IJICTR
International Journal of Information & Communication Technology Research

Volume 14- Number 4 – 2022 (28 -35)

Collaborative Mappers based on Co-evolutionary Optimization Technique in MapReduce

Mohammad Reza Ahmadi*
ICT Research Institute (ITRC)
Tehran, Iran
m.ahmadi@itrc.ac.ir

Davood Maleki
ICT Research Institute (ITRC)
Tehran, Iran
dmaleki@itrc.ac.ir

Received: 9 May 2022 – Revised: 9 July 2022 - Accepted: 20 August 2022

Abstract—**MapReduce algorithm inspired by the map and reduces functions commonly used in functional programming. The use of this model is more beneficial when optimization of the distributed mappers in the MapReduce framework comes into the account. In standard mappers, each mapper operates independently and has no collaborative function or content relationship with other mappers. We propose a new technique to improve performance of the inter-processing tasks in MapReduce functions. In the proposed method, the mappers are connected and collaborated through a shared coordinator with a distributed metadata store called DMDS. In this new structure, a parallel and co-evolutionary genetic algorithm has been used to optimize and match the matrix processes simultaneously. The proposed method uses a genetic algorithm with a parallel and evolutionary executive structure in the mapping process of the mappers program to allocate resources, transfer and store data. The co-evolutionary MapReduce mappers can simplify and optimize relational data processing in the large clusters. MapReduce using a co-evolutionary mapper, provide successful convergence and better performance. Our experimental evaluation shows that collaborative techniques improves performance especially in the big size computations, and dramatically improves processing time across the MapReduce process. Even though the execution time in MapReduce varies with data volume, in the proposed method the overhead processing in low volume data is considerable where in high volume data shows more competitive advantage. In fact, with increasing the data volume, advantage of the proposed method becomes more considerable.**

Keywords: MapReduce; Co-evolutionary; Cooperative; Distributed metadata store; DMDS.

Article type: Research Article

Publisher: ICT Research Institute

[DOI: 10.52547/itrc.14.4.28]

I. INTRODUCTION

MapReduce is a software programming and associated implementation for processing and generating data sets with a parallel and distributed algorithm on a cluster system. A MapReduce program is composed of two main procedures, a map procedure for filtering, sorting, and a reduce method, which performs a summary operation. For optimization process, GA or Genetic algorithm is an evolutionary computation algorithm which can solve many optimization problems and belongs to the larger class of evolutionary algorithms, which generate solutions to optimization problems using techniques inspired by natural evolution, such as

* Corresponding Author

inheritance, mutation, selection, and crossover [1] [2]. In this paper we take advantage of genetic algorithm to solve the problem of mapper relations and propose a new method to enhance the performance of MapReduce processes. In fact, MapReduce is a useful technic in a wide range of applications, including distributed pattern-based searching, distributed sorting, web link-graph reversal, singular value decomposition [3], web access log stats, inverted index construction, document clustering, machine learning [4], and statistical machine translation. The MapReduce System orchestrates the processing by marshaling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance. MapReduce algorithm inspired by the map and reduces functions commonly used in functional programming. The use of this model is more beneficial when the optimized distributed shuffle operation and fault tolerance features of the MapReduce framework comes into account [5]. This paper proposes an optimization method and evaluates the modified MapReduce algorithm to improve data transfers during the Shuffle phase under the bandwidth constraints. Communication cost is an essential factor in optimization for a good MapReduce algorithm. The map function is applied to each input record (i.e., key/value pair) and produces a list of intermediate records. The reduce function is applied to each group of intermediate records with the same key, and produces a list of output records [6]. Thus, we have focused on task distribution, resource optimization and execution time that enhance the Mappers utilization and reduce operational costs. Variant parameters that have influence on final decision have been considered as input to the allocation algorithm and their impact have been evaluated in different test scenarios. We have organized the paper as follows. In section 2, related work has been presented. In section 3, the proposed mappers architecture and components. In section 4, performance evaluation. Finally, a conclusion has been described in sec.5.

II. Related Work

Parallel sort and join algorithms for large datasets like MapReduce, have been widely studied since the early 1980’s [7, 24]. To gain more flexibility, new MapReduce-inspired massive data processing platforms have emerged: Dryad[8], Hyracks[9], Spark[10] – all include elements of MapReduce, but have more choices in runtime query execution. In contrast to these projects, some chose to enhance MapReduce, to leverage existing investment in the Hadoop framework and in the query processing systems built on top of it, such as Pig[11], and Hive[12]. The MapReduce paradigm has gained a lot of attention in academia and industry [13]. There are many MapReduce tools, three of which that is Hadoop [14], Apache HIVE [12], and Sqoop [15] are the most important. Each of these open source platforms has a specific MapReduce mechanism [16]. Evolution techniques studies of MapReduce are always turned on and tune themselves based on different data. A comprehensive study of MapReduce and its application in optimization algorithms is in [17].

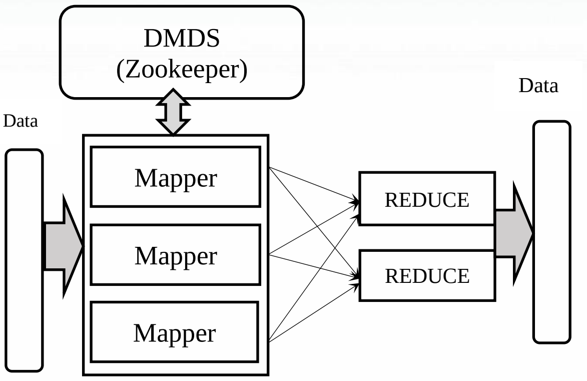

Figure 1. Positions Adaptive MapReduce with DMDS

Reference [18] has comparing Apache Spark and MapReduce with performance analysis using K-Means. The study in [19] focused on Hadoop MapReduce framework in Big Data analytics. In [20], the authors have a comparative study of classification algorithms based on MapReduce model. The Map-Reduce-Merge implemented in [21], simplified relational data processing on large clusters. A number of techniques have been proposed to improve the performance of MapReduce jobs. The study in [22] focused on grouping MapReduce jobs that perform common computations and evaluating each group as a single job. Some these studies are complementary to our study and can be used in our framework to improve the performance even further. Dryad [8] goes one step further by allowing modification of the dataflow graph once a task is finished. Dryad, as well as Hadoop, has considered techniques to direct multiple input partitions to a single task. However, all of these techniques need to be setup statically before the job starts. Hence, in the general case, they cannot balance the workload as efficiently as adaptive mappers. Adaptive aggregation algorithms have been studied in parallel shared-nothing architectures [23] as well as in multi-core architectures. In [23], the authors propose a set of parallel aggregation algorithms that dynamically adapt at run-time based on the observed selectivity of the data. Adaptive MapReduce consists of the components: DMDS, SAMs, AMs, ACs, AS and AP [16, 25].

Input data is divided into splits and the split location information is stored in DMDS. DMDS is Zookeeper that is distributed coordination service in Fig. 1. Similar to this method is also used in this paper. Here, we use a DMDS storage that is a shared distribution coordinating storage service between mappers. Hadoop MapReduce is a popular implementation that works with the Hadoop Distributed File System (HDFS). Matlab program provides a slightly different implementation of the MapReduce technique with the MapReduce function. The MapReduce uses a Data Store to process data in small chunks that individually fit into memory. Each chunk goes through a Map phase, which formats the data for processing. Then the intermediate data chunks go through a Reduce phase, which aggregates the intermediate results to produce a final result. The Map and Reduce phases are encoded by map and reduce functions, which are primary inputs to MapReduce. There are endless combinations of map and reduce functions to process data, so this technique is both flexible and extremely powerful for tackling large data processing tasks. Various methods mentioned above are different MapReduce with different mechanisms, some of which are like Hadoop for general applications where there are a number of studies and researches in which they have tried to increase the efficiency of the algorithm. Reference [16] uses the adaptive method and the coordination between the elements of the MapReduce. We've propose a collaborative technic and we've added genetic algorithms to optimize and increase the performance with better results. We have shown that it is possible to make MapReduce more flexible and collaborative by breaking a key assumption of the programming model where the mappers are completely independent. We introduce an asynchronous communication channel between mappers, by using a transactional, distributed meta-data store (DMDS) or coordinator. This enables the mappers to post some metadata about their state and see the state of all other mappers.

III. Proposed Mappers Architecture and Components

MapReduce architecture with parallelizing genetic algorithms (MRPGA) has been shown in Fig. 2 [6].

In MRPGA method, the partitioning part splits the data and performs mutation. Then it sends the data to the master for evaluation and selection. The master splits the data into *m* pieces in MapReduce for the map task. The value of *m* is the maximum number of parallel map tasks in this architecture. Each piece of data is sent to a mapper in a MapReduce. The mapper is repeated for each individual in the piece of input to execute the map function for each task, and then it generates the results by the map function which they are stored locally. After the map phase process, the reduction phase is performed. In the reduction process, the assignment tasks have been performed. Finally, in the final phase of reduce operations; the result has been obtained. We have develop mapping process using collaborative technique in MapReduce section, and with the proposed methods based on GA and optimization mechanisms, we succeed to improve the efficiency of the MapReduce process.

A. *Input Parameters*

The input parameters for the MapReduce function include three categories:

1- Input parameters of mappers: five different parameters have been considered as the input for mappers which are shown in Table 1.

- CPU running speed in Mapper, which is usually in Gigahertz.

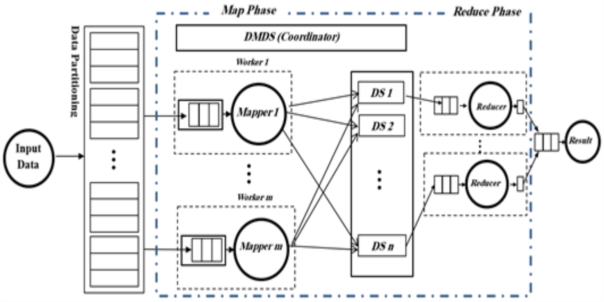

Figure 2. MapReduce architecture with parallelizing genetic algorithms (MRPGA)

TABLE I. Input Parameters of Mappers

No.	Parameter	Components	Description
1	CPU	CPU-cycle	CPU clock speed(GHZ)
2	Core	Number of CPUs	Deterministic
3	A	Amount of data	Amount of data produced by Mapper i
4	#MPs	Number of MPs	Number of MPs on each MR
5	Cache	Cache-CAP	Cache Capacity (MB)

TABLE II. Input Parameters of DataStors

No.	Parameter	Components	Description
1	RAM	RAM-Access time	RAM access speed (ms)
2	V	Volume of data to transfer	Volume of data to transfer to DSi
3	Class	Class of DS	Class of the DSi
4	#DS	Number of DSs	Number of DSs on each MR

- The number of processor (cores) per mapper that is directly related to the speed of process execution and reduces the execution time.
- The amount of data generated (A) by the mapper. The smaller the number and amount of data, the faster the mapper.
- The number of Maps (#MPs) in a MapReduce program (MR) that the higher the number of Maps, the faster the MapReduce program would be.
- The capacity of cache or cache memory per Mapper, the higher the cache, the faster the mapper program will run.

2- Input parameters of data stores: four parameters are the data storage input parameters in the MapReduce function as are shown in Table 2.

- Access speed with RAM, usually in milliseconds. The higher the speed of access to memory, the greater the execution of the program and the storage of data generated by the execution of the map.
- The amount and volume of data (V) that is transferred to the data storage. The more this data is, the more time it takes to store it.
- The storage class or technology class, the newer and higher the storage class and technology class, the better and faster the storage will be.
- The number of storage devices (#DS) in each MapReduce program, the higher the number, the faster the storage will be.

3- Input parameters of Network: the following parameters are the network input parameters between mappers and storage devices in MapReduce as are shown in Table 3.

- Network bandwidth (BW), the greater the network bandwidth between a mapper and a storage device, the faster the data is transferred.

[DOI: 10.52547/itrc.14.4.28]

TABLE III. INPUT PARAMETERS OF NETWORK

No.	Parameter	Components	Description
1	BW	NET-BW	Network bandwidth
2	SP	NET-SP	Network speed

- Network speed (SP), the higher the network speed between a mapper and a storage device, the faster the data will be sent and stored.

B. *Chromosome Representation*

In genetic algorithms, a chromosome is a set of entity or a string of data which defines a proposed solution to the problem that the genetic algorithm is trying to solve. The set of all solutions is known as the population. In computation, we assume several chromosomes in population based on variety of input items. In our assumption the chromosome size is equal to the number of mappers (MPs) with M members. M has an integer value. To map the number of mappers (M) to number of Data Stores (N), each chromosome encodes a scheduler which holds M×N genes to represent the placement of the MPs on the DSs. Thus, the value in the i'th gene states the MPs number that resides on j'th DS. As an example in Fig. 3, the selected item in each mapper resides in the related chromosome where the total member of the similar members resides in the related DS. The advantage of the GA appears where similar members in different mappers compete for placement in one step where in the standard model should wait for the coming steps. In Fig. 3, the MP_1 resides on DS_1, MP_2 reside on DS_j, and MPM resides on DS_x that x index is smaller or equal N.

MP_1					MP_2					MP_M			
D_1	D_2	D_3	…	D_N	D_1	D_2	…	D_N	…	D_1	D_2	…	D_N

DS_1	DS_j	…	DS_x

Figure 3. Chromosome dimensions

C. *Fitness Function*

We consider a fitness function which is used in genetic algorithms to guide simulations towards optimal design solutions. We assume the fitness function as follows:

$$F = \sum_{i=1}^{M} \sum_{j=1}^{N} C_{ij} X_{ij} \tag{1}$$

The algorithm try to maximize the function F where:

- i is number of gene state in MPs,
- j is number of gene state in DS,
- M is number of MPs,
- N is number of DSs,

$$Xij = \begin{cases} 1 & \text{if } i^{th} \text{ MPs Data is assigned to } j^{th} \text{ DSs} \\ 0 & \text{if } i^{th} \text{ MPs Data is not assigned to } j^{th} \text{ DSs} \end{cases} \tag{2}$$

$$C_{ij} = \sum_{k=1}^{nl} W_k P_k \tag{3}$$

$$P_k = (\{CPU \parallel Core \parallel A \parallel \# MPs \parallel Cache\}, \{BW \parallel SP\}, \{RAM \parallel V \parallel Class \parallel \# DS\}) \tag{4}$$

Where "nl" in equation 3, depicts the number of linguistic parameters considered in our fitness function. These parameters are selected from Table 1 for Mappers, Table 2 for DataStors and Table 3 for Networks. P_k is score values for each of these parameters and W_k is their weights. It should be noted that, if all the parameters have the same weights, then W_k in equation 3 is equal to 1. Besides, DS_k and MP_k are the resource value available on DS and the resource value demanded by the MP, respectively. Resources are listed in Table 1.

D. *Constraints*

We consider several constrain so that every MPs can inhibits multi DSs to Number of CPU cores. Maximum relation every MPs with DSs is Number of the CPU cores of the MPs (Eq.5). But every DSs inhibits on only one MPs. (Eq.6, Eq.7)

$$\sum_{j=1}^{N} X_{ij} \le C_i \qquad i = 1,2,\ldots,M \tag{5}$$

$$\sum_{i=1}^{M} X_{ij} \le 1 \qquad j = 1,2,\ldots,N \tag{6}$$

$$\sum_{i=1}^{M} \sum_{j=1}^{N} X_{ij} \le N \tag{7}$$

We consider eleven parameters as we have stated in Tables 1, 2 and 3; which includes CPU, CPU-core, Amount of Data, Bandwidth Net, Speed-Net, RAM access time, Volume of data transfer, Class of DS, Number of mappers and Data stores, and Cache of mappers.

IV. PERFORMANCE EVALUATION

Fig 4 show a sample test of the overall MapReduce counting process. In this example, input data is divided into three separate groups during the splitting stage.

In the input node, nine key words with four different key values of 001A, 002B, 003C, and 004D which are representation of data file index with different payload. Output of this node splits the key words in three categories for three mappers. In the shuffling phase, the data is transferred so that all the keys with the same key value are sent to the same nodes or the same data store.

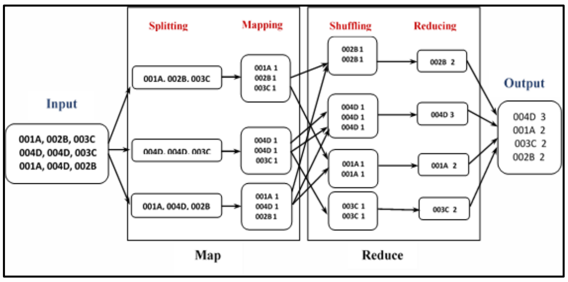

Figure 4. The overall MapReduce word count process.

[DOI: 10.52547/itrc.14.4.28]

TABLE IV. RESOURCE VALUES FOR MAPPERS (MPS) AND DATA STORES (DSS)

BW	DS1	DS2	DS3	DS4
MP1	1	1	1	1
MP2	2	1	2	2
MP3	1	1	2	2

SP	DS1	DS2	DS3	DS4
MP1	**1**	**1**	**1**	**1**
MP2	**2**	**2**	**2**	**2**
MP3	**3**	**3**	**3**	**3**

Mapper	CPU	Core	#A
MP1	**2**	**3**	**3**
MP2	**1**	**3**	**3**
MP3	**3**	**4**	**3**

Data store	RAM	V	Class
DS1	1	1	2
DS2	2	1	3
DS3	3	1	2
DS4	4	1	3

Finally, the reduction process aggregates all similar key value and the total values of the similar key words are stored in the output block.

One of the limitations in this example is generating three codes of 001A, 004D and 001A in the first line of the three-mapping block. When the system decides to transfer 001A code to shuffling group that located in the third box of the shuffling box, two vectors of 001A and 001A should transfer simultaneously; but, the system transfer one vector and postpone the second transfer for the next coming step. If similar problem accrues in the next step, the same procedure will happen and data transfer should postpone for the next cycle. This accumulation of similar tasks cause degradation in system efficiency where in the proposed method based on genetic algorithm, the system has ability to overcome to this limitation and improves the system efficiency.

In continue, we have considered the MapReduce model in Fig 4, as the testing scenario and using genetic algorithm based on Fig 2. Input parameters are applied to three mappers and four data store using the proposed genetic algorithm. The parameters values are shown in Table 4.

A. Evaluation Parameters

To evaluate the proposed method we have considered performance, efficiency and utilization as the three measurements. The first measurement is the process of collecting, analyzing and reporting information regarding the performance of system. The second is the ability to avoid wasting energy, efforts, and time in doing functionality or in producing an expected result. The last one refers to utilization factor or use factor that is the ratio of the time that system is in use to the total time that it could be in use.

In this section we have compared the MapReduce standard model with proposed DMDS model to compare standard Mappers and Cooperative co-evolutionary [26] adaptive Mappers. Evaluation is based on proposed parameters in Table 4 and the proposed model in Fig. 2.

1) Performance Evaluation in Standard Model:

In order to evaluate the efficiency of MapReduce in the standard model, the scenario has been implemented in Matlab simulator program according to the standard architecture model in Fig. 4. The details of the scenario are based on description in Section 3 and 4. The goal is to evaluate the process of word counting problem. In this example, we have considered 9 input words that are divided into three groups with three members between three mappers. Indeed, the reason to consider this example is to show how functioning the processes in a MapReduce. The desired parameters for selecting mappers are based on information in Table 4. Selected parameters include bandwidth, network speed, current status of the map processor in terms of processor speed and memory, and finally the current state of the storage in terms of speed and amount of data. The functionality evaluation has been performed using the main standard method of the MapReduce program shows in Table 5. We define the main parameters as follows:

- Resource utilization rate (RU)
- Performance (PE)
- Efficiency (E)

After running the program, we calculate the results as shown in Table 5, RU rate or utilization of the MapReduce program is about 56%, PE is about 56% and E is 20.86.

Utilization=9/16= 56%

The number 9 means that map processors have been used only 9 times to count words in the standard method. The number 12 means that Map processors have been busy 12 times. As shown in Table 5, only 9 out of 12 houses are filled

PE = Resource utilization (CPU usage and bandwidth facilities, bandwidth) - Overhead processing costs Added programs Algorithm Genetics Program.

Performance=56% - 0=56%

The value of zero 0 in the above formula indicates that we do not have additional overhead in the standard method.

E= Performance × Effectiveness Efficiency=56*(32+49+35+33)/4)= %56*149/4=20.86

TABLE V. STEPS PERFORMED USING THE STANDARD METHOD

Steps				Effectiveness
Step 1	MP1->DS3 C13=16	MP2->DS2 C22=16		F=32
Step 2	MP1->DS1 C11=13	MP2->DS2 C22=15	MP3->DS3 C33=21	F=49
Step 3	MP1->DS4 C14=16	MP3->DS2 C32=19		F=35
Step 4	MP2->DS1 C24=17	MP3->DS1 C31=16		F=33

[DOI: 10.52547/itrc.14.4.28]

Efficiency value indicates the amount of impact and average productivity. According to Objective Table 5, it shows the amount of productivity and impact in each stage.

Appendix A shows more details about the values and calculation method.

2) *Performance Evaluation in DMDS Model:*

The proposed scenario for simulation in the Matlab environment is based on the proposed architecture in Fig. 2. The rest of the information is similar to the previous model based on information in Section 4. The purpose of this scenario is to evaluate a counting problem. In this example, we have 9 input words that are divided into three groups with three members between three mappers. The parameters in the new method are the same as previous scenario. We select the maps to run according to Table 4. These parameters include bandwidth, network speed, current status of the processor in terms of processor speed and memory, and current state of the storage in terms of speed and amount of data. In this scenario, a genetic algorithm has been used. The way to create chromosomes using the genetic algorithm is shown in Section 3. Each chromosome is equal to the number of mappers multiplied by the number of reduce in binary mode. The execution and calculations in the genetic algorithm are in accordance with the main formulas 1 and 3 mentioned in the above section. After running the program, results of the proposed method with genetic algorithm is shown in Table 6. As shown in this table, the Resource Utilization of the MapReduce program for this especial example that utilize all resources is about 100%, Performance is about 80% and Efficiency is 39.73.

RU= 16/16= 100%

PE=100% - 20%=80%

Efficiency = 80%×((51+52+46)/3)=80%×149/3=39.73

Appendix B shows more details about the values and calculation method.

TABLE VI. Steps of Performed Using the Co-Evolutionary GA Method

Step 1	MP1->DS1 C11=14	MP2->DS2 C22=16	MP3->DS3 C33=21	F=51
Step 2	MP1->DS3 C13=15	MP2->DS4 C24=18	MP3->DS2 C32=19	F=52
Step 3	MP1->DS4 C14=16	MP2->DS2 C22=14	MP3->DS1 C31=16	F=46

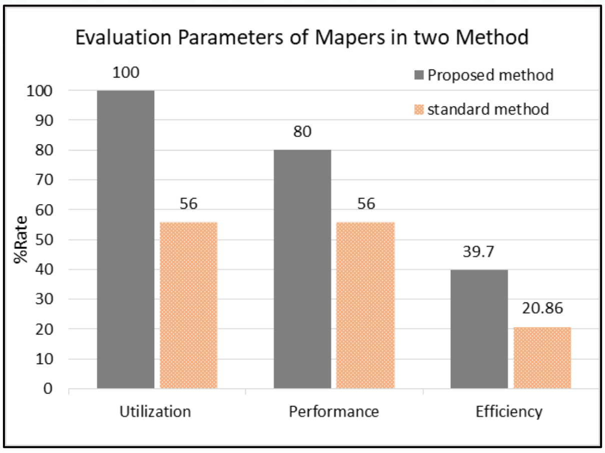

Figure 5. Comparison of the Mappers' Performances.

TABLE VII. Avg. Execute Time in Different Dataset in Standard MapReduce and Co-Evolutionary Proposed Method

Dataset size	Average Times (ms) in Standard MapReduce Method	Average Times (ms) in Co-evolutionary proposed Method
8 MB	50	60
64 MB	59	63
512 MB	110	69
1024 MB	110	70

Fig. 5, compares the utilization, performance and efficiency of the mappers in the MapReduce program for the above example in the standard and proposed methods. In continue, we compare the standard MapReduce with the Co-evolutionary proposed Method regarding the execution time for different data set volume.

Table 7 shows the approximate execution time for data with different sizes in both standard and collaborative proposed method. As shown in the table, at first the genetic algorithm creates additional overhead in the processing time, this overhead increases with data volume size. But, the number of processing steps is reduced and this advantage also affects the execution time. As a result, when increases the data volume, an equilibrium relationship between the overhead and the number of steps has been raised in the process. Thus, as shown in Table 7, by increasing the data volume, we reach to an almost constant execution time. This is fixed threshold level that is much lower in the proposed method rather than the standard method. The result also show that there is noticeable advantage regarding the average time in the proposed method, and this advantage increases with enlarging the data volume. According to the results, the larger the data volume, the shorter the execution time in accordance with standard MapReduce model.

Fig 6, shows comparison of processing time in standard Mappers and Co-evolutionary collaborative [27] Mappers in different data size. As it is shown, the capabilities and advantages of the proposed method are more pronounced in big data and the effect of the proposed method is more obvious.

[DOI: 10.52547/itrc.14.4.28]

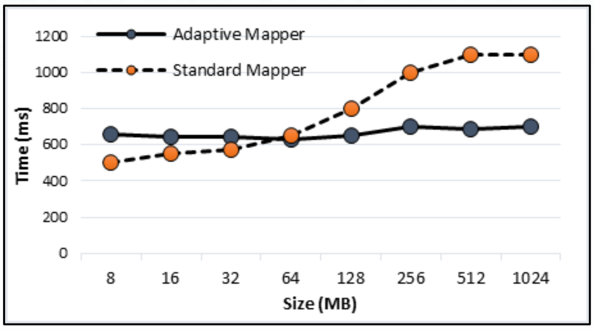

Figure 6. Comparison of processing time in Standard and collaborative Mappers.

The results show that in low data volume, both the proposed and standard methods are very close together, but with increasing data volume, this behavior changes. Furthermore the increase of data size shows more advantages in the proposed method. The important issue in this process is to compare execution time of the processes in different data volume in both methods to reach to a relative stability; where the time level of stability in the proposed method is considerably lower. As a result, the level of time stability is lower than the standard method and this is a very important advantage for processing time in the large data volume.

As it is shown in fig 6, from one hand, the effect of genetic algorithm appears in the overhead of execution process and increases in big data volume. On the other hand, the collaboration between the mappers reduces the number of steps to reach the final result and the effect of this advantage is much greater than the overhead, especially in large data volumes. As a result, using Co-evolutionary method in MapReduce functions reduces the average execution time of the process.

V. Conclusion

The purpose of this paper is to provide a new way to improve resource utilization and optimize the processing time in MapReduce program. In standard mappers, each mapper operates independently and has no functional or content relationship with other mappers. On the other hand in the proposed method, the mappers have mutual relationship and collaborated through a shared coordinator using a distributed metadata database called DMDS. In this new structure, a parallel and co-evolutionary genetic algorithm has been used to optimize and match the process matrix simultaneously. It uses a genetic algorithm with a parallel and co-evolutionary executive structure in the mapping process to allocate resources, transfer, and store. In this algorithm, processor speed, number of mappers, amount of data, temporary memory, communication speed and bandwidth of the network were considered in the decision-making algorithm to decide a better decision which serve more allocating resources and reducing the execution time. Evaluation of the proposed method has been performed in a sample test program which counts the input data words. In this scenario, the variable parameters affecting the final results where they apply as the input to the proposed allocation algorithm and their effect has been investigated on the performance for two test scenarios. The results of evaluation show that proposed technique improves performance especially for computation in big data volume, and dramatically reduce processing time across the MapReduce process. The results of the tests show that our proposed changes to the standard model of the MapReduce increases the use of mappers and reduce operating costs in implementation of the MapReduce process. According to the results, in case of small data volume, the overhead caused by the genetic algorithm optimization program slightly slows down the system. However, with increasing the data volume, effect of this overhead decreases by reducing the number of processing steps; as a result, the overall execution time of the MapReduce program has been reduced. Finally, the collaborative mappers greatly improve the efficiency and scalability compare to the standard MapReduce.

Appendix A:

C13=(CPU+ Core+#A)+BW+SP+(RAM +V+ Class)
C13=(2+3+3)+1+1+(3+1+2)=8+2+6=16
C22=(1+3+3)+1+2+(2+1+3)=7+3+6=16
F=16+16=32
C11=(2+3+2)+1+1+(1+1+2)=7+2+4=13
C22=(1+3+2)+1+2+(2+1+3)=6+3+6=15
C33=(3+4+3)+2+3+(3+1+2)=10+5+6=21
F=13+15+21=49
C14=(2+3+1)+1+1+(4+1+3)=6+2+8=16
C32=(3+4+2)+1+3+(2+1+3)=9+4+6=19
F=16+19=3516+19=35
C24=(1+3+1)+2+2+(4+1+3)=5+4+8=17
C31=(3+4+1)+1+3+(1+1+2)=8+4+4=16
F=17+16=33

Appendix B:

C11=(2+3+3)+1+1+(1+1+2)=8+2+4=14
C22=(1+3+3)+1+2+(2+1+3)=7+3+6=16
C33=(3+4+3)+2+3+(3+1+2)=10+5+6=21
F=14+16+21=51
DS=1230
MP=1000,0100,0010
C13=(2+3+2)+1+1+(3+1+2)=7+2+6=15
C24=(1+3+2)+2+2+(4+1+3)=6+4+8=18
C32=(3+4+2)+1+3+(2+1+3)=9+4+6=19
F=15+18+19=52
DS=0412
MP=0010,0001,0100
C14=(2+3+1)+1+1+(4+1+3)=6+2+8=16
C22=(1+3+1)+1+2+(2+1+3)=5+3+6=14
C31=(3+4+1)+1+3+(1+1+2)=8+4+4=16
F=16+14+16=46
DS=3201
MP=0001,0100,1000.

References

[1] Kai Zhu, Huaguang Song, Lijing Liu, Jinzhu Gao, Guojian Cheng, “Hybrid Genetic Algorithm for Cloud Computing Applications”, IEEE Asia-Pacific Services Computing Conference (APSCC), 2011, pp. 182 – 187.

[2] Arianyan, Ehsan, et al. "Efficient resource allocation in cloud data centers through genetic algorithm." 6th International Symposium on Telecommunications (IST). IEEE, 2012.

[3] Bosagh Zadeh, Reza; Carlsson, Gunnar. "Dimension Independent Matrix Square Using MapReduce" (PDF). Retrieved 12 July 2014.

[4] Ng, Andrew Y.; Bradski, Gary; Chu, Cheng-Tao; Olukotun, Kunle; Kim, Sang Kyun; Lin, Yi-An; Yu, YuanYuan (2006). "Map-Reduce for Machine Learning on Multicore". NIPS 2006.

[DOI: 10.52547/itrc.14.4.28]

[5] Ullman, J. D. (2012). "Designing good MapReduce algorithms". XRDS: Crossroads, the ACM Magazine for Students. 19: 30. doi:10.1145/2331042.2331053.

[6] Khezr, Seyed Nima, and Nima Jafari Navimipour. "MapReduce and its applications, challenges, and architecture: a comprehensive review and directions for future research." Journal of Grid Computing 15.3 (2017): 295-321.

[7] D. J. DeWitt, J. F. Naughton, and D. A. Schneider,"Parallel sorting on a shared-nothing architecture using probabilistic splitting," in PDIS, 1991, pp. 280–291.

[8] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in EuroSys, 2007, pp. 59–72.

[9] V. R. Borkar, M. J. Carey, R. Grover, N. Onose, and R. Vernica, "Hyracks: A flexible and extensible foundation for data-intensive computing," in ICDE, 2011.

[10] M. Zaharia et al., "Spark: cluster computing with working sets," in HotCloud, USENIX workshop on Hot topics in cloud computing, 2010.

[11] A. Gates et al., "Building a highlevel dataflow system on top of MapReduce: the Pig experience," PVLDB, vol. 2, no. 2, pp. 1414–1425, 2009.

[12] A. Thusoo et al., "Hive - a petabyte scale data warehouse using Hadoop," in ICDE, 2010, pp. 996–1005.

[13] Anchalia, P.P. and K. Roy. The k-nearest neighbor algorithm using MapReduce paradigm. in Intelligent Systems, Modelling and Simulation (ISMS), 2014 5th International Conference on. 2014. IEEE.

[14] Apache Hadoop, http://hadoop.apache.org.

[15] "Introducing Sqoop". Retrieved Jan 1, 2019.

[16] Vernica, Rares, et al. "Adaptive MapReduce using situation-aware mappers." Proceedings of the 15th International Conference on Extending Database Technology. ACM, 2012.

[17] Khezr, S.N. and N.J. Navimipour, MapReduce and its application in optimization algorithms: a comprehensive study. Majlesi Journal of Multimedia Processing, 2015. 4(3).

[18] Gopalani, S. and R. Arora, Comparing apache spark and map reduce with performance analysis using k-means. International journal of computer applications, 2015. 113(1).

[19] Pellakuri, V. and R. Rao, Hadoop Mapreduce framework in big data analytics. International Journal of Computer Trends and Technology (IJCTT), 2014. 8(3): p. 115-119.

[20] Pakize, S.R. and A. Gandomi, Comparative study of classification algorithms based on MapReduce model. International Journal of Innovative Research in Advanced Engineering (IJIRAE), 2014. 1(7): p. 251-254.

[21] Yang, H.-c., et al. Map-reduce-merge: simplified relational data processing on large clusters. in Proceedings of the 2007 ACM SIGMOD international conference on Management of data. 2007. ACM.

[22] T. Nykiel et al., "MRShare: sharing across multiple queries in MapReduce," PVLDB, vol. 3, no. 1, pp. 494–505, 2010.

[23] A. Shatdal et al., "Adaptive parallel aggregation algorithms," in SIGMOD Conf., 1995, pp. 104–114.

[24] Czajkowski, Grzegorz; Marián Dvorský; Jerry Zhao; Michael Conley. "Sorting Petabytes with MapReduce – The Next Episode". Retrieved 7 April 2014.

[25] Adve, S., et al. "Parallel computing research at Illinois: The UPCRC agenda." Urbana, IL: Univ. Illinois Urbana-Champaign (2008).

[26] Ma, X., Li, X., Zhang, Q., Tang, K., Liang, Z., Xie, W., & Zhu, Z. (2018). "A Survey on Cooperative Co-evolutionary Algorithms". IEEE Transactions on Evolutionary Computation, Volume: 23, Issue: 3, June 2019

[27] A N M Bazlur Rashid (Australia), Tonmoy Choudhury (Australia) "Knowledge management overview of feature selection problem in high-dimensional financial data: cooperative co-evolution and MapReduce perspectives" Problems and Perspectives in Management, Volume 17, Issue 4, 2019.

Mohammad Reza Ahmadi received the B.Sc. and M.Sc. degrees in Electrical Engineering and Communication Systems from K.N.T. University of Technology in 1986 and 1990 respectively. He received his Doctor degree in Communication Networks from Tokyo Institute of Technology, Tokyo, in 1997. Currently he is the project manager and researcher in IT department of ICT Research Institute (Iran Telecom. Research Center). His research interests are Resource Optimization in Data Centers, Virtualization, Cloud Computing Techniques, and Big Data Applications.

Davood Maleki is a faculty member of ICT Research Institute (ITRC). He received M.Sc. degree in Computer Software engineering from Ferdowsi University of Mashhad. Currently, he works as a colleague and supervisor in fundamental, practical and strategic projects in Research Institute of Communication and Information Technology.

[DOI: 10.52547/itrc.14.4.28]