International Journal of
Advances in Scientific Research and Engineering (ijasre)

E-ISSN : 2454-8006

DOI: 10.31695/IJASRE.2021.33923

Volume 7, Issue 4
April - 2021

Parameter Settings Optimization in MapReduce Big Data processing using the MOPSO Algorithm

Lennah Etyang[1] Lawrence Nderu [3] Waweru Mwangi[3]

School of Computing and Information Technology

Jomo Kenyatta University of Agriculture and Technology

Nairobi, Kenya

ABSTRACT

Big data is a commodity that is highly valued in the entire globe. It is not just regarded as data but in the world of experts, we can derive intelligence from it. Because of its characteristics which are Variety, Value, Volume, Velocity, and the growing need of how it can be handled, Organizations are facing difficulties in ensuring optimal as well as affordable processing and storage of large datasets. One of the already existing models used for rapid processing together with storage in big data is known as Hadoop MapReduce. MapReduce is used for large-scale data processing in a parallel and distributed computing environment, while Hadoop is used for running applications and storing data in clusters of commodity hardware Furthermore, the Hadoop MapReduce framework needs to tune more than 190 configuration parameters which are mostly done manually. Due to complex interactions and large spaces between parameters, manual tuning is not effective. Even worse, these parameters must be tuned every time Hadoop MapReduce applications are run. The main goal of this research is to create an algorithm that will improve efficiency by automatically optimizing parameter settings when MapReduce jobs are running. The algorithm employs the Multi-Objective Particle Swarm Optimization (MOPSO) technique, which uses two objective functions to look for a Pareto optimal solution while optimizing the parameters. The results of the experiments have shown that the algorithm has remarkably improved MapReduce job performance in comparison to the use of default settings.

Key Words: *Multi Objective Problem, MOPSO, PITCH, MapReduce, Pareto Optimality, Parallel Computing Toolbox.*

1. INTRODUCTION

Volumes of data originating from a variety of source documents have shown significant growth. For big data, there is a need to have various techniques to enable one to process and store it as it grows over a period considering its characteristics [1]. Despite many organizations knowing the values and benefits of this data as well as gaining more access to it, the challenge of ensuring methods and techniques which provide affordable storage and processing of big data is something they are struggling with [2]. This is due to the expanding demand in being able to handle data which is growing over time [3]. Hadoop MapReduce is a computing technology used in the processing and storing of big data [4]. It has the capabilities of carrying out parallel and distributed processing and storage on different clusters in a manner that ensures scalability and fault tolerance. Even with Hadoop's more than 190 configuration parameters that need to be set to facilitate the execution of jobs, the main challenge is that most users are not even aware of these parameters. Due to a lack of appropriate skills, they cannot be able to gain access to these important configuration options. Moreover, if these parameters are not assigned appropriate values, the system automatically picks the default values, and this leads to high consumption and ineffective use of the computing resources hence making it difficult for Hadoop MapReduce to achieve optimum performance. Furthermore, getting an algorithm that can develop a multi-objective function through correlating configuration parameters has been extremely difficult since there is complexity in the way parameter settings are interconnected. For this reason, we developed PITCH to automatically tune parameter settings while MapReduce jobs are running [20]. This algorithm employed the MOPSO concept which uses more than one condition to decide and conclude the best solution [21]. PITCH has proved to be very powerful in dealing with multi-objective problems in the best way possible as well as providing a set of good solutions putting into consideration Pareto optimality [24]. After running various experiments, our findings showed a major increase in the execution time of MapReduce jobs as compared to the use of default settings.

The second section of this paper gives us a conceptual overview of Hadoop and MapReduce, the third section discusses other related work, and the fourth and fifth sections show our contribution. Discussions and recommendations are contained in sections seven and eight, while the conclusion and future work are in section nine.

2. CONCEPTUAL OVERVIEW OF HADOOP MAPREDUCE

Hadoop has got two key parts; The Hadoop Distributed File System (HDFS) and MapReduce. MapReduce runs on HDFS that is scalable and parallel. It involves two distinct tasks, and they are map and reduce tasks. The input dataset is taken by the map task and then given to the pairs in between for sorting and partitioning depending on the reducer. Each output of the job is then moved into the reducer to generate ultimate outputs.

2.1 Hadoop Architecture

HDFS plays the main role in storage due to its reliability and fault tolerance feature. It can perform configuration on the block replications to ensure data is well protected and recovery mechanisms are available in scalable and fault-tolerant situations. The most important Hadoop modules are job tracker together with task tracker. The job tracker is a master, and its main role is to ensure user jobs that are taken are split into smaller tasks. It also facilitates scheduling of tasks ensuring those which do not make it are re-executed again. Jobs are then assigned task trackers in clusters of the nodes. The task tracker's role is to process the nodes used to run MapReduce tasks. It then sends a set of messages to the job tracker which carries information regarding the status of how the running tasks are and which slots are available. "Figure 1" shows the Hadoop architecture.

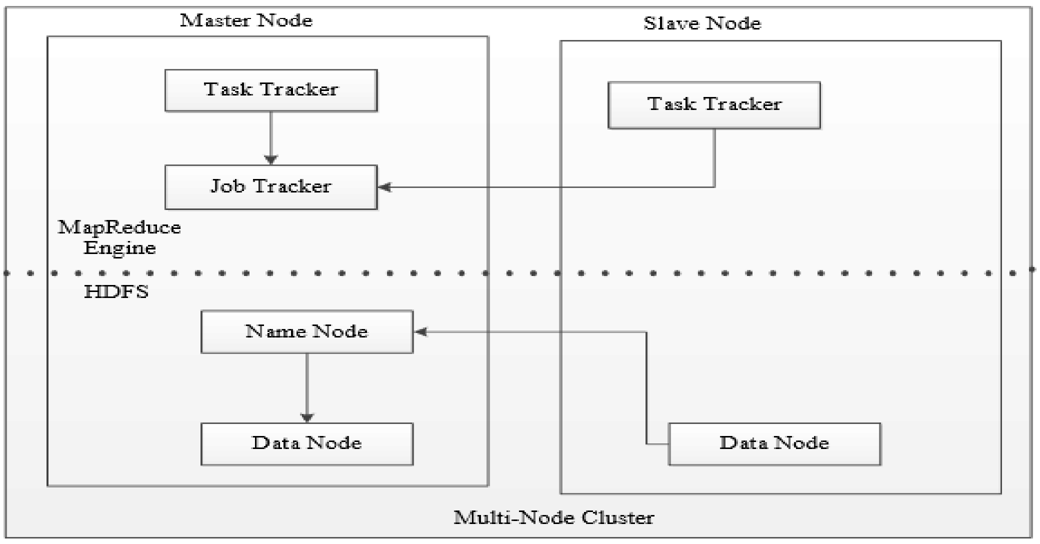


Figure 1. Hadoop Architecture

2.2 MapReduce Architecture

The entire map/reduce process does its operations using three phases. 1) Mapping: Being the first phase in MapReduce program execution, this is the phase whereby data is taken to the map function then values are gotten in the output. 2) Shuffling: Shuffling phase takes in the output from the mapping phase. This involves putting together relevant values from the Map output. 3) Reducing: The phase consolidates all the output values which come from the shuffling phase. The values are then put together as a summarized dataset which is returned as a complete output as shown in "Figure 2".

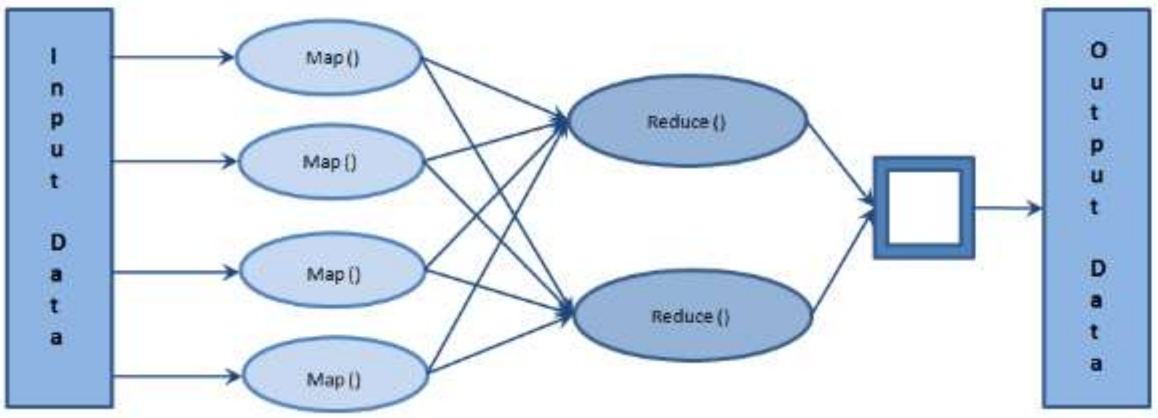


Figure 2: MapReduce Architecture

3. ANALYSIS OF EXISTING MODELS USED FOR PARAMETER OPTIMIZATION

In recent years, research has been done for Hadoop MapReduce performance optimization. Methodologies used by existing studies are varying from optimization of parameter values to scheduling of the jobs. Load balancing and data locality is also an area where the focus has been put on. Mansaf Alam [6] researched the Hadoop platform for cloud-based big data analytics. They clarified that, while the pace of growth in big data is rapid, managing the same data is difficult due to its diverse characteristics. Their work involved the classification of input data then routing this data to several nodes for processing. When processing of each node was completed, they put together the output of every node to get the result. Hadoop in this case was used to facilitate partitioning and processing of data. Nikhil Rajyaguru [7] understood, compared, and found the difference between all the big data technologies which were in use by mobile applications. MapReduce framework was used to compare these technologies while using test cases in already ongoing research. The result of their research showed that big data tools were usable in computers only but not in mobile devices.

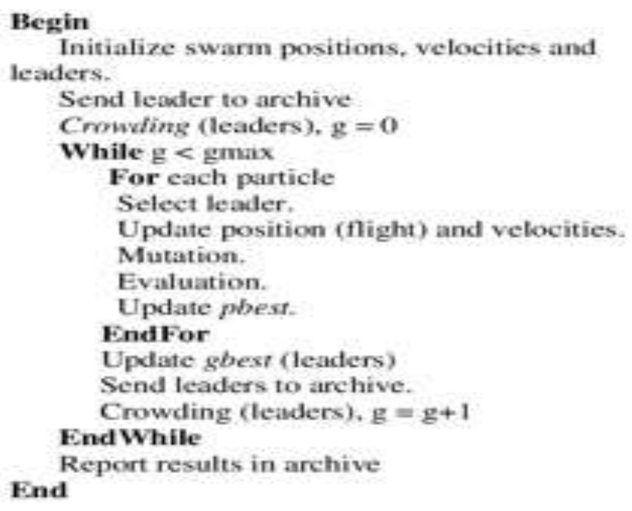
Yang [8] developed a model for big data processing in a parallel computing environment. He described the process of carrying out performance in MapReduce jobs in the cloud together with writing mapper and reducer classes by use of objects. The framework was able to handle large datasets, but it was not able to show its tedious details and complexities such as scalability. Samira Daneshyar [9] provided a systematic analysis and a comprehensive review in processing large datasets. In the same case, they described challenges encountered in data handling and all the computing requirements using Hadoop MapReduce. They showed all the requirements which make it possible for MapReduce to process big data and they finally demonstrated in an experiment how MapReduce can be run in the cloud. Voruganti [10] stated that the Hadoop framework can work well when the key/value pairs are used. The data which is less structured can handle quite a several challenges in different problem domains. Their results showed that many forms of data can be successfully analyzed after being transformed into key/value pairs. Guangdeng Liao [11] evaluated various approaches which enabled automatic parameter tuning together with machine learning models which were based on costs. After establishing that existing models were not adequate, they came up with the Gunther algorithm for optimization. Gunther was evaluated in different clusters which had distinct resources. The results of their experiment demonstrated that in a small number of trials, Gunther attained a near-optimal performance. Min Li [12] developed MRONLINE used for online performance. MRONLINE’s job was to monitor job execution and to do parameter tuning in terms of data that was collected. In MRONLINE, every task had configurations not similar as compared to other research works which used the same configurations. An algorithm was used to converge into near-optimal configurations. The results of their work demonstrated how effectively MRONLINE had improved performance up to about 30 percent as compared to default configuration settings. M. Khan, Y. J [13] developed a Gene Expression Programming (GEP) model representing a correlation in the parameters to be configured which focused on previous records of jobs run in Hadoop. The model employed the Particle Swarm Optimization (PSO) method that made use of an objective function to get parameter settings which are optimal or closer to optimal. In their results, they demonstrated that the work greatly improved Hadoop’s performance compared to default settings. All the existing work has shown that changes in parameter values are done either before or after Hadoop MapReduce jobs are executed. Also, the MOPSO concept has never been used for parameter tuning in the big data processing. For this reason, we proposed PITCH, an algorithm that used the MOPSO concept to develop a multi-objective function to optimize parameter settings while MapReduce jobs are running. MOPSO is gaining popularity due to its ability to simultaneously optimize multiple objective functions. Instead of having a single solution, MOPSO defines a set of solutions known as Pareto optimal sets [19].

4. ENHANCING MOPSO

This section describes how PITCH used the MOPSO principle to optimize MapReduce parameter settings. MOPSO was proposed by Coello Coello, in 2004 [15] to solve Multi-Objective Problems. It is an advancement of Kennedy and Eberhart's PSO algorithm, which was first published in 1995[15]. PSO was influenced by a social complex behavior of a flock of birds or a school of fish that achieves a higher degree of intelligence by simple acts of each member of the flock flying and performing repetitive tasks [18]. Swarm knowledge has been used to find an optimal solution using multi-dimensional search spaces, effectively solving single-objective problems [19]. However, since all particles in a swarm have the same flight experience, PSO can only find a global optimum solution [24]. Due to its inability to solve multi-objective problems, the algorithm was therefore modified to a regular MOPSO.

4.1 PITCH algorithm

Multi-Objective Optimization Problems (MOOP) have multiple objectives containing constraints that need to be satisfied using feasible solutions [18] found using the Pareto Optimality theory [12]. Below is the PITCH algorithm Pseudocode:



```
Begin
    Initialize swarm positions, velocities and
leaders.
    Send leader to archive
    Crowding (leaders), g = 0
    While g < gmax
        For each particle
         Select leader.
         Update position (flight) and velocities.
         Mutation.
         Evaluation.
         Update pbest.
        EndFor
        Update gbest (leaders)
        Send leaders to archive.
        Crowding (leaders), g = g+1
    EndWhile
    Report results in archive
End
```

In “(1),” a MOOP having several objective functions and a number of equality and inequality constraints is formulated:

$$
\begin{aligned}
&\text{Minimize/Maximize } fn(x), n = 1,2,3,4,5,\ldots\ldots,N;\\
&\quad \text{Subject to:}\\
&\qquad hk(x) = 0, \quad k = 1,2,3,4,5\ldots\ldots,K;\\
&\qquad gl(x) <= 0, \quad j = 1,2,3,4,5,\ldots\ldots,L. \qquad (1)
\end{aligned}
$$

Where:

The Objective function is fn(x).
x is a decision vector that denotes possible outcomes.
N are the number of objectives.
The number of equality constraints is denoted by the letter K.
The number of inequality constraints is denoted by the letter L.

The aim in such a situation is to optimize n objective functions at once, with the result being a good compromise of solutions that indicate a better trade-off between objectives [27]. Since parameter selection has a significant impact on optimization results, we must first randomly initialize the locations of the particles in the search space to ensure uniform coverage, as well as set velocity to zero, as shown in “(2)”.

$$V_i^{t+1} = \omega v_i^t + c_1 r_1 (pbest_i^t - X_i^t) + c_2 r_2 (gbest_i^t - X_i^t) \qquad (2)$$

PITCH updates the current position vector of the swarm by adding velocity 'V' using "(3)" in each iteration:

$$X_i^{t+1} = Xi^t + V_i^{t+1} \qquad (3)$$

constants c1 and c2 are local and global weights, respectively, the velocity and position of particle I are denoted by v. The inertia weight is w, and the absolute time index is t. pbest is a personal archive that contains a particle's most recent non-dominated positions in the past. The global best swarm flying experience is chosen from an external archive by gbestti. The latest external repository for non-dominated solutions, also known as Pareto optimal sets, is vit. The current external archive vit and the population Pt are used to create vit+1, which is a new external archive.

The constriction factors K and L minimize clamp speed, facilitating convergence, and are denoted by the letters "(4)" and "(5)" respectively.

$$V_i^{t+1}=k^*\ \omega v_i^t + c_1\, r_1(pbest_i^t - X_i^t) + c_2\, r_2\,(gbest_i^t - X_i^t) \qquad (4)$$
$$V_i^{t+1}= l^*\omega v_i^t + c_1\, r_1(pbest_i^t - X_i^t) + c_2\, r_2\,(gbest_i^t - X_i^t) \qquad (5)$$

Particles in PITCH, like those in the PSO algorithm, share information and aim for the global best as well as their own personal best memory. The use of several conditions to describe and conclude the best solution as either global or local [21] distinguishes it from PSO. In this case, all the swarm's non-dominated particles are gathered into a geographically based repository to preserve diversity [22]. Every particle chooses its global best objective among its members in the repository [23] and since each particle has its own swarm flying experience, a personal best particle is chosen based on the degree of supremacy as well as probabilistic laws.

4.2 PITCH implementation in MapReduce

A popular implementation of MapReduce is the Hadoop MapReduce which works with HDFS [30]. However, MATLAB also offers this implementation using the MapReduce function [31]. It processes data in small blocks that fit into memory individually using a data store. Any code block goes through the map step, in which data is formatted for processing, and then the intermediate data blocks go through the reduce phase, in which all the intermediate results are combined to generate a final output [31]. The MapReduce functions are called automatically during execution [10], and they must be able to meet certain criteria to operate properly. Data, details, and intermKVStore are the three input functions used in the map phase. The output of the call that reads the function from the input datastore is data and info, with MapReduce execution occurring automatically before each call to the map function. The map function then adds the key-value pairs to the intermKVStore object, the Intermediate Key Value Store. Here is an example of a map function in action:

```
function MeanDistMapFun(data, info, intermKVStore)
         distances = data.Distance(~isnan(data.Distance));
        sumLenValue = [sum(distances)  length(distances)];
        add(intermKVStore, 'sumAndLength', sumLenValue);
end
```

The first line filters all the NaN values in a data block. The second line's job is to construct a vector with two elements, each with a total distance other than the block count. The third line inserts a vector of all of the values into the intermKVStore using the key'sumAndLength.' After running this map function all of the data blocks d in ds, the cumulative count in data block is saved in intermKVStore object. The intermKey, intermValIter, and outKVStore are the reduce functions' inputs. The active key, which is added by the map function, is called intermKey. Any call to the reduce function by MapReduce is used to generate a new unique key from the intermediate KeyValueStore object's keys. The ValueIterator associated with the intermKey is the intermValIter. All the values in the active key are also stored in the object. Finally, the reduce function uses the outKVStore, also known as the final KeyValueStore, to add the key-value pairs. After that, the output is submitted to the output datastore. A basic reduce function is shown below:

```
function MeanDistReduceFun(intermKey, intermValIter, outKVStore)
         sumLen = [0 0];
while hasnext(intermValIter)
sumLen = sumLen + getnext(intermValIter);
 end
add(outKVStore, 'Mean', sumLen(1)/sumLen(2));
end
```

The reduce function loops through all the intermValIter's distance and count values. It keeps track of the total distance run and counts for each pass [16]. When the loop is over, the reduce function uses a simple division to measure the total mean, and then adds a single key to outKVStore. One of MapReduce's key advantages is its ability to run in a variety of programming environments [24]. MapReduce's ability to perform calculations on large collections of data, on the other hand, makes it unsuitable for performing calculations on data sets that can be loaded directly into computer memory and analyzed using conventional methods [5]. As a result, MapReduce is critical for performing statistical or analytical calculations on datasets that are too large to fit in memory [33]. Using a Parallel Computing Toolbox, the MATLAB Parallel Server, and the MATLAB Compiler, the mapreducer configuration function is used to modify the execution environment [31]. This helps you to start small and check the map and reduce functions before scaling up to run larger computations. The Parallel Computing Toolbox aids in the acceleration of MapReduce job execution by using the full processing capacity of multicore machines to run applications in a parallel pool of workers. The MATLAB Parallel Server allows you to run the same applications on remote computer clusters like Hadoop using tall arrays, enabling you to run big data applications that do not fit in memory [24]. On the other hand, the MATLAB compiler allows users to build standalone MapReduce applications or deployable archives that can be shared between applications or deployed in Hadoop systems [28].

5. PITCH EXPERIMENT SETUP

In this section, we are presenting how the implementation was and some of the experiments which were performed, showing outcomes of the MapReduce job execution using the PITCH algorithm.

5.1 Environment

Experiments were run on an HP supercomputer with an Intel(R) Core (TM) i5-8250U CPU operating at 1.60 GHz and 1.80 GHz and 8 GB of RAM. In a serial environment, we ran the experiment in a local machine while in a parallel experiment we used the MATLAB parallel Computing Server. The MATLAB Parallel Computing Toolbox changed the execution environment into a cluster of four nodes which scaled up the performance to run in larger computations. Both serial and parallel executions were run using MATLAB R2020a application.

5.2 Performance evaluation

The ZDT benchmark function [30], a test suite named after its authors Zitzler, Deb, and Thiele [29], was used to evaluate the algorithm's performance. This algorithm shared the code for particle movement and evaluation in order to find a trade-off between two of our goals: reducing the amount of Random Access Memory (RAM) allocated to each task and improving Input/Output (I/O) efficiency, thereby improving the execution time of each MapReduce job. We came up with the objective function and the constraints to be minimized after defining the problem. The MapReduce configuration parameters, which define the number of decision variables, are the constraints in our case. These are the jobs' I/O locations, data's I/O format, and the classes that comprise map and reduce functions. The inspiration for the chosen parameter values comes from the running records of MapReduce jobs, which can be memory or I/O intensive. To solve the problem, we evaluated the trade-off between two objectives, which in this case are the quantity of RAM and the I/O Performance using the following techniques:

- We set memory to be maximum and found a single objective minimal I/O performance over the designs which satisfies the maximum memory constraint.
- We set I/O performance to be maximum and found a single objective minimal memory over the designs which satisfy the I/O performance constraint.
- We then solved a multi-objective problem, visualizing the trade-off between the two objectives using the minimization function below:

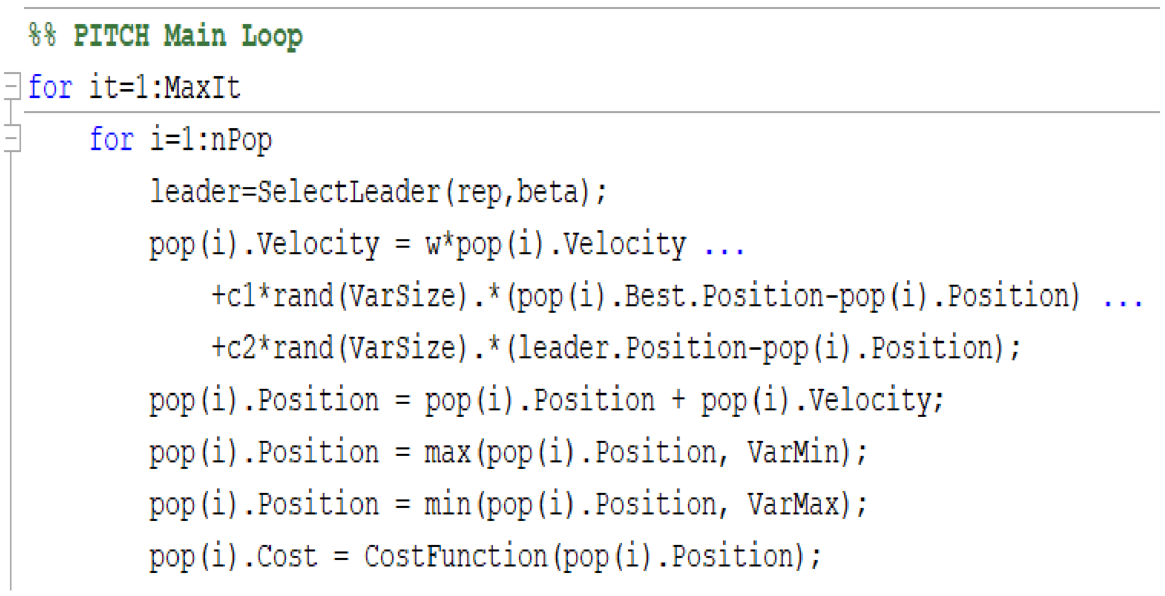
```
function z = parameters(x)
        n=numel(x);
        f1=x(1);
        g=1+9/(n-1)*sum(x(2:end));
        h=1-sqrt(f1/g);
        f2=g*h;
        z=[f1
        f2];
end
```

We equated the decision variables to 10, with a matrix size of 1 having lower bound and upper bound variables of 0 and 1 respectively. The particle swarm size was set to 100, the number of iterations to 200, and the repository size to 50. In each iteration, the values of c1 and c2 were set to 1.4269, w to 0.49, and the values of r1 and r2 were randomly selected between 0 and 1. Table 1 lists the parameters used in the PITCH algorithm.

Table 1: Parameters used in PITCH Algorithm

Parameters	**Value**
Number of Unknown variables	10
Matrix Size	1
Lower Bound Decision Variables	0
Upper Bound decision variables	1
Particle Swarm size	100
Number of Iterations	200
Repository Size	50
C1 and C2	1.4269
Inertia weight	0.49
r1 and r2	0 and 1

The number of unknown variables were used to calculate the new position of a particle in every iteration. The local best values were compared with the new fitness values within the repository then updated accordingly. In the same manner, the global best position was updated. The section of the code below is the algorithm's main loop:



```
%% PITCH Main Loop
for it=1:MaxIt
    for i=1:nPop
        leader=SelectLeader(rep,beta);
        pop(i).Velocity = w*pop(i).Velocity ...
            +c1*rand(VarSize).*(pop(i).Best.Position-pop(i).Position) ...
            +c2*rand(VarSize).*(leader.Position-pop(i).Position);
        pop(i).Position = pop(i).Position + pop(i).Velocity;
        pop(i).Position = max(pop(i).Position, VarMin);
        pop(i).Position = min(pop(i).Position, VarMax);
        pop(i).Cost = CostFunction(pop(i).Position);
```

5.3 Movement of particles in a Pareto search

Considering the number of iterations, "Fig. 3" and "Fig. 4" show how the particles of the PITCH algorithm moved in the search space towards the Pareto optimal solution:

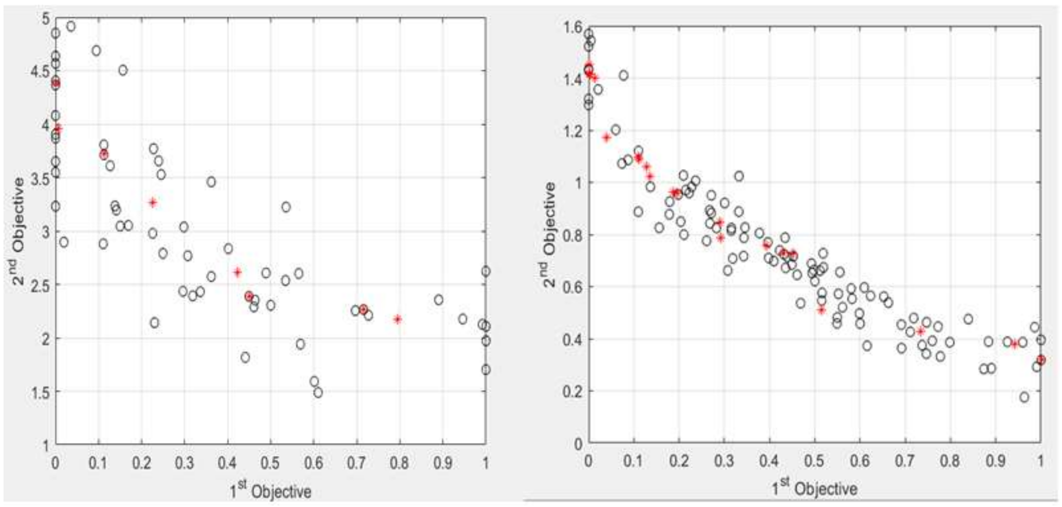


Figure 3: Iteration 5 and Iteration 25

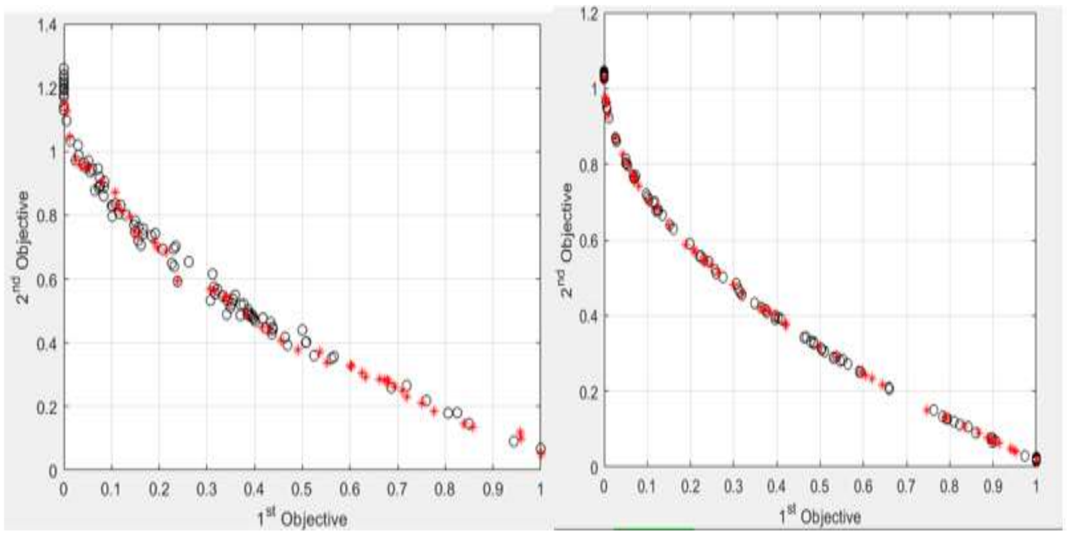


Figure 4: Iteration 50 and Iteration 200

It is well known that multi-objective algorithms have some limitations concerning convergence and diversity [21]. To avoid these issues, we used the archiving technique, which enhanced convergence, and the leader's selection method, and increased search diversity. It used a method that applied equality and inequality constraints k and l [26] to limit the velocity of each particle, which varies depending on the values of C1 and C2. By doing this, the algorithm introduced more convergence towards the Pareto front guiding the solution to a precise area within the repository. When the repository became full, the best point among all the solutions in the repository was obtained by determining the domination, adding the non-dominated members into the new repository thereby selecting a new solution. Minimization was obtained through differential evolution [32].

6. EXPERIMENTAL RESULTS OF PITCH

6.1 Experiment 1

In the first experiment, the MapReduce application was executed in a serial machine using default parameter settings. For this experiment, the execution time was 153 seconds. MapReduce application was then executed in the PITCH algorithm using Parallel Computing Toolbox expending two iterations consecutively. From the output, execution time was 24 seconds in the first iteration and 18 seconds in the second iteration.

6.2 Experiment 2

In “Fig. 5”, PITCH was executed using the following settings: Number of particles in the swarm: 125, 150, 175, and 200. Fixed parameters: 200 iterations, 10 decision variables, 50 repository members.

DOI: 10.31695/IJASRE.2021.33923

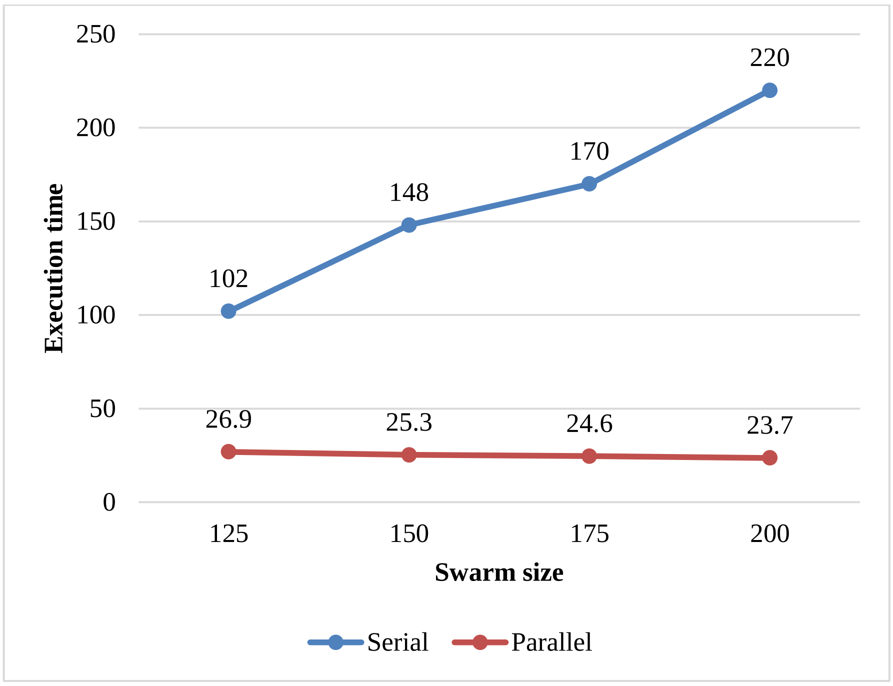


Figure 5: Increase in swarm size

From the output, it can be seen that there is a gradual reduction in the execution time from 26.9 to 23.7 seconds.

6.3 Experiment 3

In “Fig. 6”, we ran the implementation using the following settings: Number of members in the repository: 20, 30, 40, and 50. Fixed parameters: 200 iterations, 10 decision variables, and 100 particles in the swarm.

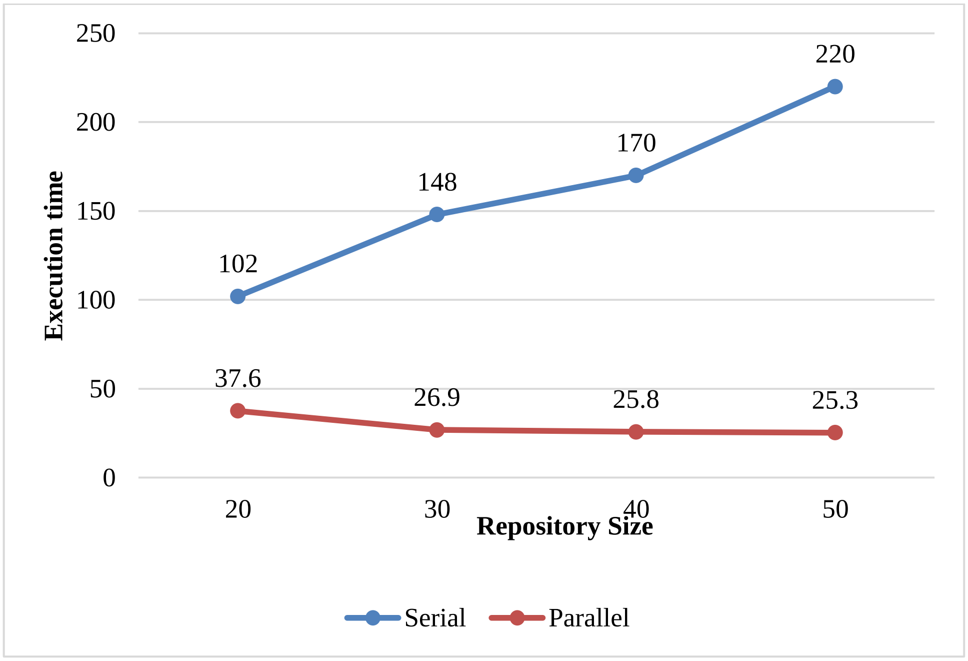


Figure 6: Reducing the repository size

Experiment 3 shows that the execution time has continued to decrease, from 37.6 seconds to 25.3 seconds.

6.4 Experiment 4

In “Fig. 7”, PITCH was executed using the following settings: Data size: 12MB, 24MB, 50MB, and 500MB. Fixed parameters: 200 iterations, 10 decision variables, 100 particles in the swarm, and repository size 20.

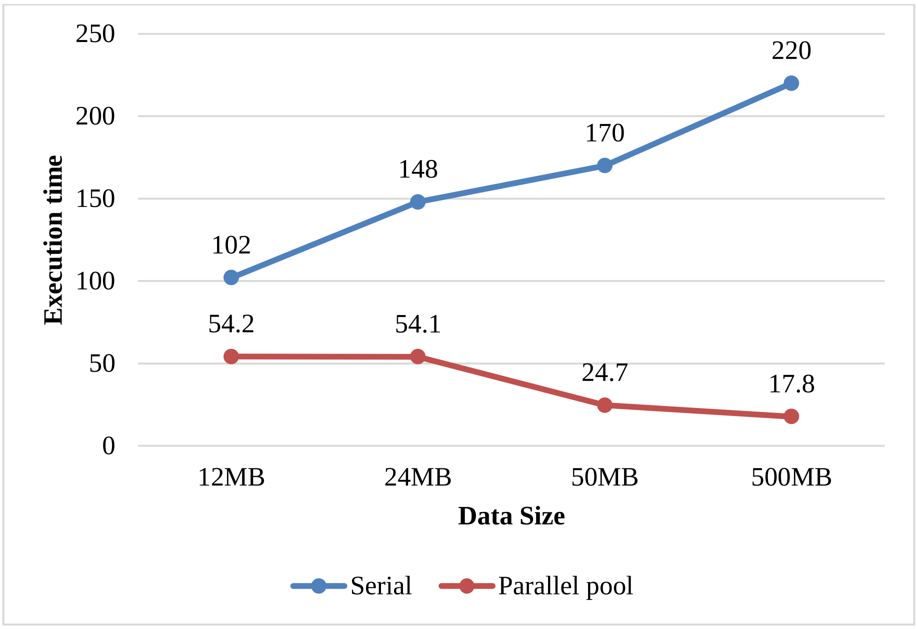


Figure 7: Increase in the Data Size

The execution time for the 12MB dataset was 54.2 seconds, while the 500MB dataset took 17.8 seconds, according to experiment 4.

7. DISCUSSION

During experiments, we noted that both serial and parallel implementations can be scaled easily and be used for larger swarm sizes to handle more complex optimization problems. We observed that during the first iteration in the Parallel Computing Toolbox, the algorithm performed slower but then it continually improved from the second and consecutive iterations. In a serial environment, execution time was constantly going up when we increased the values in the parameter settings. Nonetheless, implementation using the Parallel Computing Toolbox showed very promising results. The algorithm's efficiency improved gradually as we increased the number of particles in the swarm and decreased the repository size in each iteration. Even better, as we scaled up and increased the data size, the execution time decreased significantly. The decision variables and the number of iterations were held constant during all the experiments. For the various data sizes we used, we found that the execution time of a MapReduce job in a smaller dataset had no major effect on the algorithm's performance as opposed to when the data size was increased. This attributes to the fact that MapReduce programming technique is suitable for processing data which cannot fit into the local computer memory. Therefore, using a smaller data size is not suitable for running MapReduce applications. If we would also increase the hardware and other resources, the implementation can comparatively scale well.

8. RECOMMENDATIONS

Based on the range of values tested, there is a need to establish some guidelines which should be followed while tuning optimization parameters used by PITCH. Some of our recommendations for effective optimization are as follows:

8.1 The total number of particles in the swarm

This is the algorithm's population size, and we propose a population size of 100 to 200 particles.

8.2 Number of Iterations

This parameter relates to the particles in the swarm. The relationship is inversely proportional whereby if the number of particles is lower, then the cycles should also be smaller. The recommended use is between 100 and 200.

8.3 Repository size

This parameter gives us the maximum number of non-dominated members to be stored in the repository. It also determines the quality produced in each Pareto front. We recommend the repository size to be between 20 and 50.

8.4 Inertia weight and damping rate

This parameter has a significant effect on convergence and exploration. Inertia weight should be set to 0.49 and the damping rate to 0.99.

9. CONCLUSION

The MOPSO concept is gaining popularity due to its ability to simultaneously optimize multiple objective functions. Instead of using a single solution, PITCH defines a set of solutions known as Pareto Optimal sets. Since most real-world problems, including those in big data processing, are multi-objective, it is crucial to use simultaneous optimization to solve them. This can only be achieved using Algorithms. It is also significant to note that the way objective functions are measured is always competing and conflicting [21]. As a result of competing for objective functions, a set of multiple solutions rather than a single optimal solution arises. This is because no solution is better than another when all objectives are put into consideration. In our research, the main aim was to optimize parameters having two objective functions with the goal of exploring convergence and diversity by adjusting specific features. From the results, PITCH has proved to be very powerful in dealing with multi-objective problems in the best way possible as well as providing a set of good solutions putting into consideration Pareto optimality. [24]. Nonetheless, just as other Multi-Objective Evolutionary Algorithms (MOEA), the PITCH algorithm can decline during multi-objective optimization and may also face a problem of convergence and diversity. Future work includes exploring other features of PITCH in search of more diversified estimates of the Pareto front, without losing convergence. Also, PITCH will be analysed in other benchmarking problems which involve maximizing an objective function. There is more to do in areas that emphasize PITCH parameters' self-adaptation, efficiency as well as applied work in theoretical development.

ACKNOWLEDGMENT

The authors would like to acknowledge MathWorks® for providing MATLAB R2020a [37], the main software which was used in the implementation of this research. More directly to Thomas F. Coleman following his contribution towards constrained minimization functions and the use of the Optimization Toolbox™. Finally, to Jomo Kenyatta University of Agriculture and Technology for allowing the authors to spend time and resources in the University.

REFERENCES

[1] J. M. Cavanillas, E. Curry, and W. Wahlster, "New Horizons for a DataDriven Economy: A Roadmap for Usage and Exploitation of Big Data in Europe," New Horizons a Data-Driven Econ. A Roadmap Usage Exploit. Big Data Eur., pp. 1–303, 2016.

[2] K. N. Aye, "A Platform for Big Data Analytics on Distributed Scaleout Storage System A Platform for Big Data Analytics on Distributed Scale-out Storage System Kyar Nyo Aye University of Computer Studies , Yangon A thesis submitted to the University of Computer Studi," no. November, 2015.

[3] N. Francis and S. Kurian K, "Data Processing for Big Data Applications using Hadoop Framework," Ijarcce, no. April, pp. 177–180, 2015.

[4] I. Technologies, "Map Reduce a Programming Model for Cloud Computing Based On Hadoop Ecosystem," vol. 5, no. 3, pp. 3794–3799, 2014.

[5] Q. Lu, Z. Li, M. Kihl, L. Zhu, and W. Zhang, "CF4BDA: A Conceptual Framework for Big Data Analytics Applications in the Cloud," IEEE Access, vol. 3, no. March 2017, pp. 1944–1952, 2015.

[6] M. Alam and K. Ara Shakil, "Big Data Analytics in Cloud environment using Hadoop Mansaf Alam and Kashish Ara Shakil Department of Computer Science, Jamia Millia Islamia, New Delhi," Dep. Comput. Sci. Jamia Millia Islam. New Delhi.

[7] N. Rajyaguru and M. Vinay, "A Comparative Study of Big Data on Mobile Cloud Computing," Indian J. Sci. Technol., vol. 10, no. 21, pp. 1–10, 2017.

[8] Yang, G. (2011). The Application of MapReduce in the Cloud Computing. 2011 2nd International Symposium on Intelligence Information Processing and Trusted Computing, Hubei,, 154-156.

[9] S. Daneshyar, “Large-Scale Data Processing Using MapReduce in Cloud Computing Environment,” Int. J. Web Serv. Comput., vol. 3, no. 4, pp. 1–13, 2012.

[10] Voruganti, S. (2014). Map Reduce a Programming Model for Cloud Computing Based On Hadoop Ecosystem . (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (3) , 2014, 3794-3799, 3794-3799.

[11] Guangdeng Liao, K. D. (2013). Gunther: Search-Based Auto-Tuning of MapReduce. Part of the Lecture Notes in Computer Science book series (LNCS, volume 8097). European Conference on Parallel Processing, 406-419.

[12] M. Li et al., “Mronline: MapReduce online performance tuning,” HPDC 2014 - Proc. 23rd Int. Symp. High-Performance Parallel Distrib. Comput., pp. 165–176, 2014.

[13] M. Khan, Z. Huang, M. Li, G. A. Taylor, and M. Khan, “Optimizing hadoop parameter settings with gene expression programming guided PSO,” Concurr. Comput. , vol. 29, no. 3, pp. 1–21, 2017.

[14] A. Britto and A. Pozo, “I-MOPSO: A suitable PSO algorithm for many-objective optimization,” Proc. - Brazilian Symp. Neural Networks, SBRN, pp. 166–171, 2012.

[15] C. A. Coello Coello and M. S. Lechuga, “MOPSO: A proposal for multiple objective particle swarm optimization,” Proc. 2002 Congr. Evol. Comput. CEC 2002, vol. 2, pp. 1051–1056, 2002.

[16] Kennedy J, Eberhart R. Particle swarm optimization. In Proceedings., IEEE International Conference on Neural Networks 1995. 1995; 4:1942–1948.

[17] A. W. McNabb, C. K. Monson, and K. D. Seppi, “Parallel PSO using MapReduce,” 2007 IEEE Congr. Evol. Comput. CEC 2007, vol. 15213, pp. 7–14, 2007.

[18] J. Narayan and S. Shetty, “Handling Big Data Analytics Using Swarm Intelligence,” vol. 2, no. 6, pp. 271–275, 2017.

[19] W. Hu, G. G. Yen, and X. Zhang, “Multiobjective particle swarm optimization based on Pareto entropy,” Ruan Jian Xue Bao/Journal Softw., vol. 25, no. 5, pp. 1025–1050, 2014.

[20] J. Leiva, R. C. Pardo, and J. A. Aguado, “Data analytics-based multiobjective particle swarm optimization for determination of congestion thresholds in LV networks,” Energies, vol. 12, no. 7, 2019.

[21] T. Li and B. Yang, “A review of multi-objective particle swarm optimization algorithms in power system economic dispatch,” Int. J. Simul. Syst. Sci. Technol., vol. 17, no. 27, pp. 15.1-15.5, 2016.

[22] G. Lamont, Evolutionary Algorithms for Solving Multi-Objective Problems, no. May. 2007.

[23] M. Pospelova, “Real Time Autotuning for MapReduce on Hadoop/YARN,” 2015.

[24] J. Zhang, D. Xiang, T. Li, and Y. Pan, “M2M: A simple Matlab-toMapReduce translator for cloud computing,” Tsinghua Sci. Technol., vol. 18, no. 1, pp. 1–9, 2013.

[25] S. Mehrjoo and S. Dehghanian, “Mapreduce Based Particle Swarm Optimization for Large Scale Problems,” AICS 2015 Proceeding 3rd Int. Conf. Artif. Intell. Comput. Sci., no. October, pp. 12–13, 2015.

[26] X. Yong, C. Ying, and F. Yanjun, “Research on cloud computing and its application in big data processing of railway passenger flow,” Chem. Eng. Trans., vol. 46, no. 2011, pp. 325–330, 2015.

[27] S. Lalwani, H. Sharma, S. Chandra, S. Kusum, D. Jagdish, and C. Bansal, “REVIEW - COMPUTER ENGINEERING AND COMPUTER SCIENCE A Survey on Parallel Particle Swarm Optimization Algorithms,” Arab. J. Sci. Eng., 2019.

[28] P. M. Roth and M. Winter, “Compiling MATLAB M-Files for Usage Within an MATLAB Compiler mcc,” pp. 1–21, 2004.

[29] E. Zitzler, K. Deb, and L. Thiele, “Comparison of multiobjective evolutionary algorithms: empirical results.,” Evol. Comput., vol. 8, no. 2, pp. 173–195, 2000.

[30] W. J. Lim, A. B. Jambek, and S. C. Neoh, “Kursawe and ZDT functions optimization using hybrid micro genetic algorithm (HMGA),” Soft Comput., vol. 19, no. 12, pp. 3571–3580, 2015.

[31] S. Lalwani, S. Singhal, R. Kumar, and N. Gupta, “a Comprehensive Survey: Applications of Multi-Objective Particle Swarm Optimization (Mopso) Algorithm,” Trans. Comb. ISSN, vol. 2, no. 1, pp. 2251–8657, 2013.

[32] S. Das, S. S. Mullick, and P. N. Suganthan, “Recent advances in differential evolution-An updated survey,” Swarm Evol. Comput., vol. 27, pp. 1–30, 2016.

[33] K. A. Venkatesh, K. Neelamegam, and R. Revathy, “Using MapReduce and load balancing on the cloud: Hadoop MapReduce and virtualization improves node performance Cloud architecture,” no. July, pp. 1–10, 2010.

[34] L. Bao, X. Liu, and W. Chen, “Learning-based Automatic Parameter Tuning for Big Data Analytics Frameworks,” Proc. - 2018 IEEE Int. Conf. Big Data, Big Data 2018, pp. 181–190, 2019.

[35] M. Khan, “Hadoop Performance Modeling and Job Optimization for Big Data Analytics,” Brunel Univ. London, no. March, p. 157, 2015.

[36] S. Cheng, Q. Zhang, and Q. Qin, “Big data analytics with swarm intelligence,” Ind. Manag. Data Syst., vol. 116, no. 4, pp. 646–666, 2016.

[37] @bookMATLAB:2020,year = 2020,author = MATLAB,title = version 7.10.0 (R2020a),publisher = The MathWorks Inc.,address = Natick, Massachusetts.

[38] https://www.mathworks.com/products/parallel-computing.html

[39] P. V. Raja and E. Sivasankar, “Modern framework for distributed healthcare data analytics based on hadoop,” Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 8407 LNCS, pp. 348–355, 2014.

[40] Medel, V., Rana, O., Banares, J. A., Arronategui, U. (2016). Modelling performance and resource management in Kubernetes. Proceedings - 9th IEEE/ACM International Conference on Utility and Cloud Computing, UCC 2016, (September 2019), 257–262.