. Cross-MapReduce and Geo-Hadoop approaches. (dotted lines show geo-hadoop and solid lines show cross-map-reduce approaches).

methods, they reduce the communication cost of crawl data between clusters. Kumar and Mohbey [27], To reduce the data traffic in shuffling, tasks are assigned among cluster nodes in a fair and efficient manner. They use search space partitioning strategy for task allocation. Chen et al. [28], stated that it is not efficient to route the data flow directly from clusters. Therefore, they presented a method that is performed by hopping from alternative clusters. Marzuni et al. [20], separated intra-cluster and inter-cluster traffic. After running the MapReduce programming model on distributed Hadoop clusters, they used GShuffling to distribute keys and values only to the clusters where they reside. They used a graph structure called Global Reduction Graph (GRG) to ensure that the key/value distributions between clusters are balanced. The intensive data transfer in the shuffling process is reduced.

Studies on monetary cost optimisation in distributed data processing are presented. Xiao et al. [29], aimed to jointly reduce the computing, storage and data migration costs between distributed data centres. Lyapunov optimisation algorithm is used for data movement and reducer selection. Gouasmi et al. [30], presented an algorithm that allows cloud providers to accept more jobs and increase profits. Chen et al. [31], presented an algorithm that improves the cost of data processing by utilising the workload, number of free slots, and bandwidth in the map shuffle and reduce phases to place tasks in geographically distributed IoT clusters. Xu et al. [32], reduced the network cost of a large number of data processing jobs using Lyapunov optimisation techniques. In addition, the location of input data is determined so that future query requests provide optimal results.

In the literature, there are studies on the completion of jobs running in distributed data centres in a shorter time. Xiao et al. [33], designed a cache in iterative MapReduce jobs that require the same piece of data to be processed multiple times to avoid heavy network and disc traffic caused by moving each input data between nodes. Jayalath et al. [34], presented the G-MR framework for processing data in distributed clusters. According to this framework, data can be moved between clusters according to Map and Reduce phases. Besides the completion of the work, the monetary cost is also optimised. Heintz et al. [35], perform task assignment based on the connection speed between nodes and the processing power of the nodes. Imai et al. [36], presented a layered architecture for distributed processing of IoT data. From the lowest level where the data is located to the upper level, the data is combined with reduce. Cavallo et al. [23], presented an algorithm that takes into account the computational capacity of the node and the capacity of the link. Application profiles are created by running small instances of jobs. Wang et al. [37], replaced the Hadoop file system with the gfarm file system, which provides fault tolerance by creating copies of MapReduce tasks on different clusters.

In the studies, data transfer between data centres over the internet is not efficient. Since the performance of the machines and the number of compute nodes in the data centres may be different, the size of the data stored in the distributed data centres and the completion time of the jobs have been ignored. In contrast to the data processing methods in distributed clusters in the literature, in this paper we show that clusters that have finished the data processing process can shuffle among themselves. For this, the progress of the clusters whose data processing process continues is taken into account. Because it is not known when an ongoing work in a cluster will end, clusters shuffle among themselves in progress situations greater than a certain threshold value. In addition, the heterogeneity of clusters is taken into account. The equation coefficients obtained by the model trained with polynomial regression from machine learning models are used for key distribution. Since not all clusters wait for data exchange, generation time and shuffling traffic are reduced.

III. GSELF-MAPREDUCE METHOD

Hadoop does not provide an interface for the MapReduce programming model in its distributed clusters. The MapReduce function can be executed in distributed clusters using different approaches.

Geo-Hadoop is an approach in which data are stored in different data centers and processed distributedly. Geo-Hadoop is categorized under two architectures: centralized and decentralized [38]. A centralized architecture, a manager node located in one of the clusters, manages the resources of the other cluster nodes. In a decentralized architecture, each cluster manages its own worker node. Each cluster can run Hadoop jobs independently. In addition, clusters can share computation and data resources with each other for geographically distributed jobs.

In Hierarchical Hadoop, each cluster completes a MapReduce task. The key/value outputs of the clusters are transferred to a single global reducer. The outputs are obtained by running the reduce function in global reducer. Although the MapReduce-GlobalReducer approach has a two-layer structure, it can be adapted to Hierarchical approaches with more layers, as needed [39].

In a study called Cross-MapReduce, these two methods were combined, and a method called GShuffling was presented. As shown in Figure 1, after the map function is completed in multiple nodes in a cluster in Geo-Hadoop, multiple key/value pairs with the same key are moved between clusters. In this study, after the MapReduce job is completed in each cluster, keys are exchanged between clusters and only between data centers where the key is available. Thus, it aims to prevent unnecessary network traffic in data centers that do not have a key/value pair [20].

In the proposed GSelf-MapReduce method, when the clusters complete data processing, they are allowed to shuffle among themselves. Therefore, it is not waited that all clusters will complete their jobs. For example, six clusters are shown in Fig. 2. Because the data in these clusters may contain data of different sizes and the computational capacities of the clusters may be different, the running times of the MapReduce function may also differ.

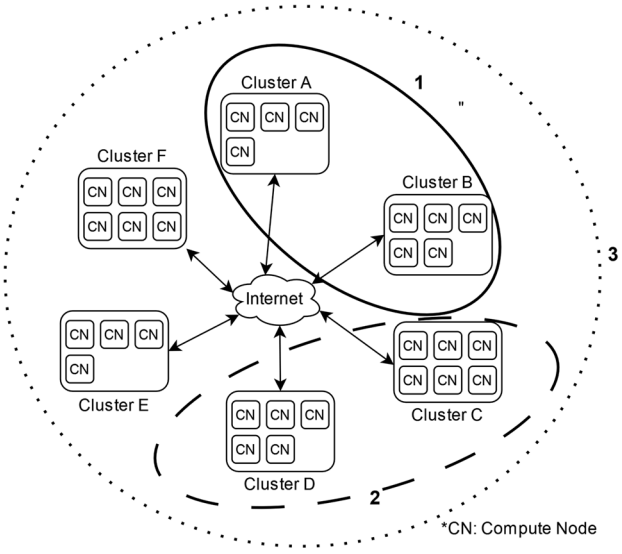


FIGURE 2. GSelf-MapReduce approach.

In case 1 in Figure 2, because the MapReduce job in Clusters A and B is completed before the others, they exchange data by shuffling among themselves. After shuffling, the reduce function is run on clusters A and B. Subsequently, after shuffling the clusters (C and D) in Case 2, and then all clusters in Case 3, the reduce function is run in the same way. To shuffle between clusters, the Cross-MapReduce was modified. The distribution of the keys/values occurs according to the computational capacity of the clusters.

For shuffling, keys are moved from the clusters that have completed their jobs to the cluster with the largest key size. Key distribution occurs. Subsequently, the shuffling process is initiated. After shuffling is completed, the reduce function is executed on the clusters. Thus, the data volume and production time in shuffling process are reduced.

A. GSELF-MAPREDUCE

In this study, a method is proposed to process data in distributed clusters using MapReduce. The proposed method consisted of two stages. First, the job was started in all clusters. Subsequently, the clusters that have completed their job shuffling among themselves according to the data volume they process per unit time.

In the physical environment used as a reference for the testbed of the proposed GSelf-MapReduce method, a structure consisting of a Data Center Gateway (DCG), Data Center Manager (DCM), and clusters was created. The clusters and DCG were located in different geographical locations. A cluster existed in each DC. The working steps of the proposed method are illustrated in Fig. 3. Accordingly;

Step 1: The client sends the MapReduce job to the DCG (1). The DCG also sends a job to the clusters defined in (2).

Step 2: Jobs are executed on clusters. Clusters that complete their jobs send notifications to DCM. (3).

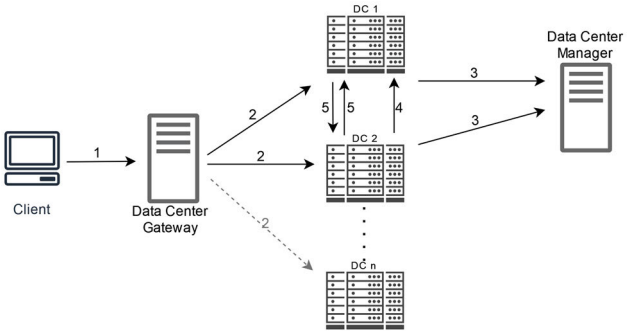


FIGURE 3. Steps of the proposed method.

Step 3: The cluster with the smallest key size sends the keys to the other cluster (4).

Step 4: Data processing cost of clusters is calculated by DCM.

Step 5: According to the costs of the clusters, the key distribution between clusters is performed on the cluster where the keys are collected. Subsequently, key distributions are sent to the clusters.

Step 6: The key/value pairs whose clusters are determined are sent to the cluster to which they belong. Therefore, they shuffled among them. (5).

Step 7: After the shuffling process is completed, the reduce function is executed on the clusters.

Inter-cluster shuffling and reduction processes (steps 3-7) continue until all clusters complete their jobs. Data processing is completed when all clusters simultaneously perform shuffling and reducing stages.

The flow diagram of these stages is given in Figure 4.

1) INTER-CLUSTER REDUCE

According to the flow diagram shown in Figure 4, the clusters that perform the key exchange are determined according to

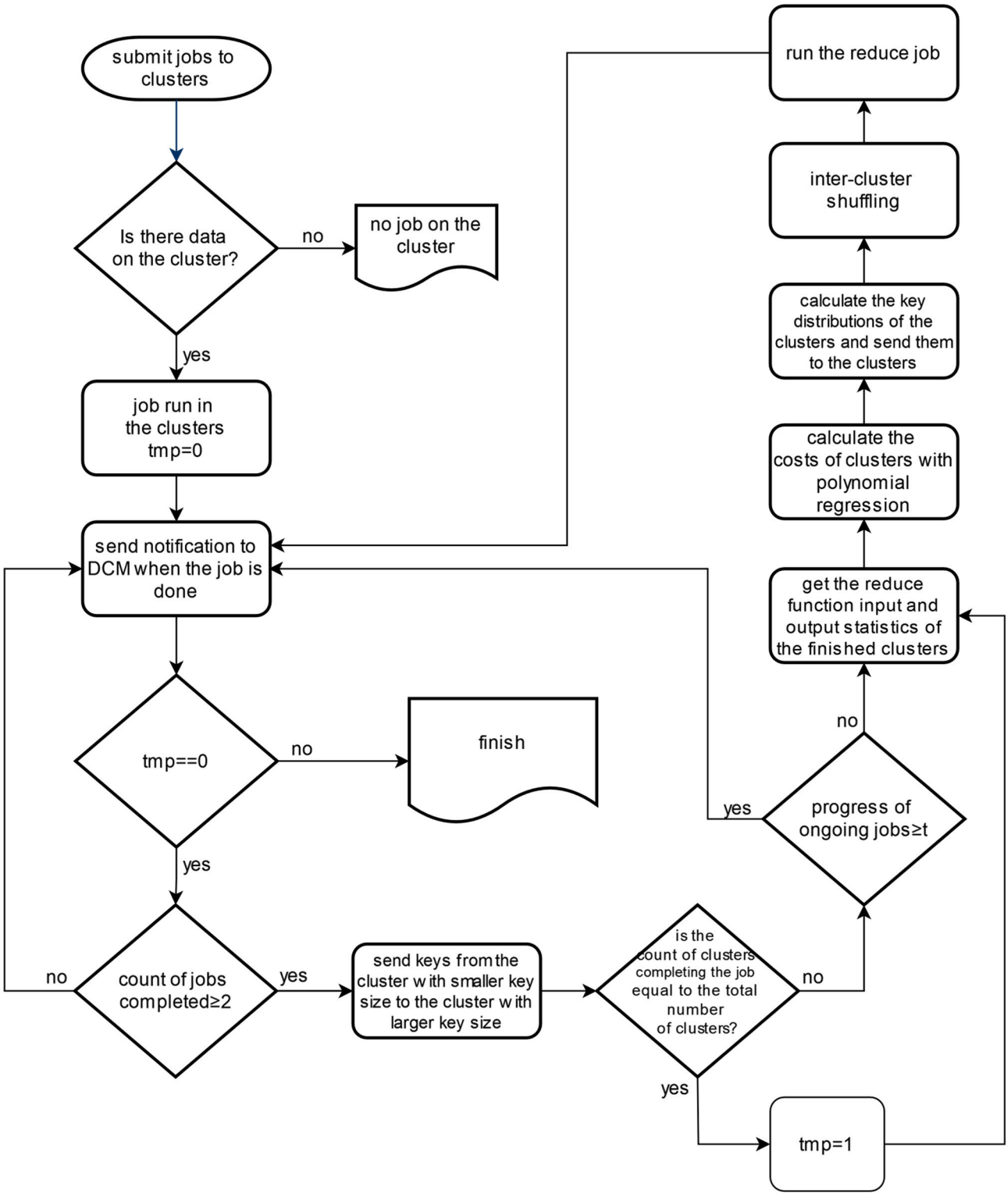


FIGURE 4. The flow diagram of GSelf-MapReduce.

their job progress, and the costs of the clusters are created by evaluating the performance of the clusters in running the reduce function. The cost created here was determined as the time taken for data processing in seconds. The parameters used at this stage are as follows:

I = *all clusters that have completed their job,*

TABLE 1. Variables in the equation.

Variable	Definition
Wi_i	ith cluster reduce input weight
Wo_i	ith cluster reduce output weight
RAM_i	ith cluster RAM size
CPU_i	ith cluster CPU core size
JT_i	ith cluster job type

$J = all\ clusters\ that\ continue\ their\ job,$
$DC_{cmp} = number\ of\ clusters\ that\ completed\ their\ job$
$DC_i = ith\ cluster\ that\ completed\ its\ job$
$P_i = progress\ of\ the\ ith\ cluster\ continuing\ its\ job,$
$Dk_i = data\ size\ of\ output\ keys\ in\ ith\ cluster,$
$SRi_i = reduce\ input\ data\ size\ of\ the\ ith\ cluster,$
$SRo_i = reduce\ output\ data\ size\ of\ the\ ith\ cluster,$
$CRi_i = reduce\ input\ data\ number\ of\ the\ ith\ cluster,$
$CRo_i = reduce\ output\ data\ number\ of\ the\ ith\ cluster,$
$Wi_i = reduce\ input\ weight\ of\ ith\ cluster,$
$Wo_i = reduce\ output\ weight\ of\ ith\ cluster.$

Different scenarios can be considered according to the number of clusters in which the data were located. In the scenario where the job is run on four different clusters, a notification with jobID is sent to DCM from DC_i who has completed the job successfully. The number of clusters that completed their jobs increased by 1. These are presented in Equations 2 and 3, as follows:

$$DC_{cmp} = 0 \tag{1}$$

$$DC_i \xrightarrow{jobID} DCM \forall i \in I \tag{2}$$

$$DC_{cmp} = DC_{cmp} + 1 \tag{3}$$

If the count of notifications DCM receives is 2 (4),

$$DC_{cmp} \geq 2 \tag{4}$$

From the clusters that complete their jobs, keys are sent from clusters with small output sizes (5) to those with large output sizes (6). The following equations are used for the operations mentioned here.

$$Dk_i \geq Dk_{i+1} \tag{5}$$

$$DC_{i+1} \xrightarrow{keys} DC_i \tag{6}$$

In the next step, the progress of the job of the clusters in the DCs whose data processing operations are in progress is checked, and if it is greater than the default value (t - the percentage of completion of the data processing operation), a new notification is waited from the clusters that have completed their jobs (7). If the output size of the keys of the clusters that sent the last notification is smaller than the size of the keys on DC_i (8), the keys are moved from DC_{i+2} to DC_i (9).

$$P_j > t \forall j \in J, 0 \leq P_j \leq 100 \tag{7}$$

$$\sum_{i=1}^{DC_{cmp}-1} Dk_i \geq Dk_{DC_{cmp}} \tag{8}$$

$$DC_{i+2} \xrightarrow{keys} DC_i \tag{9}$$

If the progress of the clusters in the data processing process is less than t (7), the data processing performance of the reduced function of clusters per unit time is calculated by taking into account the key distribution of the clusters who had completed their jobs. For the calculation, job type, reduce the input data size, reduce input key count, reduce output data size, reduce output key count, RAM, and CPU values of the node where the reduce function is running and the running time of the function are considered.

The input weight is calculated as the input data size of the reduction function divided by the number of input records (10), and the output weight is calculated as the output data size divided by the number of output records (11).

$$Wi_i = \frac{SRi_i}{CRi_i} \tag{10}$$

$$Wo_i = \frac{SRo_i}{CRo_i} \tag{11}$$

Because each key in the reduce function is computed and concluded by iteration, the input and output weights have a direct impact on computational power. Therefore, the input and output weights are calculated.

Because RAM and CPU have a direct impact on computational power, 2596 tests were performed on different data of the same data size with different numbers of RAM and CPU cores in the test environment specified in the Experimental Studies section of this study.

The parameters of the equation are presented in Table 1.

Sixth order (5-variable) polynomial regression model was used to calculate the coefficients showing the relationship between the variables forming the equation. The system was trained using 75% of the test data, and the prediction performance of the system was tested using 25% of the data. The prediction graph of the training data is shown in Figure 5, the prediction graph of the test data is shown in Figure 6, and the overlap graph of the actual and predicted data is shown in Figure 7.

The training time of the polynomial regression machine learning algorithm was 0.76 seconds and the prediction time was 1.4 ms. As a result on this model, an equation with 3002 coefficients was obtained. The performance of this model was evaluated using the total error squared score, which was 98%.

Using this equation, the costs of the nodes in the clusters where the reduce function runs are calculated. The clusters to which the key/values will be moved according to the costs is presented in the next section.

2) KEY DISTRIBUTION

This section presents the steps for shuffling the keys/values between clusters. Based on the cost calculated in the previous section and the condition that the key is in the cluster where

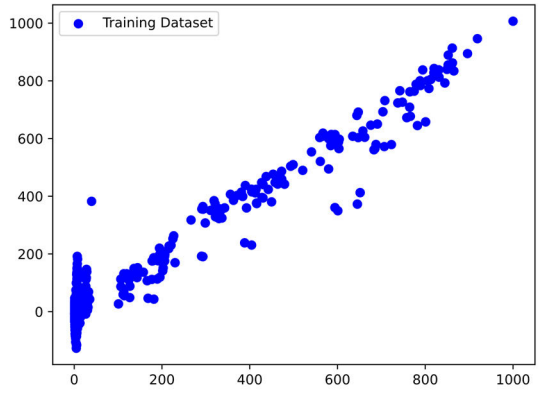


FIGURE 5. Result of predictions of training data.

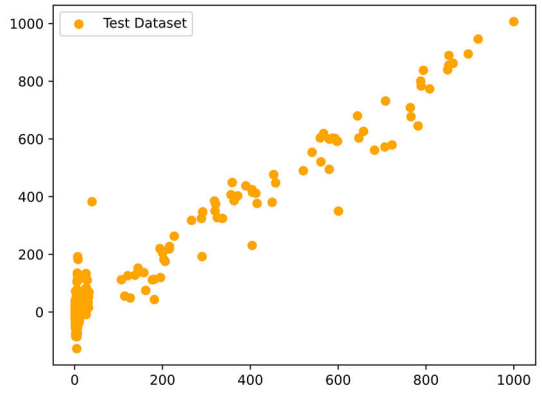


FIGURE 6. Result of predictions of test data.

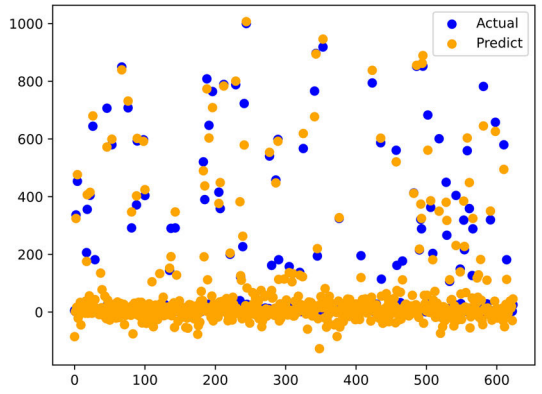


FIGURE 7. Overlap plot of actual and predicted data.

it is located, data distribution is performed between clusters, as presented in [20].

In the previous section, the clusters sent notifications of job completion. When the number of completed jobs in DCM is at least two, the size of the outputs on the clusters that only belong to the keys is checked using DCM. Keys are moved from clusters with smaller data sizes to clusters with larger data sizes. The determination of the clusters to which the keys/values are moved occurs in the cluster where the keys are collected.

The key distribution steps are as follows:

Step 1: The common keys in the data collected in the clusters are identified using a MapReduce job. Keys in clusters with the same key are grouped. For example, in Figure 8, key1, key3 and key6 are common to both DC_i and DC_{i+1} they form a group.

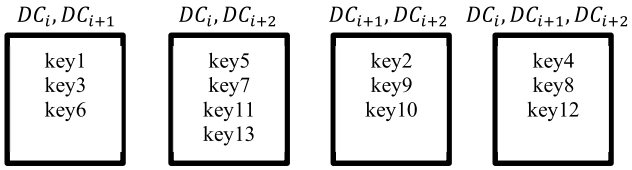


FIGURE 8. Cluster groups with the same keys.

Step 2: Grouping allows to calculate the costs in clusters. The number of keys to be assigned to clusters is determined according to the data processing costs in each group. The maximum number of groups to which the keys in clusters are allocated is calculated using the following formula (12).

$$2^{D_{cmp}} - \left(D_{cmp} + 1\right) \tag{12}$$

Because D_{cmp} is the number of clusters that have completed their jobs, the formula creates $2^{D_{cmp}}$ different subsets. (D_{cmp}) removes the key group and empty set that belongs to only one cluster in the subsets. If a key exists only in its own cluster, there is no need to group keys and then process the key/value pair in shuffling and move it between clusters. Therefore, it is not included in the cost calculations.

Step 3: Distribution of keys in multiple clusters is performed according to the cost coefficients calculated in the previous section.

All the clusters have an initial cost of 0. To determine the keys to be processed by the clusters forming each group, the inverse proportions of the costs of the clusters forming the group to each other are considered. Thus, the costs are intended to be close. The keys are assigned according to the cost of the clusters in each group. These are presented in Equations 13 and 14, as follows:

$$\frac{Cost_i}{Cost_{i+1}} = \frac{Ck_{i+1}^{DC_i,DC_{i+1}}}{Ck_i^{DC_i,DC_{i+1}}} \tag{13}$$

$$Ck_i^{DC_i,DC_{i+1}} + Ck_{i+1}^{DC_i,DC_{i+1}} = Count\left(DC_i, DC_{i+1}\right) \tag{14}$$

Step 4: The costs incurred when keys are assigned to each cluster are calculated. First, the cost of each cluster is calculated as given in Equation 15, which consists of distributing keys with the smallest cluster groups.

$$TCost_i = Ck_i^{DC_i,DC_{i+1}} * Cost_i \tag{15}$$

Step 5: In order to equalise the costs of clusters in other groups except for the group with the most clusters, the total cost of other clusters is subtracted from the cluster with the

largest total cost and the difference is calculated as shown in Equation 16.

$$Diff_{i+1,i} = Max\,(TCost_{i+1}) - TCost_i \quad 0 \le i \le DC_{cmp}, \forall i \in I \tag{16}$$

Step 6: The number of keys to be assigned to the clusters with the low cost is determined from the group with the most clusters, as shown in Equation 17.

$$Diff\ \ Count\left(Ck_i^{DC_i,DC_{i+1},DC_{i+2}}\right) = \frac{Diff_{i+1,i}}{Cost_i} \tag{17}$$

Step 7: After determining the number of keys to be assigned to the low-cost cluster, the distribution of remaining keys is determined according to their cost. key distributions are calculated as shown in Equation 18.

$$Ck_i^{DC_i,DC_{i+1},DC_{i+2}} + Ck_{i+1}^{DC_i,DC_{i+1},DC_{i+2}} + Ck_{i+2}^{DC_i,DC_{i+1},DC_{i+2}} = Count\,(DC_i, DC_{i+1}, DC_{i+2}) \tag{18}$$

Key distributions are sent to clusters. The target cluster is not sent the keys belonging to its cluster. The distributed keys are matched with the key/value pairs in the clusters and then shuffling is performed. After shuffling is completed, the reduce job is run.

Let us explain this method using the example shown in Figure 9. A cluster exists in each DC. Let us start a MapReduce job on four clusters located at different geographical locations. DC_A and DC_F send notifications to the DCM when the jobs running on them are completed. Suppose that the key output size on DC_A is 110 KB and 150 KB on DC_F. The keys are sent from DC_A to DC_F. DCM checks the progress of the clusters in progress. If there is a cluster with a progress status higher than a predefined value (let us assume 90%), according to the figure, the DC_B is waited to complete the job because it has 91%. After completion, suppose that the key size in DC_B is 100 KB. Because the key size in DC_F is 260 KB, the keys are moved from DC_B to DC_F. The job statuses of the clusters that continued their jobs are checked.

If there is no cluster with a progress status greater than 90%, the costs of the clusters (DC_A, DC_B, DC_F) that have completed their job are calculated using the equation determined with the weights of RAM, CPU, job type, and the input and output weights of the Reduce function. The costs are listed in Table 2.

TABLE 2. Cost table of clusters.

DC_A	DC_B	DC_F
1.5	3	2

The common keys in the clusters are determined. They are grouped as clusters with the same key (Figure 10). Starting from the smallest groups, the keys are distributed according to the costs.

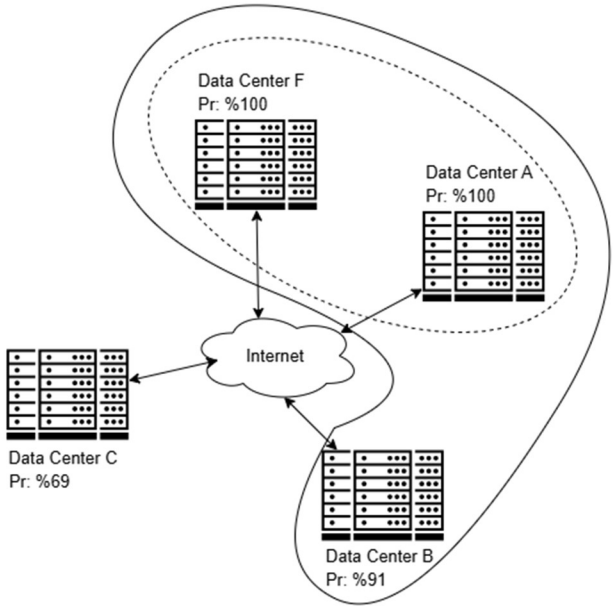


FIGURE 9. Example topology for the GSelf-MapReduce.

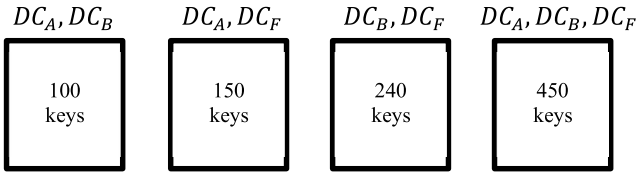


FIGURE 10. Cluster groups with the same keys.

According to the costs in the table, the costs of DC_A, DC_B and DC_F. are 1.5, 3, and 2, respectively. There are 100 keys in both the group with DC_A, DC_B. DC_A is assigned 66 keys and, DC_B is assigned 34 keys. The costs of DC_A and DC_B are similar and are approximately 100. The group with DC_A, DC_F has 150 keys. According to the costs, DC_A is assigned 85 keys and DC_F is assigned 65 keys. The costs of DC_A and DC_F are similar and are approximately 130. There are 240 keys in both the group with, DC_B, DC_F. According to these costs, DC_B is assigned 96 keys and DC_F is assigned 144 keys. The costs of DC_B and DC_F are close to each other and are approximately 288. The sum of the cluster costs by distributing the keys in these three groups, $TCost_A$, $TCost_B$ and $TCost_F$ are 230, 388, and 418, respectively. This is the key distribution for all other key groups except for the group to which all clusters belong. In the last group, the distribution is performed by taking the difference in the cluster costs. According to the keys distributed in the previous groups, $TCost_F$ incurs the largest cost. Before determining the distribution of keys in this group, the costs of the clusters must be equalized.

TABLE 3. Number of keys in clusters and cost of clusters after key distribution.

Group	Cluster	Keys
Group1	DC_A	66
	DC_B	34
Group2	DC_A	85
	DC_F	65
Group3	DC_B	96
	DC_F	144
Group4	DC_A	140+125
	DC_B	70+10
	DC_F	105

The cost of DC_F is 188 more than DC_A and 30 more than DC_B. Assigning 125 keys to DC_A and 10 keys to DC_B equalizes the costs approximately. The remaining 315 keys in this group were distributed according to the costs. The distributions of the remaining keys in the last group to DC_A, DC_B and DC_F are 140, 70, and 105, respectively. The key counts according to the costs of the clusters forming the groups are listed in Table 3, and the costs of the clusters are listed in Table 4. The key distributions are then sent to the clusters. Shuffling occurs between the clusters. Reduce job is run. This process continues until all clusters are exchanged in the same phase.

TABLE 4. Costs after keys are distributed.

Cluster	Total Cost
DC_A	627.5
DC_B	630
DC_F	628

Then the key distributions are sent to the clusters. Shuffling takes place between clusters. Reduce job is run. This process continues until all clusters change in the same phase.

IV. EXPERIMENTAL RESULTS

The proposed GSelf-MapReduce method is implemented in a distributed cluster environment. An environment consisting of four clusters was created for this purpose. In this environment, each cluster has a different hardware and data size. Clusters were created on servers at Düzce University, each on different networks. Each server node was considered an independent data center. Each data center had a cluster. In addition to the clusters, DCM ensures the continuity of the workflows of the clusters (Figure 11).

The hardware specifications and bandwidth of each cluster are listed in Table 5, and the data volumes are listed in Table 6.

The test applications were run on Hadoop 3.2.4 on an Ubuntu 20.04 LTS operating system. The Puma dataset [40]was used as data, and word count, inverted index, and adjacency-list applications were used to test the method.

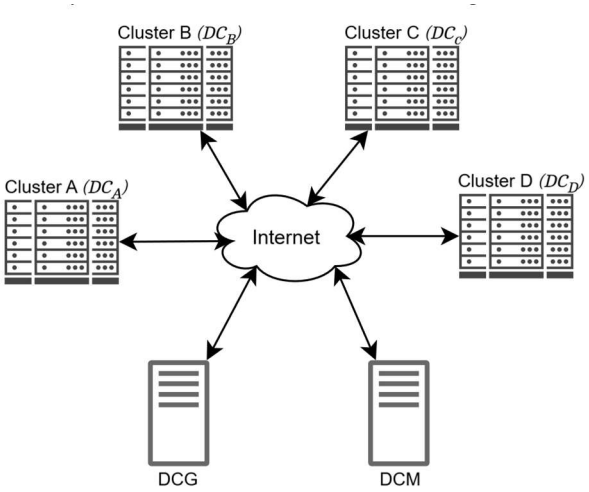


FIGURE 11. Schematic of the Testbed.

TABLE 5. Table of hardware specifications of clusters.

Cluster	Hardware	Bandwidth (Mbit)
DC_A	Intel Xeon E5-2680 2.50GHz 2 CPU 6 GB RAM	20
DC_B	Intel Xeon E5-2670 2.60GHz 4 CPU 4 GB RAM	20
DC_C	Intel Xeon Gold 5218R 2.10GHz 4 CPU 2 GB RAM	20
DC_D	Intel(R) Xeon E5-26602.60GHz 2 CPU 4 GB RAM	20
DCM	Intel Xeon E5-2680 2.50GHz 2 CPU 2 GB RAM	-

TABLE 6. Data volume for each application in the clusters.

Cluster	Word Count	Inverted Index	Adjacency-List
DC_A	4G	4G	7.5G
DC_B	6G	7.5G	6.5G
DC_C	6.5G	5G	4.2G
DC_D	9G	12G	8.3G

The testing of the proposed method in experimental studies consisted of two phases. This is because the hardware and data volumes between clusters are considered to be close to each other owing to the hardware and data volume limitations in academic work environments. If they are not close to each other, it may consist of more than two phases.

Word count, inverted index and adjacency-list applications were run on four clusters in the test environment. The key and key/value volumes generated in each cluster after execution are shown in Figure 12 for the word count application, Figure 13 for the inverted index application, and Figure 14 for the adjacency-list application.

As seen in Figure 12, the output key and key/value volumes of the word count application are close to each other, as seen

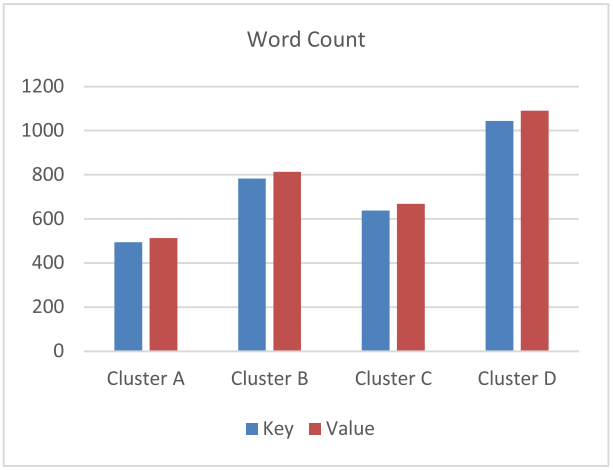


FIGURE 12. Word count output key and output key/value volume (mb).

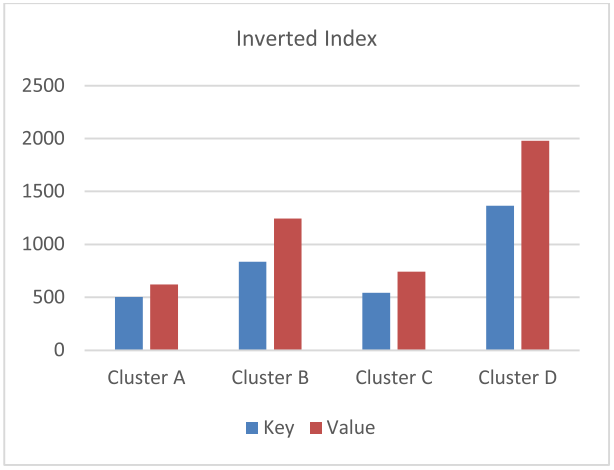


FIGURE 13. Inverted index output key and output key/value volume (mb).

in Figure 13, the output key volume of the inverted index application is approximately 25% less than the key/value volume, and as seen in Figure 14, the output key volume of the inverted index application is very small than the key/value volume.

As in Cross-MapReduce and Hierarchical method, in GSelf-MapReduce each cluster runs the MapReduce application on itself first. In the Hierarchical approach, after running the MapReduce job in each cluster, all the data generated are sent to a cluster selected as the global reducer and the reducer application are run. In Cross-MapReduce, only the outputs containing the keys are sent to the main cluster and the GRG is created. The keys are then sent to multiple global reducers.

In GSelf-MapReduce, keys are sent from a cluster with a small key volume to a cluster with a large total key volume. The job progress of the clusters that continued their jobs was checked. If the progress is less than the predefined threshold value, the key distribution of the clusters that have completed their job is determined according to the running performance of the reduce function.

Keys are sent to the clusters. According to these distributions, the clusters perform shuffling. After the shuffling process is completed, the reduce function is executed on the data in these clusters (Phase-1). Then, the processes in phase-1 are performed until all clusters complete their jobs.

A. TEST RESULTS OF WORD COUNT ON THE GSELF-MAPREDUCE

For the word count job, the first DC_A completes it in 14.5 min. Send notification to DCM. The DCM waits for notifications from another cluster. Then, the DC_B notifies that it has completed at 16.9 minutes. Because the key volume is smaller in DC_A, keys are transferred from DC_A to DC_B. DCM checks the job status of the clusters that continue their jobs. If their job status is more than 60%, it waits for the clusters to complete their jobs. DC_C's progress was 94.7% and DC_D's is

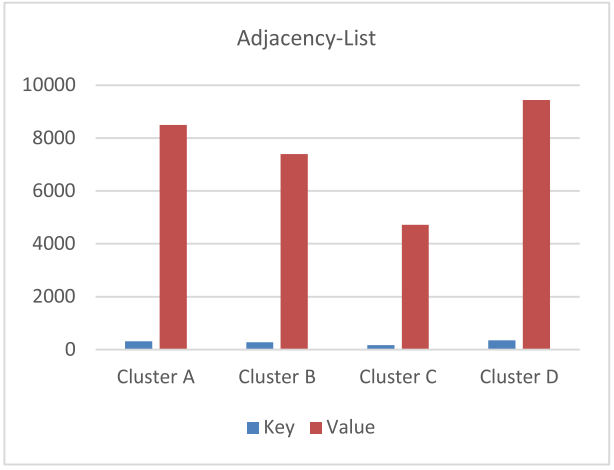


FIGURE 14. Adjacency-list output key and output key/value volume (mb).

52.1%. DC_C completes its job. The total key volume in DC_B is compared to the volume of DC_C. Because DC_C is small, keys are moved from DC_C to DC_B. Subsequently, in DC_B, the common keys in the clusters are grouped.

The computational capacities of the three clusters (DC_A, DC_B, DC_C) are determined. The grouped keys are distributed according to the capacities of these clusters. Shuffling is performed between clusters. Reduce function is executed.

The execution times of DC_A, DC_B and DC_C clusters are shown in Figure 15, and the data volume moved from the clusters is shown in Figure 16.

Then DC_B completes the Reducing-1 phase in 34.64 minutes, followed by DC_A in 34.67 minutes. Since the key size of DC_A is smaller than DC_B, keys are transferred from DC_A to DC_B. The progress states of DC_C and DC_D are checked. Since their job progress are 99.1% and 98.7% respectively, they are waited to complete their jobs. After all clusters complete the job, the steps in Phase-1 are repeated. When all clusters complete their jobs at the same time, the distributed MapReduce job is completed.

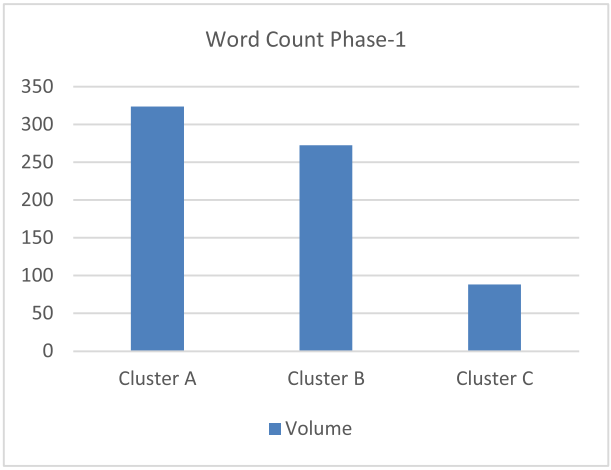


FIGURE 15. Word count phase-1 make span (min).

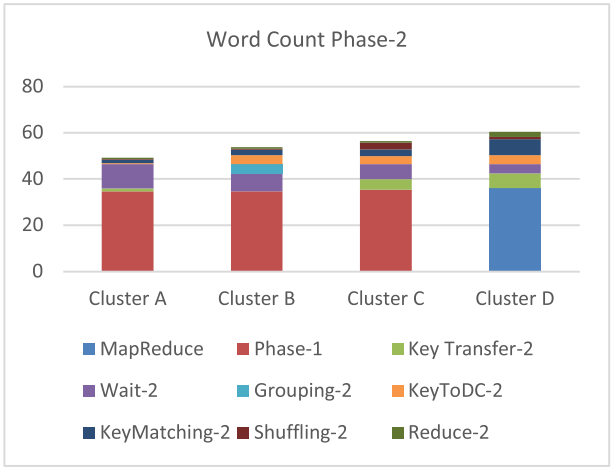


FIGURE 17. Word count phase-2 make span (min).

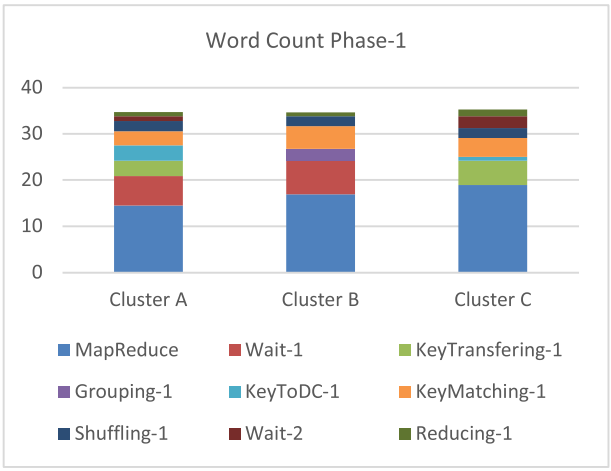


FIGURE 16. Data volume moved between clusters for word count (mb).

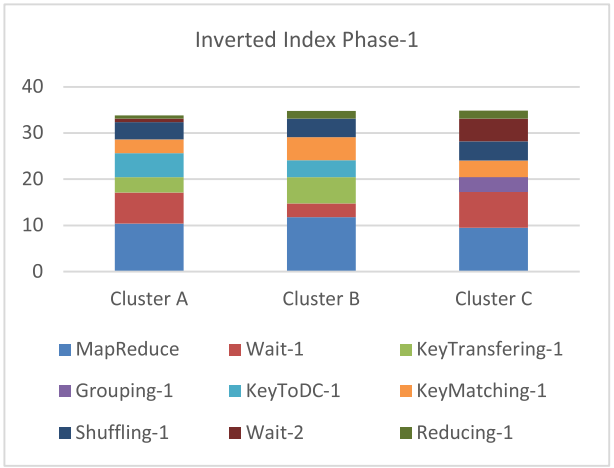


FIGURE 18. Inverted index phase-1 make span (min).

The data processing times of the clusters using the proposed method are shown in Figure 17.

DC_A, DC_B and DC_C reduce their data by completing the first phase among themselves according to the threshold value before the initial job of DC_D.

B. TEST RESULTS OF INVERTED INDEX ON THE GSELF-MAPREDUCE

For the inverted index job, DC_C first reports completion in 9.45 min and then DC_A reports completion in 10.38 min. Since the key size is smaller in DC_A, keys are transferred from DC_A to DC_C. If the work progress of the ongoing clusters is more than 60%, the clusters are waited to complete their work. DC_B's progress is 97.7% and DC_D's is 46.3%. DC_B completes its work. Since the key size of DC_B is smaller, keys are moved from DC_B to DC_C. Then, in DC_C, the common keys of the clusters are grouped. The computational capacities of the three finished DCs (DC_A, DC_B, DC_C) are determined. The grouped keys are distributed according to the capacities of the clusters. Shuffling and reduce function are executed between clusters.

The execution times of DC_A, DC_B and DC_C is given in Figure 18, and the data size moved from clusters is given in Figure 19.

Then, DC_D completes the Reducing-1 phase in 33.23 min. Then, DC_A completes its work in 33.81 min. Since the key size of DC_A is smaller than DC_D, keys are transferred from $DC_A$to DC_D. Since the progress states of DC_B and DC_C are 99.3% and 99.6% respectively, they are waited to complete their jobs. After all clusters complete the job, the steps in phase-1 are repeated.

The execution times of inverted index for GSelf-MapReduce method is given in Figure 20. Since DC_D is below the threshold value, DC_A, DC_B, DC_C performed the shuffling without waiting for DC_D. DC_A, DC_B and DC_C reduced their data by completing the first phase among themselves earlier than DC_D's initial job.

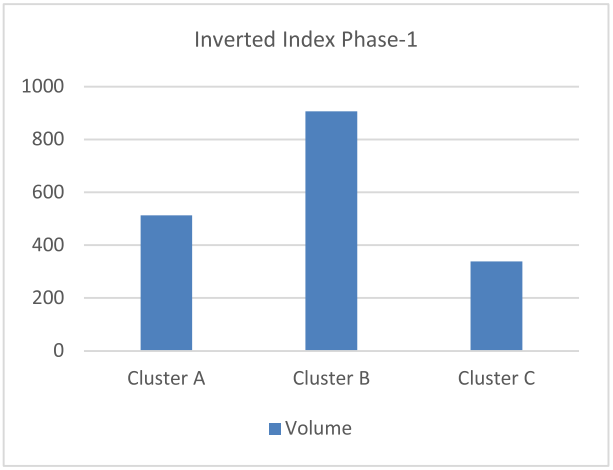


FIGURE 19. Data volume moved between clusters for inverted index (mb).

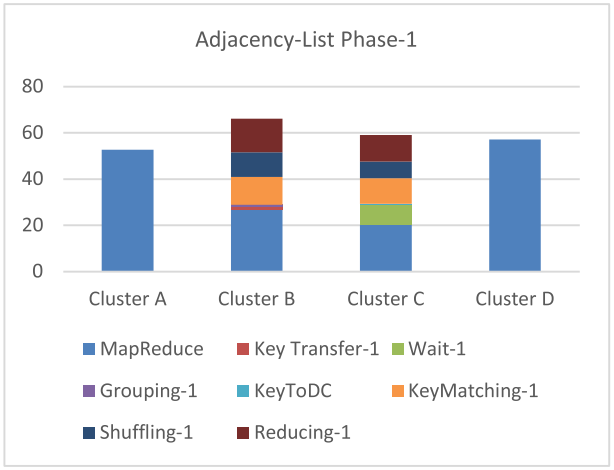


FIGURE 21. Adjacency-List phase-1 make span (min).

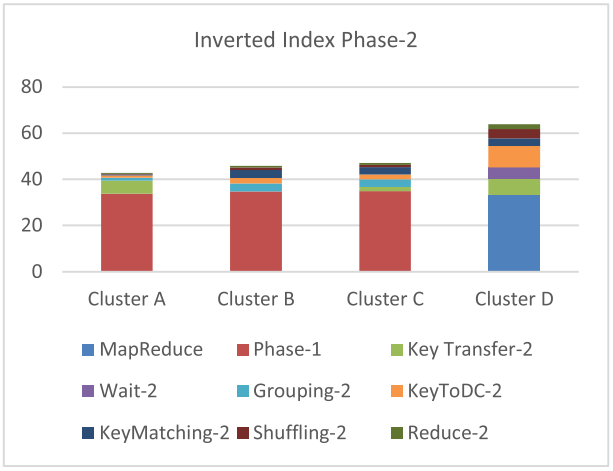


FIGURE 20. Inverted index phase-2 make span (min).

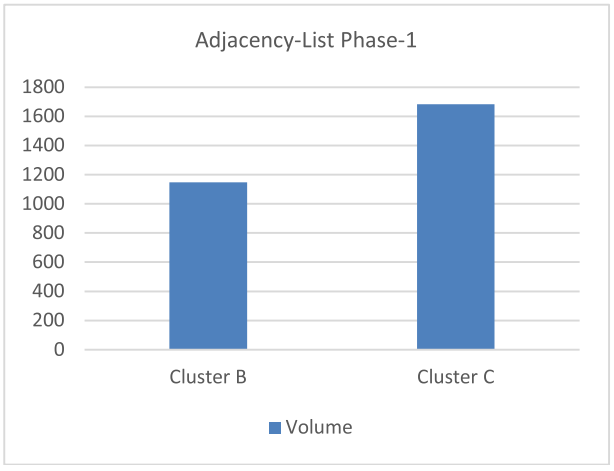


FIGURE 22. Data volume moved between clusters for adjacency-list (mb).

C. TEST RESULTS OF ADJANCECY-LIST ON THE GSELF-MAPREDUCE

DC_C and DC_B complete the adjacency-list job in 20.17 and 26.77 minutes respectively. Since the key size is smaller in DC_C, keys are transferred from DC_C to DC_B. If the job status is more than 60%, it waits for the servers to complete their work. The progress of DC_A is 55.6% and that of DC_D is 48.4%. The common keys in the clusters are grouped in DC_B. The keys are distributed according to the costs of the two clusters (DC_B, DC_C) that complete their work. Shuffling is performed between clusters. Reduce function is executed.

The generation time of DC_B and DC_C is given in Figure 21 and the data size moved from clusters is given in Figure 22.

Then $\boldsymbol{DC_C}$ completes the Reducing-1 phase in 50.36 min. Then, $\boldsymbol{DC_A}$ completes its job in 52.68 min. Since the key size of $\boldsymbol{DC_C}$ is smaller than $\boldsymbol{DC_A}$, keys are transferred from $\boldsymbol{DC_C}$ to $\boldsymbol{DC_A}$. The progress states of $\boldsymbol{DC_B}$ and $\boldsymbol{DC_D}$ are checked. Since their work progress are 94% and 85% respectively, they are waited to complete their work. Similarly, keys from other clusters that complete their jobs are moved to $\boldsymbol{DC_D}$. When $\boldsymbol{DC_B}$ completes its job, keys are moved from $\boldsymbol{DC_B}$ to $\boldsymbol{DC_A}$ since the key size of $\boldsymbol{DC_B}$ is small. The data processing capacities of all clusters are realised in key distribution $\boldsymbol{DC_A}$ The shuffling and reduce processes are performed as presented in the previous section.

The execution times of adjacency-list for the GSelf-MapReduce method is given in Figure 23.

Since DC_A and DC_D were below the threshold value, DC_B, DC_C perform the shuffling without waiting for DC_A, DC_D as shown in Figure 23.

D. COMPARISON OF GSELF-MAPREDUCE RESULTS WITH OTHER METHODS

In the Cross-MapReduce, Hierarchical and Geo-Hadoop approaches, shuffling is performed when all clusters are completed. Because GSelf-MapReduce can have multiple phases, we compared the data volume realized in the last phase of shuffling with the data volume of the other approaches.

Figure 24 shows the shuffling data volume in the word count application. For GSelf-MapReduce, because DC_A,

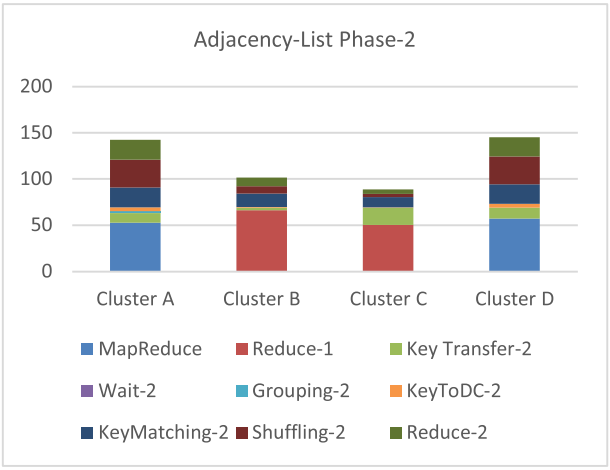


FIGURE 23. Adjacency-list phase-2 make span (min).

DC_B and DC_C perform phase-1 shuffling and reduction, the keys in these clusters are reduced. In other words, the keys between these clusters are unique. Figure 24 shows the shuffling data volumes between clusters in the second phase of the word count application. It can be observed that there is no same between the keys of DC_A and DC_D. Furthermore, the same is true of DC_B and DC_D. There are no shuffling from, DC_A and DC_B to DC_D. Therefore, as shown in Figure 24, the shuffling size of, DC_A and DC_B are zero. Because the common keys are located between DC_C and DC_D, shuffling occurs between these two clusters.

The shuffling data traffic is less than that of the other methods. In the Hierarchical approach, DC_D, in which the key/value volume is large, is selected as the global reducer for this operation. Thus, the data in DC_D are ignored during shuffling. Nevertheless, Cross-MapReduce generates less data traffic than Hierarchical and Geo-Hadoop because it shuffles in the clusters where the keys are located.

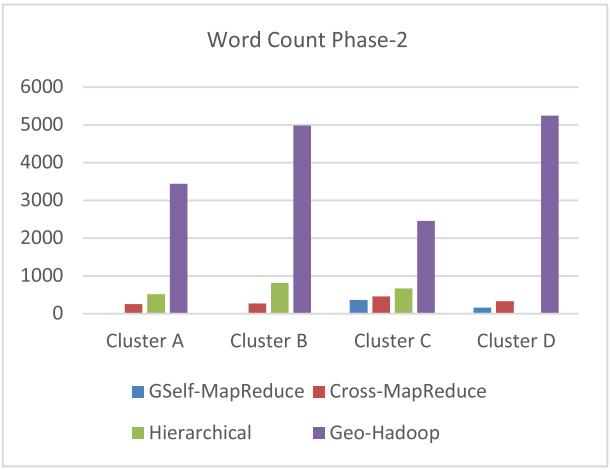


FIGURE 24. Data volume moved from all clusters for word count (mb).

For the inverted index, because DC_A, DC_B and DC_C clusters perform shuffling and reduction in the first phase, the data in these clusters are reduced. Since the key in DC_A is not present in DC_D, no shuffling takes place in these DC_A and DC_D. Because common keys are found between DC_B, DC_C and DC_D, shuffling occurs between these clusters. Because the key/values between DC_B and DC_C are unique, shuffling occurs between DC_B and DC_D, and as well as between DC_C and DC_D. In GSelf-MapReduce, the shuffling data traffic is less than that of the other methods, as shown in Figure 25. In the Hierarchical approach, DC_D is chosen as the global reducer for the inverted index because of its large key/value volume. Thus, the data in DC_D are ignored for shuffling. The Hierarchical and Cross-MapReduce approaches have almost the same shuffling traffic in the clusters. Geo-Hadoop generates a large volume of data.

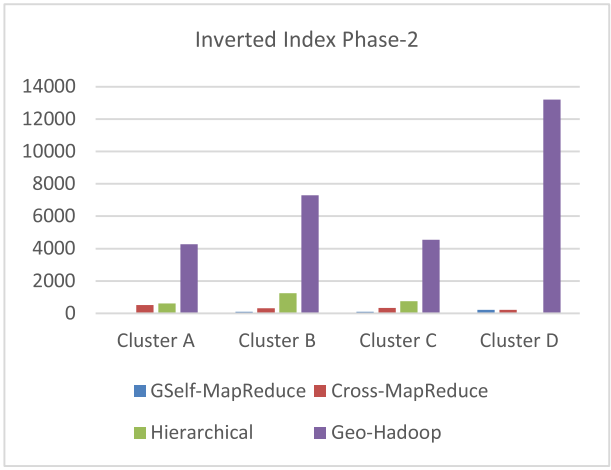


FIGURE 25. Data volume moved from all clusters for inverted index (mb).

For the adjacency-list, because $\boldsymbol{DC_B}$ and $\boldsymbol{DC_C}$ clusters perform shuffling and reduction in the first phase, the data to be processed in these clusters are reduced. Although the key/values between $\boldsymbol{DC_B}$ and $\boldsymbol{DC_C}$ are unique, shuffling occurs because there are common keys between all clusters. In GSelf-MapReduce, as shown in Figure 26, although the shuffling volume in $\boldsymbol{DC_D}$ is greater than that in Cross-MapReduce, the total shuffling data traffic is less than that in Cross-MapReduce.

In the Hierarchical approach, DC_D, in which the key/value volume is large, is chosen as the global reducer. In the Hierarchical approach, the data in DC_D are ignored. The Hierarchical approach has more data traffic than the Cross-MapReduce approach does. Geo-Hadoop has the highest data volume compared with the other approaches.

Table 7 shows the total volume of shuffled data traffic and the total processing time of the applications that we ran. The data processing time in the word count application is approximately the same as that in our proposed GSelf-MapReduce and Hierarchical approach. The most important reason is that the key volume is close to the key/value volume. This is because grouping and distributing keys in a cluster and moving these keys to clusters incurs time costs. Although the Cross-MapReduce approach produces less shuffling volume

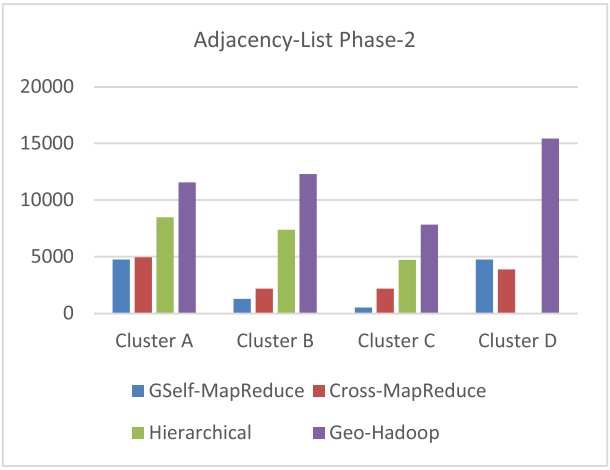


FIGURE 26. Data volume moved from all clusters for adjacency-list (mb).

TABLE 7. Processing times (min) and shuffling volumes (mb) of the applications in different approaches.

Approaches	Word Count		Inverted Index		Adjacency-List	
	Make Span	Shuf. Vol.	Make Span	Shuf. Vol.	Make Span	Shuf. Vol.
GSelf-MapReduce	60.33	519	63.86	420	145.3	11323
Cross-MapReduce	77.2	1316	74.34	1384	182.3	13210
Hierarchical MapReduce	61.7	1994	82.56	2607	546.6	20610
Geo-Hadoop	202.1	16125	248	29321	2304	47152

than the Hierarchical approach, its production time is higher. In addition, the shuffling volume of the proposed method is 60.4% less than that of Cross-MapReduce, which is the closest method in terms of volume. This is because 66% of the total data were processed in the first phase.

The data-processing time of GSelf-MapReduce in the inverted index is shorter than that of Cross-MapReduce. In addition, the shuffling data volume was 69.7% smaller than that of the same method. Cross-MapReduce is followed by the Hierarchical and Geo-Hadoop approaches. The Geo-Hadoop shuffling data volume is significantly larger than the input data volume. This creates a burden on the network traffic, which considerably increases the data-processing time. Moreover, in the first phase of GSelf-MapReduce, 68.9% of the total data are reduced by processing.

In the adjacency-list, the closest approach to the shuffling data size of GSelf-MapReduce is Cross-MapReduce. It can be seen that the final phase results in about 15% less data volume than Cross-MapReduce. The reduce phase of the adjacency-list application requires intensive data processing. Because the volume of the shuffled data directly affects the volume of the data that the reducers process, the processing time of the data increases. An example is the Hierarchical approach. This is because the data moved from the clusters are moved to a single cluster and processed by the global reducer. In Geo-Hadoop, on the other hand, because of the participation in the output shuffling of the map phase, the volume of data coming out of the clusters is very large. In addition, the proposed method processed 44.6% of the total data in the first phase, reducing the data volume to be processed.

One of the advantages of GSelf-MapReduce over Cross-MapReduce in terms of time is that no time is spent creating GRGs. Instead, it is sufficient to determine the cost of machines. The keys of the clusters in the common key groups are distributed proportionally to these costs. In addition, the data-processing costs of the clusters are considered.

As observed in this study, clusters with different hardware specifications and data volumes have different data processing times. While Cross-MapReduce, Hierarchical Hadoop, and Geo-Hadoop, which are distributed MapReduce data processing approaches, wait for all clusters to complete their job, in the proposed approach, clusters with progress greater than a certain threshold are shuffled among themselves and then reduced. Thus, the amount of data to be processed is reduced. Thus, clusters that are slower, have larger data volumes, and have lower processing speeds are not waited to complete their jobs. This reduces the shuffling traffic and processing time for all clusters.

V. DISCUSSION

To process data in distributed data centres, the data must either be moved to a location or processed at that location. In this study, a method for processing data in distributed data centres with MapReduce is presented. In this method, it is ensured that the clusters that finish their work shuffle among themselves. For this, the data processing progress of the clusters is checked. If there is a cluster with a progress status greater than a certain threshold value, it is waited to complete its work. They are allowed to shuffle among themselves. The threshold value is predefined according to the experience in previous completed jobs. Determining the threshold value correctly is critical for GSelf-MapReduce. When a low threshold value is set, more clusters complete shuffling and reduce data among themselves. However, since the threshold value is low, the clusters that complete their work wait for the other ongoing clusters to complete their work. Thus, although the data carried by all clusters in the last phase is reduced, the total job completion time may increase. In addition, in cases where the threshold value is set too high, the total job duration may be affected. Because there is little time left for the completion of the jobs in the ongoing clusters. In order to distribute the keys for the clusters that have completed their work in the remaining time, collecting the keys in a cluster, ensuring the distribution of these keys, shuffling and then reducing creates a time cost. While this situation continues, when the last remaining cluster completes its work, it waits for clusters that shuffle and reduce among themselves. When the threshold value is set correctly, the clusters complete the shuffling and reduce processes among

themselves, reducing the shuffling data volume in the last phase and total job completion time.

Since the tests of the study were carried out in an isolated network in an experimental environment, real-time throughput and latency were not focused.

In addition, the data processing costs of the clusters in the data centres are determined for the distribution of keys. Clusters are assigned keys in proportion to their costs. To determine the assignment of keys, the keys are grouped. This ensures a balanced and fast distribution of keys between the clusters in which the key distribution is located.

VI. CONCLUSION

Storing and processing data in distributed data centers is a necessity for this age. This is because it is important to reduce the cost of data transportation by storing data at the location where it is generated. It may be desirable to obtain a result by processing the distributed data. MapReduce is one of the methods proposed for processing distributed data. In this study, a method called GSelf-MapReduce was proposed based on Cross-MapReduce. A similar but more efficient approach is presented for distributing the keys in Cross-MapReduce by considering the heterogeneity of clusters. Using this method, clusters are shuffled among themselves by examining their job progress. Thus, not all clusters are waited to complete the data processing. The data is shuffled. After shuffling is completed, a reduction operation occurs. Because this operation is performed, the amount of data to be shuffled in the next stage is reduced. It also reduces the data to be processed for jobs that need to run an intensive reduction function. Therefore, the runtime of the reduced function is reduced.

The proposed GSelf-MapReduce method is compared with Geo-Hadoop, Hierarchical and Cross-MapReduce. Word count, inverted index and adjacency-list applications were used to compare this study with other methods. The advantages of GSelf-MapReduce in terms of time and shuffling data volume are presented.

In this study, the bandwidth factor is assumed to be homogeneous. In future studies, we aim to evaluate the effects of different bandwidths on this method. In addition, in this method, the threshold value is used as the default for the progress of jobs. In the future, the effects of determining this value by evaluating it using data volume and job completion time can be investigated.

REFERENCES

[1] S. Yu, M. Liu, W. Dou, X. Liu, and S. Zhou, "Networking for big data: A survey," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 531–549, 1st Quart., 2017, doi: 10.1109/COMST.2016.2610963.

[2] A. Katal, M. Wazid, and R. H. Goudar, "Big data: Issues, challenges, tools and good practices," in *Proc. 6th Int. Conf. Contemp. Comput. (IC3)*, Aug. 2013, pp. 404–409, doi: 10.1109/IC3.2013.6612229.

[3] P. Zikopoulos, C. Eaton, D. DeRoos, T. Deutsch, and G. Lapis, *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. New York, NY, USA: McGraw-Hill, 2012.

[4] E. Kirda, E. Schwartz, W. H. Stewart, A. Yoran, and S. Curry. (2013). *Big Data Fuels Intelligence-Driven Security*. RSA Secur. Brief. Accessed: Jul. 29, 2023. [Online]. Available: https://silo.tips/download/big-data-fuels-intelligence-driven-security

[5] S. Sagiroglu and D. Sinanc, "Big data: A review," in *Proc. Int. Conf. Collaboration Technol. Syst. (CTS)*, May 2013, pp. 42–47, doi: 10.1109/CTS.2013.6567202.

[6] Y. Zhang, H. Ma, and Y. Xu, "An intelligence gathering system for business based on cloud computing," in *Proc. 6th Int. Symp. Comput. Intell. Design*, vol. 1, Oct. 2013, pp. 201–204, doi: 10.1109/ISCID.2013.57.

[7] *77+ Surreal Big Data Statistics to Map Growth in 2024*. Accessed: Feb. 28, 2024. [Online]. Available: https://www.g2.com/articles/big-data-statistics

[8] *10 Google Search Statistics You Need to Know In 2023*. Accessed: Feb. 28, 2024. [Online]. Available: https://www.oberlo.com/blog/google-search-statistics

[9] *30+ Instagram Statistics*. Accessed: Feb. 28, 2024. [Online]. Available: https://www.zippia.com/advice/instagram-statistics/

[10] *Facebook Messenger Revenue and Growth Statistics*. Accessed: Feb. 28, 2024. [Online]. Available: https://www.usesignhouse.com/blog/facebook-messenger-stats

[11] *Apache Hadoop*. Accessed: Jul. 16, 2023. [Online]. Available: https://hadoop.apache.org

[12] *What is Apache Hadoop?*. Accessed: Aug. 4, 2024. [Online]. Available: https://cloud.google.com/learn/what-is-hadoop

[13] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, K. Karanasos, J. Padhye, and G. Varghese, "WANalytics: Geo-distributed analytics for a data intensive world," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*. New York, NY, USA: ACM, May 2015, pp. 1087–1092, doi: 10.1145/2723372.2735365.

[14] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low latency geo-distributed data analytics," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 421–434, Sep. 2015, doi: 10.1145/2829988.2787505.

[15] Z. Liu, X. Yuan, J. Yuan, J. Zhang, Z. Gu, and L. Zhang, "Multi-stage geo-distributed data aggregation with coordinated computation and communication in edge compute first networking," *J. Lightw. Technol.*, vol. 41, no. 8, pp. 2289–2300, Apr. 15, 2023, doi: 10.1109/JLT.2022.3232840.

[16] Y. Luo and B. Plale, "Hierarchical MapReduce programming model and scheduling algorithms," in *Proc. 12th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2012, pp. 769–774, doi: 10.1109/CCGRID.2012.132.

[17] J. P. Verma, S. H. Mankad, and S. Garg, "Big data analytics: Performance evaluation for high availability and fault tolerance using MapReduce framework with HDFS," in *Proc. 5th Int. Conf. Parallel, Distrib. Grid Comput. (PDGC)*, Dec. 2018, pp. 770–775, doi: 10.1109/PDGC.2018.8745770.

[18] T. White, *Hadoop: The Definitive Guide*, 3rd ed., Sebastopol, CA, USA: O'Reilly Media, 2012.

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

[20] S. M. Marzuni, A. Savadi, A. N. Toosi, and M. Naghibzadeh, "Cross-MapReduce: Data transfer reduction in geo-distributed MapReduce," *Future Gener. Comput. Syst.*, vol. 115, pp. 188–200, Feb. 2021, doi: 10.1016/j.future.2020.09.009.

[21] A. Verma, A. H. Mansuri, and N. Jain, "Big data management processing with Hadoop MapReduce and spark technology: A comparison," in *Proc. Symp. Colossal Data Anal. Netw. (CDAN)*, Mar. 2016, pp. 1–4, doi: 10.1109/CDAN.2016.7570891.

[22] I. Chebbi, W. Boulila, N. Mellouli, M. Lamolle, and I. R. Farah, "A comparison of big remote sensing data processing with Hadoop MapReduce and spark," in *Proc. 4th Int. Conf. Adv. Technol. Signal Image Process. (ATSIP)*, Mar. 2018, pp. 1–4, doi: 10.1109/ATSIP.2018.8364497.

[23] M. Cavallo, G. Di Modica, C. Polito, and O. Tomarchio, "H2F: A hierarchical Hadoop framework for big data processing in geo-distributed environments," in *Proc. IEEE/ACM 3rd Int. Conf. Big Data Comput. Appl. Technol. (BDCAT)*. New York, NY, USA: ACM, Dec. 2016, pp. 27–35, doi: 10.1145/3006299.3006320.

[24] A. Iordache, C. Morin, N. Parlavantzas, E. Feller, and P. Riteau, "Resilin: Elastic MapReduce over multiple clouds," in *Proc. 13th IEEE/ACM Int. Symp. Cluster, Cloud, Grid Comput.*, May 2013, pp. 261–268, doi: 10.1109/CCGRID.2013.48.

[25] J. Wang, X. Li, R. Ruiz, J. Yang, and D. Chu, "Energy utilization task scheduling for MapReduce in heterogeneous clusters," *IEEE Trans. Services Comput.*, vol. 15, no. 2, pp. 931–944, Mar. 2022, doi: 10.1109/TSC.2020.2966697.
[26] D. L. Quoc, C. Fetzer, P. Felber, É. Rivière, V. Schiavoni, and P. Sutra, "UniCrawl: A practical geographically distributed web crawler," in *Proc. IEEE 8th Int. Conf. Cloud Comput.*, Jun. 2015, pp. 389–396, doi: 10.1109/CLOUD.2015.59.
[27] S. Kumar and K. K. Mohbey, "A utility-based distributed pattern mining algorithm with reduced shuffle overhead," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 1, pp. 416–428, Jan. 2023, doi: 10.1109/TPDS.2022.3221210.
[28] L. Chen, S. Liu, and B. Li, "Optimizing network transfers for data analytic jobs across geo-distributed datacenters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 2, pp. 403–414, Feb. 2022, doi: 10.1109/TPDS.2021.3093232.
[29] W. Xiao, W. Bao, X. Zhu, and L. Liu, "Cost-aware big data processing across geo-distributed datacenters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 11, pp. 3114–3127, Nov. 2017, doi: 10.1109/TPDS.2017.2708120.
[30] T. Gouasmi, W. Louati, and A. H. Kacem, "Geo-distributed BigData processing for maximizing profit in federated clouds environment," in *Proc. 26th Euromicro Int. Conf. Parallel, Distrib. Netw.-Based Process. (PDP)*, Mar. 2018, pp. 85–92, doi: 10.1109/PDP2018.2018.00020.
[31] Y. Chen, L. Luo, B. Ren, and D. Guo, "Geo-distributed IoT data analytics with deadline constraints across network edge," *IEEE Internet Things J.*, vol. 9, no. 22, pp. 22914–22929, Nov. 2022, doi: 10.1109/JIOT.2022.3186173.
[32] X. Xu, W. Li, R. Xu, H. Qi, K. Li, X. Zhou, and S. Chen, "Trading cost and throughput in geo-distributed analytics with a two time scale approach," *IEEE Trans. Cloud Comput.*, vol. 10, no. 3, pp. 2163–2177, Jul. 2022, doi: 10.1109/TCC.2020.2994195.
[33] Z. Xiao, H. Chen, and B. Zang, "A hierarchical approach to maximizing MapReduce efficiency," in *Proc. Int. Conf. Parallel Archit. Compilation Techn.*, Oct. 2011, pp. 167–168, doi: 10.1109/PACT.2011.22.
[34] C. Jayalath, J. Stephen, and P. Eugster, "From the cloud to the atmosphere: Running MapReduce across data centers," *IEEE Trans. Comput.*, vol. 63, no. 1, pp. 74–87, Jan. 2014, doi: 10.1109/TC.2013.121.
[35] B. Heintz, A. Chandra, R. K. Sitaraman, and J. Weissman, "End-to-end optimization for geo-distributed MapReduce," *IEEE Trans. Cloud Comput.*, vol. 4, no. 3, pp. 293–306, Jul. 2016, doi: 10.1109/TCC.2014.2355225.
[36] S. Imai, C. A. Varela, and S. Patterson, "A performance study of geo-distributed IoT data aggregation for fog computing," in *Proc. IEEE/ACM Int. Conf. Utility Cloud Comput. Companion (UCC Companion)*, Dec. 2018, pp. 278–283, doi: 10.1109/UCC-Companion.2018.00068.
[37] L. Wang, J. Tao, R. Ranjan, H. Marten, A. Streit, J. Chen, and D. Chen, "G-Hadoop: MapReduce across distributed data centers for data-intensive computing," *Future Gener. Comput. Syst.*, vol. 29, no. 3, pp. 739–750, Mar. 2013, doi: 10.1016/j.future.2012.09.001.
[38] M. Bergui, S. Najah, and N. S. Nikolov, "A survey on bandwidth-aware geo-distributed frameworks for big-data analytics," *J. Big Data*, vol. 8, no. 1, p. 40, Dec. 2021, doi: 10.1186/s40537-021-00427-9.
[39] Y. Luo, Z. Guo, Y. Sun, B. Plale, J. Qiu, and W. W. Li, "A hierarchical framework for cross-domain MapReduce execution," in *Proc. 2nd Int. Workshop Emerg. Comput. Methods Life Sci.* New York, NY, USA: ACM, Jun. 2011, pp. 15–22, doi: 10.1145/1996023.1996026.
[40] *Puma Dataset*. Accessed: Feb. 24, 2022. [Online]. Available: https://engineering.purdue.edu/puma/datasets.htm



EMIN SESEN received the B.Sc. and M.Sc. degrees in computer engineering from Düzce University, Düzce, Türkiye, in 2009 and 2017, respectively, where he is currently pursuing the Ph.D. degree, under the supervision of Resul Kara. He is also a Lecturer with the IT Department, Düzce University. His research interests include big data, computer networks, and network security.



SERDAR KIRISOGLU received the B.Sc. degree in electrical-electronics engineering from Kahramanmaraş Sütçü İmam University, Türkiye, in 2005, and the M.Sc. and Ph.D. degrees in electrical-electronics and computer engineering from Düzce University, Düzce, Türkiye, in 2009 and 2018, respectively. He is currently an Assistant Professor with the Department of Computer Engineering, Faculty of Engineering, Düzce University. His research interests include computer networks, machine learning, and deep learning.



RESUL KARA received the B.Sc. and M.Sc. degrees in electrical-electronics engineering from Gazi University, Ankara, Türkiye, in 1997 and 1999, respectively, and the Ph.D. degree from Sakarya University, Sakarya, Türkiye, in 2009. He is currently a Professor with the Department of Computer Engineering, Faculty of Engineering, Düzce University, Düzce, Türkiye. His research interests include computer networks, network security, wireless networks, and the Internet of Things.