Cross-MapReduce: Data Transfer Reduction in Geo-Distributed MapReduce

Saeed Mirpour Marzuni
Department of Computer Engineering
Faculty of Engineering
Ferdowsi University of Mashhad
Mashhad, Iran
mirpour@um.ac.ir

Abdorreza Savadi
Department of Computer Engineering
Faculty of Engineering
Ferdowsi University of Mashhad
Mashhad, Iran
savadi@um.ac.ir

Adel N. Toosi
Faculty of Information Technology
Monash University
Clayton, Australia
adel.n.toosi@monash.edu

Mahmoud Naghibzadeh
Department of Computer Engineering
Faculty of Engineering
Ferdowsi University of Mashhad
Mashhad, Iran
naghibzadeh@um.ac.ir

***Abstract*— The MapReduce model is widely used to store and process big data in a distributed manner. MapReduce was originally developed for a single tightly coupled cluster of computers. Approaches such as Hierarchical and Geo-Hadoop are designed to address geo-distributed MapReduce processing. However, these methods still suffer from high inter-cluster data transfer over the Internet, which is prohibitive for processing today's globally big data. In line with our thinking that there is no need to transfer the entire intermediate results to a single global reducer, we propose Cross-MapReduce, a framework for geo-distributed MapReduce processing. Before any massive data transfer, our proposed method finds a set of best global reducers to minimize transferred data volumes. We propose a graph called *Global Reduction Graph* (GRG) to determine the number and the location of global reducers. We conduct extensive experimental evaluations using a real testbed to demonstrate the effectiveness of Cross-MapReduce. The experimental results show that Cross-MapReduce significantly outperforms the Hierarchical and Geo-Hadoop approaches and reduces the amount of data transfer over the Internet by 40%.**

Keywords*— *MapReduce, Geo-distributed, Data Center, Big Data.

I. INTRODUCTION

With the dramatic increase in the size of collected and stored data that is known as big data, the need for building data-driven applications for analyzing large datasets becomes increasingly essential in many areas of science and business. The current Internet-based applications such as Internet of Things (IoT), smart cities, and social networks produce huge amount of data and the processing has to be fast. The input data for many of these applications are often distributed in different locations. Moreover, in many cases, the geo-distributed data is generated at even much higher speed compared to the actual data transfer speed [2,3], for example, data from modern satellites [4].

There are three common reasons for having geo-distributed data: (*i*) many organizations work in different countries and create local data in different parts of the world; (*ii*) organizations may prefer to use multi-clouds to enhance their reliability, security, and processing [5,6]; (iii) data is often stored close to where it is produced and needs to be processed in other locations, for example, sensor data is stored close to the sensors and needs to be processed in the cloud infrastructure.

The ability to analyze and process geo-distributed data has become an important and challenging mission in many domains. Many applications need to process and analyze a massive amount of geo-distributed data [18]. For example, a bioinformatics application that analyzes existing genomes in different laboratories, a smart surveillance application that analyzes video feeds from distributed cameras, a monitoring system that inspects log files from distributed servers, or a social networking application that finds common friends of its users.

Processing massive amount of data can be done best by running many parallel tasks operating on various parts of the dataset. Several frameworks have been proposed for big data processing, for example, Hadoop [20], Spark [21], Storm [27] and Flink [28]. Thus, in this paper, we focus on the MapReduce programming model, a well-accepted model for big data processing. The traditional frameworks supporting MapReduce (e.g., Hadoop and Spark) are not designed to process geo-distributed data. For instance, Telegram servers are spread worldwide or Facebook maintains a growing number of data centers across the world and both of them use the MapReduce for processing their batch data [30,31]. In practice, a naïve solution of gathering all raw data into a single cluster to process geo-distributed data is used, which is not scalable. In such a naïve solution, data transfer between clusters can become a bottleneck. Moreover, it is also unreasonable to move the raw data to a single location when the output results of the computation in each cluster is smaller than its input data [7,8,9]. Thus, two other approaches to facilitate geo-distributed MapReduce are proposed in the literature which we call them *Hierarchical* and *Geo-Hadoop* approaches [22].

The Hierarchical and Geo-Hadoop approaches are far from perfect since they require a large amount of data transfer over the Internet. In the Hierarchical approach, each cluster processes data independently, then the entire results are transmitted to a single cluster (global reducer), and the final process is executed on a single global reducer. This approach requires a significant amount of data to be transferred to a single cluster. In the Geo-Hadoop approach, all required inter-cluster transfers are performed in the shuffle phase of the MapReduce process. It is needless to say that in the geo-distributed MapReduce the inter-data center data transfer is much slower than the data transfer among the cluster nodes of a single cluster. Therefore, this approach, in particular, increases the processing time for many applications whose intermediate results are more than the final results. For example, the *invertedindex* application with an input data of 1.4 GB generates 4.5 GB intermediate data, as shown in [9].

The use of frameworks which support only the original MapReduce model do not provide acceptable performance for processing geo-distributed data in multiple data centers. The MapReduce model needs to be extended in order to provide appropriate solutions for processing data scattered across multiple data centers. Therefore, in this paper, we aim to tackle this issue and address the research problem of "how to reduce the total data transfer over the Internet in processing big data volumes scattered over multiple geographically distributed data centers?" We develop Cross-MapReduce to answer three important questions: (i) How many clusters should be selected as global reducers and which clusters are selected? (ii) What fraction of the results will be sent to the global reducers? (iii) What are the best parameters for selecting a global reducer?

Cross-MapReduce is inspired by the integration of Hierarchical and Geo-Hadoop approaches to reduce inter-cluster data transfers. Cross-MapReduce runs jobs in each cluster independently, similar to the Hierarchical approach. In the next step, instead of transferring all the results to a single cluster, like Geo-Hadoop approach, it shuffles the results that are required between clusters. The primary purpose of Cross-MapReduce is to cover the weaknesses of both Hierarchical and Geo-Hadoop approaches. Moreover, Cross-MapReduce is a framework-independent approach that can work with any other frameworks supporting MapReduce such as Spark and Hadoop. In fact, Cross-MapReduce is a framework which manages the several clusters each capable of supporting MapReduce.

Our **key contributions** in the Cross-MapReduce framework are as follows:

- *Gshuffling*: We present a novel process called Gshuffling to distinguish between inter-cluster traffic over the Internet and intra-cluster traffic within the cluster. In MapReduce jobs, the volume of intermediate data is often greater than or equal to the volume of the final results. Thus, in Cross-MapReduce, the data transfer in the shuffle phase of MapReduce is divided into two phases. The first phase is between nodes of each cluster (intra-cluster), which is performed independently within each MapReduce cluster. The second phase that includes the inter-cluster transfer over the Internet which is performed via Gshuffling. Gshuffling finds multiple global reducers in a way that the amount of data transfer between clusters is reduced.

- *GRG*: In order to transfer the required data between clusters, as part of Gshuffling process, we propose and build a novel graph called Global Reduction Graph (GRG). GRG represents the required inter-cluster data transfer and determines the number and the location of global reducers. For the subsequent reduce cycles, instead of transferring the entire results, Cross-MapReduce identifies the portion of the results which is required by the global reducers.

- *Load balancing*: We propose a new load balancing algorithm to increase performance and spread tasks among clusters. All the existing *Hierarchical* methods select a single global reducer for the final processing. However, Cross-MapReduce selects multiple global reducers to reduce overall data transfer and balance it between global reducers.

The rest of the paper is organized as follows: Section 2 describes the MapReduce programming model. In the next section, we discuss the problem tackled in this research. The Cross-MapReduce framework is proposed in Section 4. Section 5 presents the experimental results. Section 6 covers the study of existing methods and related work, and the final section concludes the work.

II. BACKGROUND

MapReduce is one of the most commonly used programming models for big data processing. In the MapReduce model (Figure 1), data transfer is needed in two phases: *map* and *reduce*. The map function receives the key-value input pairs and generates a list of key-value intermediate pairs. Then the reduce function is run, which integrates all values with the same key. The output results of map tasks are the inputs of the reduce tasks. The input data is divided into input splits, and a map task processes each split. After the completion of one the map task, the shuffle phase is started, and the required data for the reduce task is moved to the reducer nodes. The reduce tasks are started when all map tasks are completed, while the shuffling phase can be overlapped with the mapping tasks.

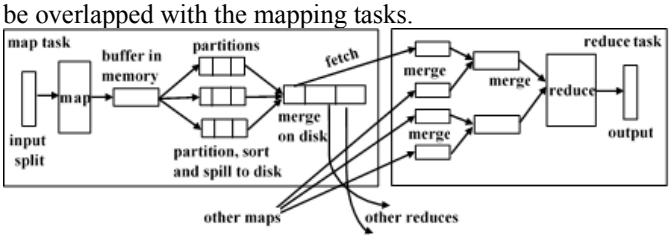


Fig. 1 The MapReduce model [1]

The processing of map tasks includes *read*, *map*, *collect*, *spill*, and *merge*. Each map task processes a logical piece of input data located on a distributed file system. Data is split into blocks of the same size (the default block size is 64 or 128 MB) and are distributed to cluster nodes. The map task reads a block of data and runs the map function (the code written by the user) on each record. Output results are stored in the main memory. If the volume of output results (intermediate data) is greater than the buffer, then it is written to the local disk which is known as spill. Before data is written to the disk, a thread divides the data into partitions. For each partition, a reduce task is created. A reduce task needs particular partitions from several map tasks across the cluster.

A reduce task includes *shuffle*, *reduce*, and *write* steps. In the shuffle stage, reduce tasks fetch the intermediate data from completed map tasks. The fetched intermediate data from all map tasks are sorted and merged in this stage. The reduce function is then executed on the merged data. Finally, the reduce phase output data is written to the distributed file system in the write step.

There is another element in the MapReduce programming model: *combiners*. Combiners allow for the local aggregation.

They are "mini-reducers" that take place on the output of the mappers, prior to the shuffle and sort phase. Each combiner operates in isolation and therefore does not have access to the intermediate output from other mappers. Our method extends the combiners in multi-cluster level.

III. PROBLEM STATEMENT

We assume that there are several MapReduce clusters connected through the Internet. Any framework supporting the MapReduce model can be set up on clusters. All clusters include one master node that the Cross-MapReduce communicates with it for running the desired commands.

In each cluster, there is a portion of the data that should be processed. For the sake of simplicity, in this paper, the clusters and network bandwidth are considered to be homogeneous. We also assume that clusters are point-to-point interconnected over the Internet. The bandwidth between clusters over the Internet is limited which can become a bottleneck of the system and increases the runtime.

As a pilot experiment, we ran the MapReduce model on different volumes of data and observed the results. In all observations, the amount of produced intermediate data by map tasks is much larger than the amount of final data produced by reducer tasks. This is reasonable because, in the MapReduce structure, a new key is not produced in the reduce phase, only the records that were produced in the map phase are merged. Therefore, it is obvious that the volume of intermediate data is practically always greater than or equal to the volume of the final results.

We distinguish between inter-cluster data transfer which happens over the Internet and intra-cluster data transfer which happens within each cluster. So, by proposing GShuffling, we postpone the inter-cluster data transfer to the time that MapReduce jobs are finished in each cluster. Instead of shuffling time, the data transfer between clusters happens after all reduce tasks are finished in all clusters. Using GShuffling, we expect that the volume of data transfer will be significantly reduced since the number of records transmitted between clusters is reduced.

A. Motivational Examples

In this section, we describe a very simple example to motivate the idea behind this work. The volume of data in this example is chosen to be small to be easily understood by the readers. However, in the performance evaluation section, the high-volume datasets are selected to evaluate the proposed method. In Figure 2, we consider 3 clusters, each of which independently processes its job. If we use the Hierarchical approach and select Cluster 1 as a global reducer, then the entire output of Clusters 2 and 3 are sent to Cluster 1 with a total of 12 records. But in Cross-MapReduce, the output of Cluster 3 is not transmitted at all, since in the production of the final result, there is no need for those keys. Only 4 overlapping records (Key_1, Key_3, Key_5, Key_6) from Cluster 2 are transmitted to Cluster 1.

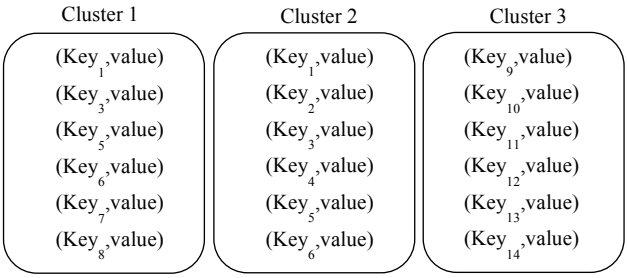


Fig. 2. An example of output results in each cluster

In Figure 3, we present another example to illustrate how data transfers can be reduced in the proposed Cross-MapReduce framework compared to the Geo-Hadoop approach. Suppose that the map phase is done and intermediate data is produced. As shown in Figure 3, there are two records with the key "key1" in two separate nodes in both clusters. The dashed lines represent the data transfer between nodes for the Geo-Hadoop approach. Now, we select one of the nodes as the reducer node. If Node N_1 in Cluster 1 is selected as the reducer, we find that it needs to read its data from three other nodes, so that there are two nodes in the other cluster. Thus, the number of records to be transferred between clusters is two. Now consider the solid lines representing the data transfer between nodes for Cross-MapReduce. In this model, records containing the key "key1" are combined together in each cluster; therefore, only one record is transmitted from Cluster 2 to Cluster 1. This means that inter-cluster data transfer will be reduced. Note that inter-cluster bandwidth is more limited (almost 60 times slower) than intra-cluster connections since traffic has to traverse through the internet. Cross-MapReduce and Geo-Hadoop both have three edges for merging but Cross-Hadoop has only one over the internet.

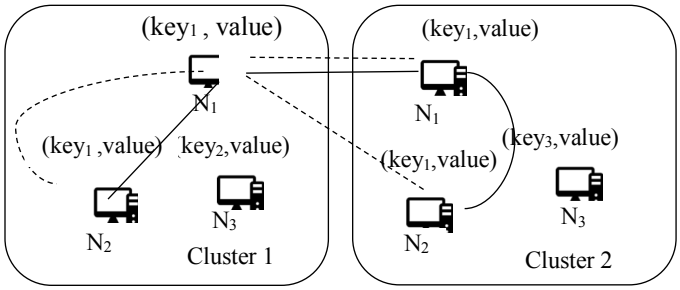


Fig. 3. Inter-cluster data transfer in the Cross-Hadoop and Geo-Hadoop approaches. Dashed lines show data transfer in Geo-Hadoop while solid lines show Cross-Hadoop data transfer.

Furthermore, to test our hypothesis, we prepared two purposely small files, *hadoop_tutorial* [1] and *mapreduce_tutorial*[2] and executed *wordcount* job on them. After processing, the *hadoop_tutorial* output file contained 2899 records and the *mapreduce_tutorial* output file contained 2821 records. By comparing two output files, it is found that there are only 202 records with the same keys which is interestingly small for two relevant topics. Therefore, according to our hypothesis, it is only necessary to transfer 202 records (2 KB) between clusters to generate final results. However, if we

[1] https://www.tutorialspoint.com/hadoop/hadoop_tutorial.pdf

[2] https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.pdf

produce the final results using the *Hierarchical* approach, 2821 records (36 KB) should be transferred (18 times more data). We also tested this for the *Geo-Hadoop* approach, which we found that it is necessary to transfer 8005 records (90 KB) in the shuffle phase (45 times more data). The example shows that we can save significant amount of data transfer if we only transfer common keys between clusters.

IV. CROSS-MAPREDUCE

In the following sections, we propose the Cross-MapReduce framework for the processing of geo-distributed data. In the beginning, Cross-MapReduce works similar to the Hierarchical approach, and all clusters independently run their jobs and produce their results. In the Geo-Hadoop approach, after generating intermediate data (output of map tasks), the shuffling phase occurs on all the nodes. However, in Cross-MapReduce, all the inter-cluster transfers happen after the job is completed in each cluster. Figure 4 shows the entire process of running a job in Cross-MapReduce, where MR_i represents the MapReduce job in the i^{th} cluster that processes the data independently, and GR_i shows the i^{th} global reducer.

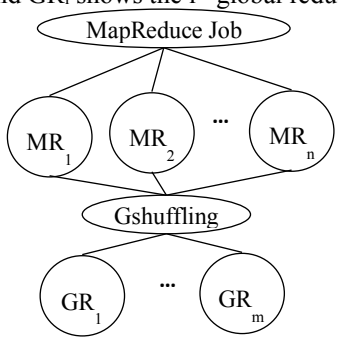


Fig. 4. The process of running a job in the Cross-Hadoop framework

The combiners in MapReduce merge the records that have the same key on a single machine in the cluster. The combined map result is transferred to reducers as one record. In Cross-MapReduce, we use the same concept at the cluster level and we call it MapCombine. Cross-MapReduce merges the records that have the same keys in a cluster using the reduce function. Thus, our proposed method follows these steps:

MapCombine -> Gshuffle -> GlobalReducer.

MapCombine runs user's jobs containing the map and reduce functions in each cluster. After running MapCombine it is expected that there would be unique keys in each cluster.

One of the main novelties of the Cross-MapReduce is Gshuffling, where a single global reducer in the Hierarchical approach is replaced with multi global reducers. We identify the portion of results that must be transferred by the early transfer of keys instead of the total intermediate results. Then, global reducers are determined by the construction of the Global Reduction Graph (GRG). Having built GRG, we send the required key-value pairs to the global reducers. GShuffling has two key advantages: (i) unlike the Hierarchical approach, only a portion of data that is required for processing is transmitted; (ii) the reduce task is executed on local values of the same key in each cluster prior to GShuffling. As a result, the data transfer between clusters is reduced compared to the Geo-Hadoop approach.

A. System Design

The proposed Cross-MapReduce system architecture is shown in Figure 5. It is a distributed multi-cluster system consisting of three layers. Cross-MapReduce architecture like SDN and NFV [34,35] architectures splits the control and data transferring elements. The SDN controller is a logically central entity that receives instructions or requirements from the application layer and relays them to the networking components. Similarly, in Cross-MapReduce architecture, the lowest layer includes MapReduce clusters in which the real data is saved. The second layer is a management and orchestration layer which includes GRG and Gshuffling components. In fact, Layer 2 is a software layer that determines the data transfer between clusters of Layer 3 and global reducers for a job. The first layer is the control layer which includes *JobManager* and *DataManager* components.

First, a user submits the job to JobManager. JobManager divides the job into sub-jobs and sends each one to the clusters in the third layer. In the third layer, data is processed independently in each cluster. The completion of each sub-job is reported to JobManager, including the address of the results which is recorded in the DataManager. Afterwards, DataManager selects a cluster whose key volume is the largest. This cluster is called *master*. In Layer 2, the master fetches the keys from all clusters. Then, in the master cluster, GRG is formed for the Gshuffling process. The required global reducers and their best placement are determined according to the GRG algorithm in Layer 2, which is explained in more details in the following section. Afterwards, a portion of the result that is needed is sent to the selected global reducers. Finally, the process is completed when the address for results of global reducers is added to the DataManager.

Algorithm 1 shows the steps of Cross-MapReduce. In Line 5-6, all clusters send their key set to the master cluster in parallel. Then, GRG is built on the master in Line 7. After running the GRG algorithm on the GRG Graph, global reducers are determined. Each cluster is requested to send the required key-values to the allocated global reducers. In fact, the master sends keys along with the target global reducer to clusters and asks them to send their key-value pairs to their global reducer. The input data for global reducers is collected in Line 10-11. Cross-MapReduce ensures that all the intermediate results (sub-job execution results) for a given key in multiple clusters is collected to one of the global reducers. More details are given when we discuss GRG in the next section. In Line 12, each global reducer runs a MapReduce job written by the user. Global reducers are run and generate the final results. Eventually, the addresses of results in global reducers are sent to DataManager. In Cross-MapReduce results are distributed among the clusters, and DataManager keeps track of the address for the specific key-value pair.

Algorithm 1: Cross-MapReduce

1: *submit a job to* ***JobManager****;*
2: *run a sub-job in each cluster on local data;*
3: *register results address along with their volume in* ***DataManager****;*
4: *select cluster* C_i *for* ***Gshuffling*** *process;*
5: *for cluster* C_j*;* $0 < j \leq$ *number of cluster and* $j \neq i$*, do in parallel*
6: C_j *sends the* ***key set*** *along with their number of values to the* C_i*;*
7: *create* ***GRG graph*** *in* C_i*;*
8: *run* ***GRG algorithm*** *on* ***GRG graph****;*
9: *Ci sends* ***required key*** *to other clusters;*
10: *for cluster Cj;* $0 < j \leq$ *number of cluster*
11: *Cj sends the* ***required key-value*** *to its* ***global reducer****;*
12: *run* ***global reducers****;*
13: *add new results address in* ***DataManager****;*
14: *end;*

B. GRG Graph

In this section, we describe how to form a graph to select global reducers. We create an undirected Global Reduction Graph (GRG) using all the keys in the *master*. The vertices of the graph represent clusters with intermediate results of each sub-job. GRG itself is created by running a MapReduce job. The input to this job is a key set, and its output is a list of key-value pairs with values representing the cluster number associated with the key. For instance, if key "A" exists in Clusters 1, 2, and 3, key "B" in Clusters 2 and 3, and key "C" in Clusters 2, 3 and 4, the output of the MapReduce job includes ("A", <1,2,3>), ("B", <2,3>), ("C", <2,3,4>) key-value pairs.

To build GRG, we scan through the output. We assume that the values in the list of clusters (e.g., <1,2,3>) for a specific key (e.g. "A") are sorted based on the cluster numbers. We do this to avoid the creation of any loop in the graph for that key. The first cluster in this list (e.g. 1 for "A" and 2 for "B") is selected as the target cluster for the key. Then, we create an edge between all other vertices containing the key to the target vertex in the graph. If the edge already exists between two vertices, the weight of the edge is increased by one unit. Note that the way that clusters are numbered would not affect the overall inter-cluster data transfer after GRG is formed.

The number of created edges impacts the selection process of global reducers. In fact, the edges represent the existence of the common keys among the clusters and their weight show the number of the common keys between two clusters. It means that the higher the weight, the more the number of common keys between two corresponding clusters. We define a weight coefficient, υ_i, to represent the ratio of the volume of values to the number of keys for every vertex. υ_i is calculated by Formula 1.

$$\upsilon_i = \frac{Volume\ of\ values\ in\ C_i}{Number\ of\ keys\ in\ C_i} \qquad (1)$$

where C_i represents the i^{th} cluster (or vertex).

We also define weight for vertices. Vertices weight are calculated based on the number of common keys and the volume of corresponding values. By using vertices weight, we estimate the data volume that should be exported from the vertex. Therefore, the algorithm selects the maximum weight as the first global reducer to keep the maximum data volume stagnant (not transferred). This way, Cross-MapReduce reduces data transferring. The weight of a vertex is calculated by Formula 2.

$$W_i = \sum_{i \neq j} \upsilon_j \times weight\ of < i.j > \qquad (2)$$

where W_i is the weight of the i^{th} vertex.

We use υ_i to incorporate the volume of value in the selection process of the global reducer. Accordingly, the cluster

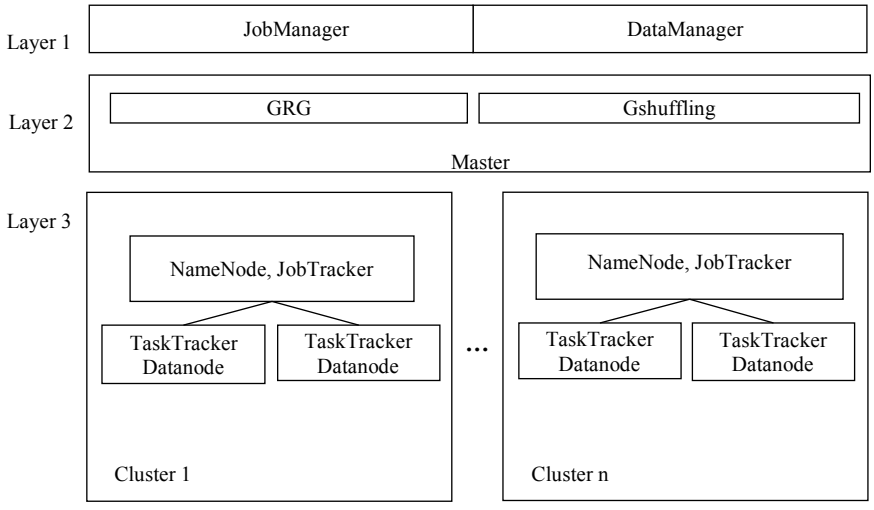


Fig. 5. System architecture

that has the highest weight keeps the key-values, and other clusters send the required key-values to this cluster. For example, consider the above list, we have keys "A", "B", and "C". For the key "A", the edge is created from Vertices 2 and 3 to 1. Similarly, an edge is created from 3 to 2 for "B", and 3 and 4 to 2 for "C".

Figure 6 shows the corresponding GRG for the above list. Note that edgeless GRG means no need to transfer data between the clusters.

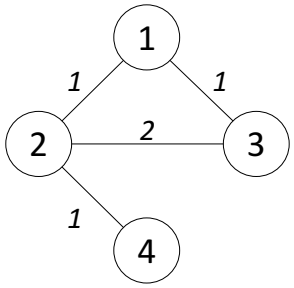


Fig. 6. Example of GRG graph

In order to determine global reducers, we use GRG algorithm. Algorithm 2 shows how global reducers are selected and load balancing between them is performed.

Algorithm 2: *GRG*

Input: GRG Graph
Output: List<global reducers, weight>

1: *Function GRG (GRG Graph)*
2: *NewVertex =* $\underset{vertex_i}{ArgMax}$ *(weight (vertex$_i$));*
3: *If (weight(NewVertex) == 0)*
4: *Return List;*
5: *If there exist several vertices with the same weight*
6: *Select one that has a higher weight in the original graph*
7: *List.Add(NewVertex, $W_{NewVertex}$);*
8: *If (List.lenght > 1)*
9: *If there exist edge between NewVertex and other member of List (V_i) in the original graph*
10: *For each edge <NewVertex, V_i>*
11: *D = $W_{NewVertex} - W_{Vi}$;*
12: *If D > weight of <NewVertex, V_i>*
13: *$W_{NewVertex} = W_{NewVertex}$ + weight of <NewVertex, V_i>;*
14: *$W_{vi} = W_{vi}$ – weight of <NewVertex, V_i>;*
15: *GRG Graph ← Remove adjacent edges of NewVertex;*
16: *GRG (GRG Graph);*
17: *End function;*

Algorithm 2 selects the global reducers based on the weight of the vertex. The highest weight vertex is selected as a global reducer for the key-value pairs of that vertex. These key-value pairs are kept in the global reducer and other key-values are sent to it. Since the weight of the vertex is obtained based on the volume of key-values, this action results in reduced inter-cluster data transfer. In Line 1, the algorithm selects a vertex that has the maximum weight. Then selected vertex is added to the list in Line 7. In Line 15, adjacent edges of the selected vertex are removed and the graph is updated. The new graph is passed to the GRG and all previous steps are run until there is no edge in the graph (Line 3). In Line 5-6, the algorithm gives priority to the vertices with the same weight. If there exist several vertices with the same weight in the GRG graph, the algorithm searches the maximum vertex weight between them in the original graph (the initial graph). If the vertices have the same weight in the original graph as well, then a vertex is selected arbitrarily. Lines 7 to 13 show the load balancing between the global reducers in each level. These steps are run when more than one global reducer is needed. In fact, we specify the direction of data movement between global reducers. Typically, from the global reducer with a lower weight to a global reducer with a higher one. In each iteration of the algorithm, a vertex is selected as a global reducer. Suppose that the algorithm is run in two iterations and two global reducers are selected (the first global reducer is selected in the first iteration and the second one is selected in the second one). To move data, in this case, the algorithm moves data from the second global reducer to the first one. However, in order to perform load balancing between global reducers, the algorithm can change the direction of data movement in each iteration. In Line 8, the algorithm checks whether the data is transferred between global reducers. If there exists an edge between global reducers, the difference between two global reducers weight is obtained (global reducer's weight is fetched from the graph in the corresponding iteration). Note that, global reducer's weight shows the number of key-values that is needed to process and the edge's weight also shows the number of key-values that is needed to transfer. After that, if the difference is higher than the edge weight between the two global reducers, then the direction of data movement is reversed. The algorithm's operation is explained using Figure 7. Figure 7 shows two global reducers, and the weight difference between them (gap). The goal is to reduce the gap between them. The data is moved from Vertex 2 to Vertex 1 typically. If the gap's value is higher than the edge weight (d), then edge weight is removed from first global reducer and is added to the second weight. This means the direction of data movement is reversed, and the gap value is reduced.

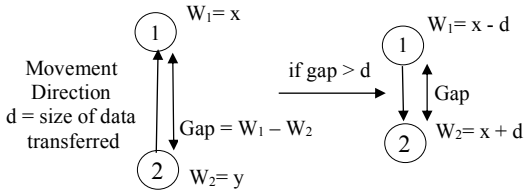


Fig. 7. Operation of load balancing

Consider the graph in Figure 6. For the sake of simplicity, we assume that $\upsilon_i = 1$ while in Algorithm 2 υ_i is calculated based on Formula 1. In this graph, Vertex 2 has the maximum weight of 4. The algorithm in the first iteration selects it and add it to list. Then adjacent edges of Vertex 2 are removed. Vertices 1, 3 and 4 include data transfer to Vertex 2 so far. In the next iteration, one edge remains in the graph between Vertices 1 and 3. Since the weight of these vertices are equal, so the algorithm searches the maximum weight in the original graph. In the original graph, Vertex 3 has the maximum weight; therefore, it is selected by the algorithm. Currently, Vertex 2 should process four key-values and Vertex 3 only process one key-value. Here, the algorithm performs the load balancing between Vertices 2 and 3. The weight of Vertices 2 and 3 are 4 and 1, respectively. Their weight difference is 3, that is higher than the edge weight 2. So the algorithm changes the direction

of data movement from Vertex 3 to 2. Therefore, the weight of Vertices 2 and 3 is updated, and the data is transferred from Vertex 2 to 3. Now, after the load balancing, Vertices 2 and 3 should process 2 and 3 key-values respectively.

Algorithm 2 is a recursive algorithm with polynomial time complexity. The maximum number of edges in the graph is n(n-1)/2, where n is the number of vertices (clusters). In each iteration, the algorithm removes one the edges. The algorithm finishes when there is no other edge in the graph. Therefore, the overall time complexity of Algorithm 2 is $O(n^2)$.

C. Global Reducer

In the final stage, some clusters are selected to run a job as global reducers by the GRG algorithm. Each global reducer cluster runs a MapReduce job. Two methods for the programming of the global reducer's job can be followed. The first method runs a MapReduce job like a sub-job with a do-nothing map function. Since the MapReduce job cannot run without map function, the map function only reads the input data and saves it as an intermediate key-value without any modification. Then the reduce function is run and generates the final results. By this way, jobs like GetAverage (calculate the average of numbers) cannot be executed by Cross-MapReduce. The second method runs a MapReduce job written by the user. In this method, the user has the responsibility to develop the task for the global reducers. This way, the scope of MapReduce jobs that can be executed by Cross-MapReduce is expanded and includes the non-associative or non-commutative reducer functions. An example can better explain the purpose of choosing this method. Consider two jobs: WordCount and GetAverage (calculating the average of several numbers). The first method generates the correct results in multi clusters for the WordCount job because the reducer functions are associative (each sub-job in WordCount sums the values of the common keys, and then the global reducers sum the common keys value.). However, for GetAverage job, sub-job calculates the average locally. So, the input of global reducers is the average values from multiple clusters. In this case, the first idea cannot generate the correct results. Therefore, in Cross-MapReduce, users submit two jobs, sub-job and global reducers job for non-associative jobs. Sub-job is run on each cluster locally, and finally the global reducers' job is run on the clusters determined by GRG.

V. Performance Evaluation

In this section, we conduct experiments to evaluate the effectiveness of the Cross-MapReduce in reducing data transfer among multiple geographically distributed clusters when MapReduce is used. We evaluate the performance of Cross-MapReduce using a testbed and compared it to the Hierarchical and Geo-Hadoop approaches. Here, we use four applications, *wordcount*, *invertedindex, adjacency-list* and *sql-query* with different ratios of the key to value volume. Adjacency-list [17] is similar to search-engine computation to generate adjacency and reverse adjacency lists of vertices of a graph to be used by PageRank-like algorithms. In *sql-query*, the log file of administrative data of a job scheduling mechanism is used. For confidentiality reasons, the algorithm and log file data are obfuscated. The log data has 8 columns of information:

- TaskId: a unique number identifies each task.
- TaskStatus: indicates that a task has been successfully completed.
- TaskExecutionTime: shows the execution time of the task.
- TaskDeadline: it shows the deadline of the task.
- TaskArriveTime: shows the time when a task was submitted.
- TaskStartTime: indicates the time when a task is run.
- TaskWaitTime: shows the time when a task waits to run.
- TaskType: it shows the type of the task.

We do data processing on the log data such as counting the number of task runs and obtaining the average of task execution time, and the average of TaskWaitTime where the TaskStatus is 'Success' in the log file. In *wordcount*, the ratio of the value volume is less than the key volume in contrast to others. In this paper, we focus on the data transfer size and job run time. We use Hadoop 2.6.5 for MapReduce processing in our experiments. We run our experiments on three clusters over three different hosts. Each node has 2 cores, 4 GB memory and 50 GB disk. To measure the bandwidth, we send 500 MB data from one host to another five times and calculate the bandwidth between the hosts. The average bandwidth between each host is 1.37 MB/s.

For *wordcount, adjacency-list and invertedindex,* we use PUMA [17] dataset for our evaluation. Our clusters are point-to-point interconnected, and all clusters and bandwidth are homogeneous. Table 1 shows the existing data volume in each cluster. Note that, due to limitations in academic research laboratories, running experiments in big scales (e.g., petabyte and exabyte scale) is not feasible. However, this does not imply that the Cross-MapReduce cannot be executed on big data scales. The proposed method can be easily applied for big data scale and reduce inter-cluster data transferring. As noted in section IV-B, the order of algorithm is $O(n^2)$ and the number of GRG vertices (clusters) will be practically small in real cases. In sql-query, our data is distributed over three clusters in each data about 5 GB (Table 1 shows the amount of data exactly). Our query results are the number of tasks runs, the average of task execution and task waiting time where the TaskStatus is 'Success'.

We compared Cross-MapReduce with the Hierarchical and Geo-Hadoop approaches in terms of inter-cluster data transfer and makespan. First, we examine the volume of the key set and the final output results for all applications, shown in Figures 8, 9, 10 and 11 for every three clusters. According to Figure 8, 10 and 11 in *invertedindex, sql-query* and *Adjacency-list*, the volume of the key set is almost less than the half of the volume of output results. Whereas, as it can be seen in Figure 9, in *wordcount*, the volume of the key set is very large in comparison with the volume of output results, so that almost 90% of the results is made up of keys.

In both Cross-MapReduce and Hierarchical approaches, each cluster independently runs its own application. After running MapReduce on each cluster and producing intermediate results in the Hierarchical approach, one cluster is selected as a global reducer and results of the other clusters are sent to this cluster. Here, we select the cluster whose volume of intermediate results is larger than the other clusters. The reason is that the size of data transfer between clusters in the Hierarchical approach is minimized this way. In Cross-MapReduce, instead of sending the total intermediate results to a single cluster, we only send keys to the master cluster in order to build GRG. Then we determine clusters that should run global reducers, and only part of the data that is required is sent to global reducers. The amount of exported data from each cluster is shown in Figures 12, 13, 14 and 15 for each application.

According to experiments, Geo-Hadoop has the highest data transfer between clusters since the applications' intermediate data (output of map tasks) size is very large compared to the raw data. Figure 12 shows the size of the exported data for the *invertedindex* application. As shown in Figure 18, the volume of intermediate results in cluster 1 is higher than in other clusters. Therefore, for the Hierarchical approach, Cluster 1 is selected as the individual global reducer, and the results of the other clusters are transferred to it. Hence, the size of the exported data for the Hierarchical approach in Cluster 3 is zero in Figure 12. But in Cross-MapReduce, clusters send their keys to Cluster 1. After processing and forming GRG, Cluster 3 is selected as the first global reducer, so the exported data from Cluster 3 only contains the keys that are sent from the master. Cluster 1 is then selected as the second global reducer. The size of the exported data in Cluster 1 consists of the data that is required by the first global reducer and the master. And in Cluster 2, the amount of exported data is equal to the total volume of keys and the data that is required by other clusters.

Figure 13 shows the same process for the *wordcount* application. Since in *wordcount*, the volume of the key set is very large compared to the volume of the output, hence the amount of exported data in Cross-MapReduce, in Cluster 1 and Cluster 2, is greater than the Hierarchical approach and only Cluster 3 is slightly less than the other one. In *wordcount*, Cluster 1 is chosen as a global reducer for the Hierarchical approach, so the size of the exported data in this cluster is zero. As shown in Figure 10, the size of results in cluster 3 is larger than other clusters in *sql-query*. So, cluster 3 is selected as the global reducer in Hierarchical approach and results of other clusters are transferred to it. Thus, the size of the exported data for the Hierarchical approach in Cluster 3 is zero in Figure 15. In Cross-MapReduce, Cluster 3 is selected as the master where GRG is created. Then Cluster 1 is selected as the first global reducer, and Cluster 3 is the second one. As shown in Figure 11, the result size of all clusters is almost equal. However, the result size of Cluster 1 is slightly more, so Cluster 1 is selected as a global reducer in *Adjacency-list* for Hierarchical approaches. Other Clusters transfer their result to Cluster 1. Therefore, the total size of inter-cluster data transferring is 6.5 GB for the Hierarchical approach. In Cross-MapReduce Cluster 1 is selected as the master. Finally, Cluster 1 and 3 are selected as global reducers. The summary of the total data transfers in all approaches for each application is shown in Table 2.

Table 1. The cluster data volume for each application

	wordcount	**invertedindex**	**Sql-query**	**Adjacency-List**
Cluster 1	4.6 G	4.6 G	5.2 G	3.1 G
Cluster 2	2.8 G	2.8 G	5.1 G	3.1 G
Cluster 3	4.3 G	4.3 G	5.2 G	3.2 G

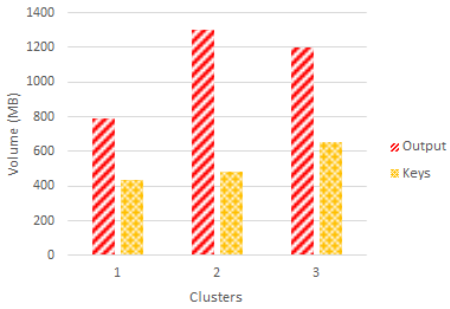


Fig. 8. The volume of key set and output results in *invertedindex*

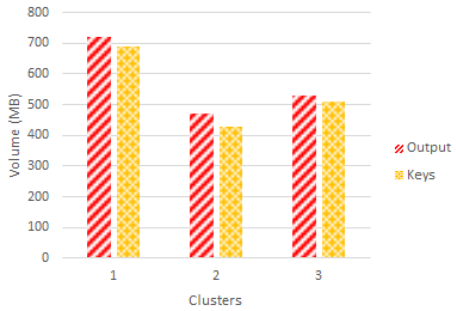


Fig. 9. The volume of key set and output results in *wordcount*

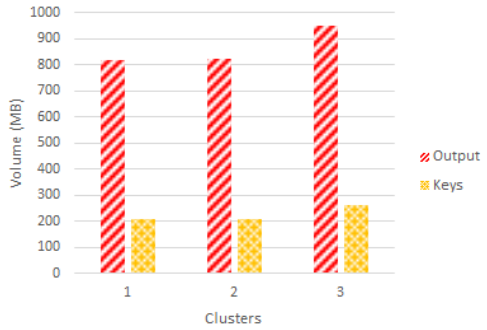


Fig. 10. The volume of key set and output results in *Sql-query*

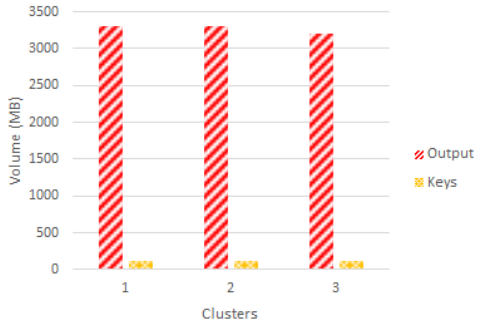


Fig. 11. The volume of key set and output results in *Adjacency-List*

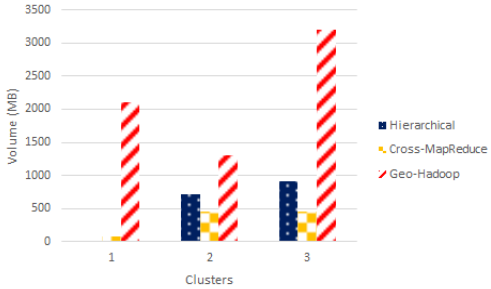


Fig. 12. The amount of exported data from clusters in *invertedindex*

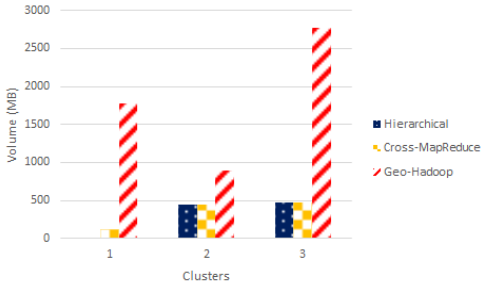


Fig. 13. The amount of exported data from clusters in *wordcount*

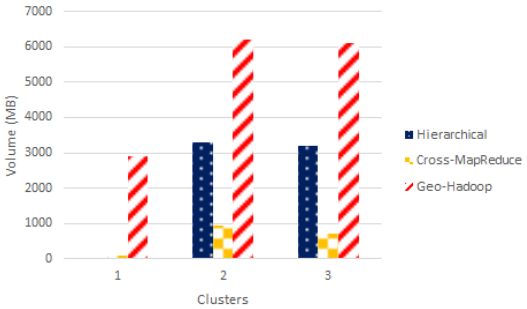


Fig. 14. The amount of exported data from clusters in *Adjacency-List*

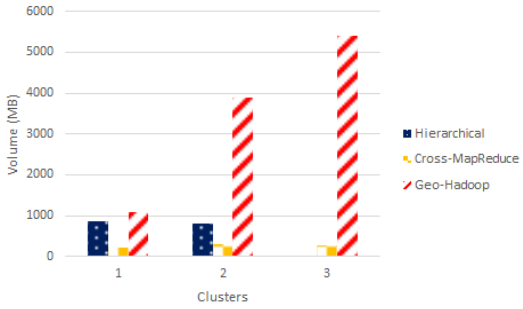


Fig. 15. The amount of exported data from clusters in *Sql-query*

Table 2. Compression of Cross-MapReduce, Hierarchical and Geo-Hadoop approaches

	Make-span				Inter-cluster transfer			
	Adjacency-List	***Sql-query***	***Invertedindex***	***Wordcount***	***Adjacency-List***	***Sql-query***	***Invertedindex***	***Wordcount***
Cross-MapReduce	**44.53 min**	**32.03** min	**61.38** min	32.1 min	**1775.3 MB**	**788.6** MB	**1080.3** MB	1020.9 MB
Hierarchical	80.8 min	42.81min	65.76 min	**30.41** min	6500 MB	1664.4 MB	1791.6 MB	**1008.3** MB
Geo-Hadoop	105.15 min	51.1min	73.40 min	37.2 min	15.2 GB	11.1 GB	6.6 GB	5.6 GB

In *invertedindex, Sql-query* and *Adjacency-list* the total data transfer by Cross-MapReduce is about 39%, 52% and 73% less than the Hierarchical approach, respectively, as shown in Table 2. But in *wordcount*, the size of data transfer among clusters increased by 2% in Cross-MapReduce. This is expected due to the volume of the key set. Cross-MapReduce is efficient in applications whose key volume is smaller than the value volume.

Table 2 also shows the makespan of execution of applications under three approaches. Due to the data transfer reduction in Cross-MapReduce, the makespan has significantly decreased for *Invertedindex, sql-query* and *Adjacency-list*. The important point at the *wordcount* makespan is that in spite of the slight increase in data transfer for Cross-MapReduce, its makespan is still very competitive to the Hierarchical approach. There are two reasons for this observation: (i) in the Hierarchical approach, the amount of processing data for the global reducer is considerably high compared to Cross-MapReduce. The total data volume to be processed for the global reducer is about 1.5 GB while in Cross-MapReduce, each global reducer processes less than 300 MB of data; (ii) in Cross-MapReduce, multiple clusters are selected as global reducers rather than a single global reducer, and the final processing is performed in parallel to produce the final results.

Cross-MapReduce is divided into nine phases as shown in Figure 16. This Figure shows the timespan of different phases of Cross-MapReduce for *wordcount*, *invertedindex, adjacency-list* and *sql-query* applications in details. In the first phase, each

cluster runs sub-job for the application. Then, in the Get Key phase, another job is run to extract keys from the sub-job results. Instead of sending the result to a single cluster, in the keys transferring phase, Cross-MapReduce sends keys to a single cluster (*master*). After that, it finds keys that are common among the clusters in the next phase (find common keys). Based on these common keys, GRG graph is created (create GRG) and all required keys are transferred (send keys) to acquire the corresponding values. Before receiving these values, a MapReduce job is run to extract the required values (getting required key-values). Then the global reducers receive the required key-values (receiving key-values) and in the last phase, all global reducers run their jobs. The timespan of each step is shown with a different color in Figure 16.

Our approach reduces the data transfer between clusters in the cost of more processing. As shown in Figure 16, in each application, the maximum time is used by running sub-jobs and transmission of keys. Figure 16 shows that the extra local processing time compared to the data transferring time is negligible, in Cross-MapReduce.

In the experiments that have been performed so far, the distribution of data keys is not considered, and the data is distributed among clusters randomly.

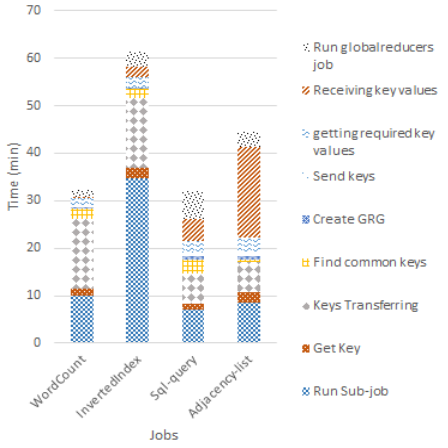


Fig. 16. Process of Cross-Hadoop in each application

We investigate the impact of the percentage of common keys among the clusters on the overall data transfer in the following. Since the *InvertedIndex, Adjacecy-list* and *WordCount* data are text documents, we only use *sql-query* data which is structured to control the distribution of common keys in these experiments.

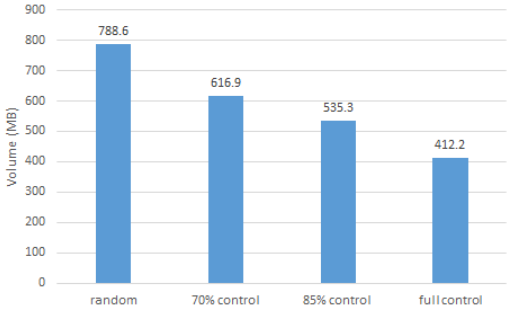
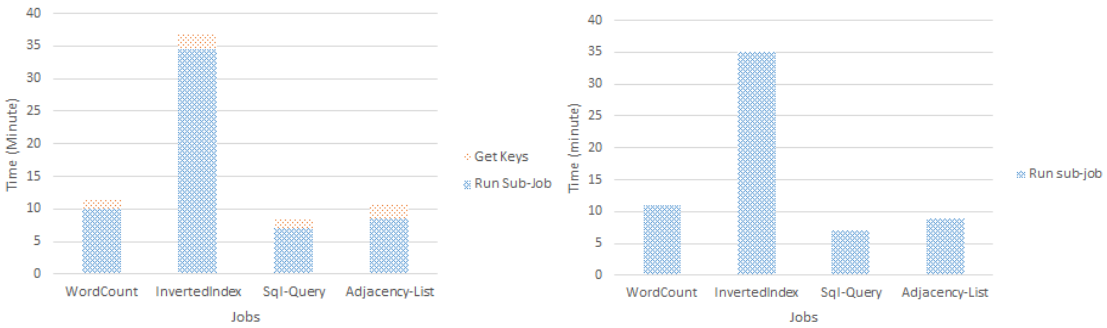


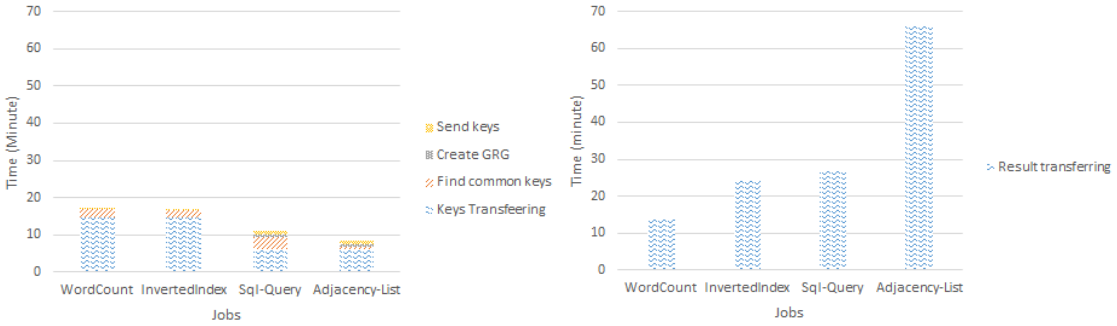
Fig. 17. Inter-cluster data transfer in the controlled distribution of keys.

Figure 17 shows the total inter-cluster data transferring among clusters in Cross-MapReduce in different distribution of common keys. As shown in Figure 17, four cases are considered. In the first case, keys are distributed randomly. In this case, the inter-cluster data transferring is 788.6 MB which is the highest. In the second case, the distribution of keys is controlled in a way that only 30% of keys are common among clusters. In this case, the inter-cluster data transferring is 616.9 MB, and Cross-MapReduce performs better than others. In the third case, only 15% of keys are common among clusters. The total inter-cluster data transferring is reduced to 535.3 MB. In the last case, the distribution of keys is fully controlled in a way that no common keys exist among clusters. Cross-MapReduce transferred 412.2 MB data among the clusters in this case. According to Figure 17, the data transfer reduction is linear because the data is structured, and the size of value for all keys are almost equal. As we expected, when the key distribution is controlled, Cross-MapReduce performs better as less inter-cluster data transfer is required.



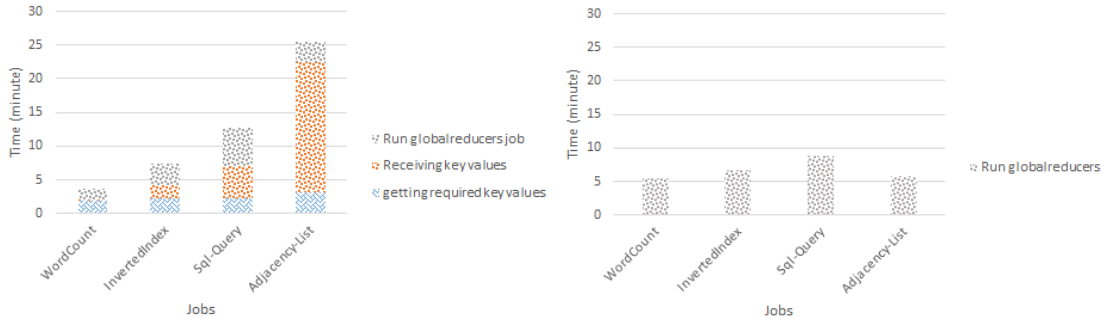
(a) Fisrt phase of Cross-MapReduce (b) First phase of Hierarchical

Figure 18: Comparison of Cross-MapReduce phase 1 with Hierarchical



(a) Second phase of Cross-MapReduce (b) Second phase of Hierarchical

Figure 19: Comparison of Cross-MapReduce phase 2 with Hierarchical



(a) Third phase of Cross-MapReduce (b) Third phase of Hierarchical

Figure 20: Comparison of Cross-MapReduce phase 3 with Hierarchical

In Figures 18 to 20, the runtime of *Hierarchical* and Cross-MapReduce are compared in details. There exists tree phases in *Hierarchical* approach, *Run sub-job, Result Transferring* and *Run global reducers job.* In fact, the first phase in *Hierarchical* is computation phase, the second phase is communication or data transferring and the third phase is a computation phase again. According to this, we divide Cross-MapReduce into three phases and compare it with the Hirarchical approach.

Figure 18 (a) shows the first phase including run sub-job and Get Keys jobs for the Cross-MapReduce approach. In contrast, Figure 18 (b) shows the first phase of Hierarchical approach that includes the run sub-job. In the first phase, Hierarchical approach finishes the job earlier in all applications because it does not need to run the GetKey job. While in phase 2 (Figure 19), although Cross-MapReduce runs more jobs, it finishes them earlier than the Hierarchical approach. This is due to the fact that Cross-MapReduce reduces the inter-cluster data transfer significatly at the cost of a higher computation volume. Figure 20 shows the phase three for Hierarchical and Cross-MapReduce approaches. In this phase, Cross-MapReduce transfers the required keys-values and then runs the global reducer job. Hence, in this phase, Cross-MapReduce runtime is more than the Hierarchical approach. But the data volume that should be transferred is not considerable. Because Cross-MapReduce only transfers the data that is required. In Cross-MapReduce, the runtime of Global reducer job is lower than that of the Hierarchical aproach. Because Cross-MapReduce selects multi global reducers and runs the global reducer job in parallel on multiple clusters. However, the Hierarchical overall.

A. Discussion

In this section, we formalize the possible performance gain in Cross-MapReduce. In general, three types of data transfer happen in Cross-MapReduce: 1) the collection of keys in the master, 2) the set of required keys sent from master to each cluster to inform them to transfer their data to the global reducers, and 3) key-value results that should be sent to the global reducers from the informed clusters. Cross-MapReduce would be deemed beneficial when its total data transfer is at least less than the Hierarchical approach. Let D denote the total data transfer in the Hierarchical approach. Equation 3 represents this line of thinking:

$$\sum_i \mathrm{Key}_i + \sum_i \mathrm{RequiredKey}_i + \sum_i \mathrm{KeyValue}_i < \mathrm{D} \qquad (3)$$

where key_i presents the size of the key set in the i^{th} cluster. $RequiredKey_i$ shows the size of the key set sent to the i^{th} cluster from the master and $KeyValue_i$ is the size of the key-value set that are transferred from the i^{th} cluster to a global reducer as an input. For the sake of brevity, we rewrite Equation 3 as:

$$\mathrm{K} + \mathrm{R} + \mathrm{K}_{\mathrm{val}} < \mathrm{D} \qquad (4)$$

Let $D - V \approx K$, where V is the size of values in the final results. Here, we assume that the size of the final results is almost equal to the amount of data transfer for the Hierarchical approach. This is true to a large extent if the size of locally

produced intermediate results in the global reducer in the Hierarchical approach is relatively small. K_{val} is equal to $R+V_R$, where V_R stands for the size of the value set that is transferred to the global reducers in Cross-MapReduce. Thus, Equation 5 is obtained as follows:

$$\begin{aligned} & D - V + R + R + V_R < D \\ & \Rightarrow 2R - V + V_R < 0 \\ & \Rightarrow V > 2R + V_R \end{aligned} \quad (5)$$

According to Equation 5, when the volume of the value set in the final results is greater than twice size of the key set requested to be transferred to the global reducers plus their values, the use of Cross-MapReduce will be beneficial. This is the case for many applications whose values are considerably larger than their keys.

VI. RELATED WORK

In the data-driven application, communication among nodes is a critical element of the system performance. Many studies have been performed in a single cluster and schedule tasks to reduce data transfer and run-time. Chang [36] presents a framework based on MapReduce for analyzing weather data and simulating temperature distributions. The author first used MapReduce to forecast temperature of three cities in a period of over two years and demonstrated its accuracy. Then the paper illustrates an optimized eight-step process of MapReduce for visualizing temperature distribution. Deldari et al. [23] propose the algorithm that attempts to minimize the execution cost considering a user-defined deadline constraint. They divide tasks into a number of clusters, and then an extendable and flexible scoring approach chooses the best cluster combinations to achieve the algorithm's goals. Benelallam et al. [32] argue that model transformation with rule-based languages like AtlanMod Transformation Language (ATL) is a problem that fits to the MapReduce execution model. As a proof of concept, they introduce semantics for ATL distributed execution on MapReduce. Also, they propose a distributed engine model transformation with ATL on MapReduce [33]. They utilize MapReduce as a tool to execute ATL on a MapReduce cluster. In contrary, we focus on data transferring among multiple clusters in this paper. However, when data is distributed in multi-cluster, the inter-cluster data transfer can become a bottleneck. UniCrawl [26] is an efficient geo-distributed crawler that aims at the minimization of inter-site communication costs. UniCrawl uses multiple geographically distributed sites. Each site uses an independent crawler and relies on well-established techniques for fetching and parsing the content of the web. In general, there are three primary solutions to solve the problem of processing geo-distributed data using a MapReduce model [8][10]. (1) collecting all raw data from different clusters in a single cluster and process them locally; (2) run MapReduce task in an entirely distributed fashion on all clusters while data transfers and communications among inter-cluster nodes happen through the Internet, also known as Geo-Hadoop approach; (3) MapReduce processing is run independently at each cluster and results are aggregated in a Hierarchical fashion. The first solution for big data is not cost-effective. Therefore, in the following, we review related works based on Hierarchical and Geo-Hadoop approaches.

A. Geo-Hadoop Approach

Wang et al. [11] presented the G-Hadoop framework for processing geo-distributed data across multiple clusters without changing the architecture of the existing cluster. G-Hadoop stores data in a geo-distributed file system, known as Gfarm file system. The G-Hadoop framework contains a master node in a central location. The master node accepts jobs from the user, splits them into several sub-jobs and distributes them across the slave nodes. The Master node also manages all metadata files in the system. The master node contains a metadata server and a global *JobTracker*. The slave node contains a *TaskTracker*, a local *JobTracker*, and an I/O server.

Jayalath et al. [8] presented the G-MR which is a hadoop-based framework (See Figure 21). The G-MR runs MapReduce jobs across multiple data centers. Unlike G-Hadoop, G-MR does not place reducers randomly [13] and uses a single directional weighted graph for data movement using the shortest path algorithm. G-MR deploys a *GroupManager* at a single data center. In each data center there exist a *JobManager*. *GroupManager* distributes map and reduce codes to all data centers and executes a data transformation graph (DTG) algorithm. Using a Hadoop cluster, a *JobManager* manages and executes assigned local MapReduce jobs. Each *JobManager* has two components, namely a *CopyManager* for copying outputs of the job of a data center to other data centers and an *AggregationManager* for aggregating results from data centers.

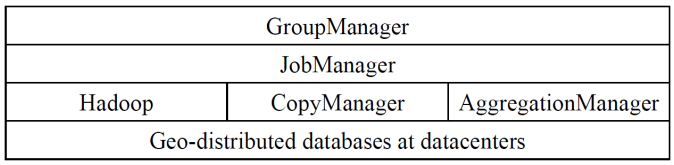


Fig. 21. G-MR

Heintz et al. [9] presented shuffle-aware data pushing at the map phase. In this method, they find all those mappers that affect the completion of the job in a DC, then reject those mappers that cause delay. In other words, they select mappers which can execute a job and shuffle the intermediate data under a time constraint. Mappers are selected based on recent jobs monitoring. In this method, an algorithm is provided for a single data center that can be extended to a geo-distributed environment. Similarly, Resilin [24] provides a hybrid cloud-based MapReduce computation framework. Resilin, implements Amazon Elastic MapReduce (EMR) [25] interface and uses the existing Amazon EMR tools for interacting with the system. In particular, Resilin allows a user to process data stored in a cloud with the help of other clouds resources. Nithyanantham and Singaravel [19] present Multivariate Metaphor based Meta-Heuristic Glowworm Swarm Map-Reduce Optimization (MM-MGSMO) for processing massive data in GDDC by selecting resource and cost-optimized virtual machines. The selection of resource and cost-optimized virtual machine results in the minimization of workload between data

centers. A multi objective functions were defined for each virtual machine in terms of bandwidth, storage capacity, energy and computation cost. Finally, with the aid of the MapReduce function, the optimal virtual machine was identified using the mapping and in turn, allocated the big data to the selected optimal virtual machine. Wang et al. [29] propose task scheduling with deadlines and data locality to save energy consumption in MapReduce clusters with a variable total number of slots. In each heartbeat, a new job sequence is generated in order to better meet deadline constraints and a new assignment among tasks and slots is produced to increase data locality.

B. *Hierarchical Approach*

Hierarchical MapReduce (HMR) [12] is a two-level programming model (See Figure 22), so that the upper level is a global controller layer and the lower layer consists of several clusters that execute MapReduce jobs. HMR processes data separately in each cluster and then a single global reducer collects all the results generated in other clusters. Finally, the global reducer is executed and final result is generated.

A simple extension to HMR is proposed in [14], where the authors suggested to consider the amount of data to be moved and the resources required to produce the final output at the global reducer. However, like HMR, this extension does not consider heterogeneous inter-DC bandwidth and available resources at the clusters [18]. Another extension is provided in [10], where the authors consider the availability of clusters' resources and different network link capacities. Cavallo et al. [22] focused on the data fragmentation technique as a way to improve the performance of the scheduling system. They designed many distributed computing scenarios, both balanced and imbalanced, and for each scenario they analyzed the performance of several type of jobs by applying many data fragmentation schemes. They aim to distribute data on clusters so that performance is improved, while in this paper the data is distributed and Cross-MapReduce should process the data so that the inter-cluster data transferring is reduced. Medusa [15] handles three new types of faults: processing corruption that leads to wrong outputs, malicious attacks and power outages that may lead to the unavailability of MapReduce instances and their data. A job is executed on $2f + 1$ clouds to handle faults in a way that f faults are tolerable. In addition, a cloud is selected based on parameters such as available resources and bandwidth so that the job completion time is decreased [18]. Chrysaor [16] is based on a fine-grained replication scheme that tolerates faults at the task level. It modifies the user code and does not change the Hadoop framework. It consists of two phases: first, the MapReduce job is run in each cluster, second, the global MapReduce job is run to aggregate the results of all clusters to produce a final result.

In this paper, we presented a novel solution aimed at rectifying the weaknesses of Hierarchical and Geo-Hadoop solutions. One of the problems in the Hierarchical solution is that all outputs of clusters should be sent to a single cluster and then the final results are produced there. While a large portion of the data that is sent are not be needed to produce final results. So, in Cross-MapReduce framework, we proposed the Global Reduction Graph (GRG) to answer the following three important questions: (i) How many clusters are selected as global reducers and which clusters are selected? (ii) What fraction of the results will be sent to the global reducers? (iii) What are the best parameters for selecting a global reducer? On the other hand, one of the main problems in the Geo-Hadoop approach is that when the volume of intermediate results is greater than the input data, a high volume of data transfer is required.

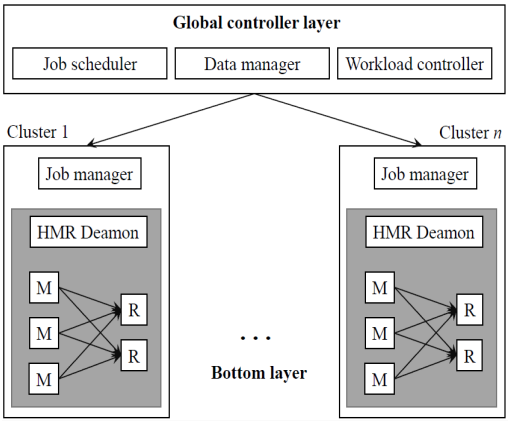


Fig. 22. HMR [18]

VII. Conclusions and Future Work

Many applications require data to be read and processed by multiple data centers, since the data is produced and stored in a distributed fashion. One of the main challenges in the processing of distributed data in multiple data centers is that the data transfer between data centers significantly affects the processing time. Another problem of a distributed big data processing is that frameworks such as Hadoop and Spark are not designed to support multi-clusters. Therefore, novel solutions are required to process such data. In this paper, we proposed Cross-MapReduce based on the MapReduce model and the combination of Hierarchical and Geo-Hadoop approaches. Our novel solution reduces the inter-cluster data transfers compared to the existing common approaches for the execution of MapReduce over geo-distributed data. We argued that for most applications, there is no need to send the entire intermediate results to a global reducer especially when the volume of keys is less than the volume of the value. Therefore, by only transferring keys instead of transferring entire datasets, we created a graph to determine the global reducers and the portion of data that is required for processing in global reducers. We chose *Wordcount*, *Invertedindex* and *Sql-query* applications to compare our proposed approach with the Hierarchical and Geo-Hadoop methods. These applications are different in the size of key and value volumes and are selected with the purpose of showing their impact on the proposed Cross-MapReduce. We conducted experiments on real clusters. Results show that Cross-MapReduce is remarkably effective in cases where the key set volume is less than the value volume. It also reduces the

amount of data transfer and makespan by 40% and 23% respectively.

In this paper, the bandwidth between the clusters and their computational power are assumed to be homogeneous. One of the future work is that heterogeneous clusters and bandwidth are considered. One of the other challenges in this area is privacy and the geolocation of the sensitive data. Therefore, in future, we will also investigate privacy and legislation/policy awareness in the proposed method. Besides, we are interested in adapting our proposed method to applications with deadline-constraints. Cross-MapReduce relies on the performance of underlying MapReduce frameworks. A future direction can be the optimization of these frameworks.

REFERENCES

[1] T. White, "Hadoop: The Definitive Guide", Fourth edition O'Reilly Media, Inc., 2015.

[2] A. Vulimiri, C. Curino, B. Godfrey, K. Karanasos, and G. Varghese, "WANalytics : Analytics for a Geo-Distributed Data-Intensive World." ACM SIGMOD International Conference on Management of Data, pp. 1087-1092, 2015.

[3] D. A. Reed and J. Dongarra, "Exascale computing and big data," *Commun. ACM*, vol. 58, no. 7, pp. 56–68, 2015.

[4] K. A. Hawick, P. D. Coddington, and H. A. James, *Distributed frameworks and parallel algorithms for processing large-scale geographic data*, vol. 29, no. 10. 2003.

[5] K. Kloudas, M. Mamede, and N. Preguic, "Pixida : Optimizing Data Parallel Jobs in Wide-Area Data Analytics," pp. 72–83, 2014.

[6] http://www.computerworld.com/article/2834193/cloud-computing/5-tips-for-building-a-successful-hybrid-cloud.html.

[7] M. Cardosa, C. Wang, A. Nangia, A. Chandra, and J. Weissman, "Exploring MapReduce efficiency with highly-distributed data," *Proc. Second Int. Work. MapReduce its Appl. - MapReduce '11*, p. 27, 2011.

[8] 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, 2014.

[9] 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, 2016.

[10] M. Cavallo, G. Di Modica, C. Polito, and O. Tomarchio, "H2F: A Hierarchical Hadoop Framework for big data processing in geo-distributed environments," *Proc. - 3rd IEEE/ACM Int. Conf. Big Data Comput. Appl. Technol. BDCAT 2016*, pp. 27–35, 2016.

[11] L. Wang, J. Tao, R. Ranjan and H. Marten, "G-Hadoop: MapReduce across distributed data centers for data-intensive computing," *Futur. Gener. Comput. Syst.*, vol. 29, no. 3, pp. 739–750, 2013.

[12] Y. Luo and B. Plale, "Hierarchical MapReduce programming model and scheduling algorithms," *Proc. - 12th IEEE/ACM Int. Symp. Clust. Cloud Grid Comput. CCGrid 2012*, pp. 769–774, 2012.

[13] J. Zhang, L. Zhang, H. Huang, Z. L. Jiang, and X. Wang, "Key based data analytics across data centers considering bi-level resource provision in cloud computing," *Futur. Gener. Comput. Syst.*, vol. 62, pp. 40–50, 2016.

[14] M. Cavallo, G. Di Modica, C. Polito, and O. Tomarchio, "Application profiling in Hierarchical Hadoop for geo-distributed computing environments," *Proc. - IEEE Symp. Comput. Commun.*, vol. 2016–Augus, pp. 555–560, 2016.

[15] P. A. R. S. Costa, X. Bai, F. M. V. Ramos, and M. Correia, "Medusa: An Efficient Cloud Fault-Tolerant MapReduce," *Proc. - 2016 16th IEEE/ACM Int. Symp. Clust. Cloud, Grid Comput. CCGrid 2016*, pp. 443–452, 2016.

[16] P. A. R. S. Costa, F. M. V. Ramos, and M. Correia, "Chrysaor: Fine-Grained, Fault-Tolerant Cloud-of-Clouds MapReduce," *Proc. - 2017 17th IEEE/ACM Int. Symp. Clust. Cloud Grid Comput. CCGRID 2017*, pp. 421–430, 2017.

[17] PUMA: Purdue mapreduce benchmark suite. https://engineering.purdue.edu/~puma/

[18] S. Dolev, P. Florissi, E. Gudes, S. Sharma, and I. Singer, "A Survey on Geographically Distributed Big-Data Processing using MapReduce," vol. 7790, no. c, 2017.

[19] S. nithyanantham and G. Singaravel, "Resource and Cost Aware Glowworm Mapreduce Optimization Based Big Data Processing in Geo Distributed Data Center" in Wireless Personal Communications, 2020.

[20] Apache Hadoop. Available at: http://hadoop.apache.org/

[21] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker and I. Stoica, "Spark: Cluster computing with working sets," in HotCloud, 2010.

[22] M. Cavallo, G. Di Modica, C. Polito, O. Tomarchio, "Fragmenting Big Data to boost the performance of MapReduce in geographical computing contexts", *International Conference on Big Data Innovations and Applications,* 2017.

[23] A. Deldari, M. Naghibzadeh, S. Abrishami, "CCA: a deadline-constrained workflow scheduling algorithm for multicore resources on the cloud", Journal of Supercomputing, 2016.

[24] A. Iordache, C. Morin, N. Parlavantzas, E. Feller, and P. Riteau, "Resilin: Elastic MapReduce over multiple clouds," in CCGrid, pp. 261–268, 2013.

[25] Amozon Elastic MapReduce. Available at: http://aws.amazon.com/elasticmapreduce/.

[26] D. L. Quoc, C. Fetzer, P. Feiber, E. Rivier, V. Schiavoni and P. Sutra, "UniCrawl: A Practical Geographically Distributed Web Crawler", IEEE 8th International Conference on Cloud Computing, 2015.

[27] Apache Storm. Available at: http://storm.apache.org/.

[28] Apache Flink. Available at: https://flink.apache.org/.

[29] Jia Wang, Xiaoping Li, Rub´en Ruiz, Jie Yang and Dianhui Chu, "Energy Utilization Task Scheduling for MapReduce in Heterogeneous Clusters", *IEEE, Transactions on Services Computing,* 2020.

[30] M. Traverso, "Presto: Interacting with petabytes of data at facebook. https://www.facebook.com/notes/facebook-engineering/presto-interacting-with-petabytes-of-data-at-facebook/10151786197628920/

[31] Telegram, https://www.telegram.org.

[32] A. Benelallam, A. Gomez, M.Tisi, J. Cabot, "Distributed model-to-model transformation with ATL on MapReduce", *International Conference on Software Language Engineering*, 2015.

[33] A. Benelallam, A. Gomez, M. Tisi, J. Cabot, "Distributing Relational Model Transformation on MapReduce", *Journal of Systems & Software*, 2018.

[34] C. Kuo, V. Chang, C. Lei, "A feasibility analysis for edge computing fusion in LPWA IoT environment with SDN structure", *International Conference on Engineering and Technology (ICET),* 2017.

[35] G. Sun, Z. Xu, H. Yu, X. Chen, V. Chang, A. V. Vasilakos, "Low-latency and Resource-efficient Service Function Chaining Orchestration in Network Function Virtualization", *IEEE Internet of Things Journal*, 2019.

[36] V. Chang, "Towards data analysis for weather cloud computing", Knowl. Based Syst, 2017.