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

Lennah Etyang
School of Computing and Information Technology
Jomo Kenyatta University of Agriculture and Technology
Nairobi,Kenya
etyang.lennah@students.jkuat.ac.ke

Lawrence Nderu
School of Computing and Information Technology
Jomo Kenyatta University of Agriculture and Technology
Nairobi, Kenya
lnderu@jkuat.ac.ke

Waweru Mwangi
School of Computing and Information Technology
Jomo Kenyatta University of Agriculture and Technology
Nairobi, Kenya
waweru_mwangi@icsit.jkuat.ac.ke

***Abstract*—Big data is a commodity which 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 it's 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 technology used for rapid processing together with storage in big data is known as Hadoop MapReduce. MapReduce is used in parallel and distributed computing environment for large scale data processing whereas Hadoop on the other hand is for running applications and storage of data in clusters of commodity hardware. Furthermore, 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 have to be tuned each and every time Hadoop MapReduce applications are run. The main objective of this research is to develop an algorithm which shall enhance performance by automatically optimizing parameter settings while MapReduce jobs are running. The algorithm employs Multi Objective Particle Swarm Optimization (MOPSO) concept which makes use of two objective functions in order to perform optimization in the parameters by searching for a pareto optimal solution. The results of the experiments have shown that the algorithm has remarkably improved MapReduce job performance in comparison to the use of default settings.**

***Index Terms*—Multi Objective Problems, MOPSO, PITCH, MapReduce, Pareto Optimality, Parallel Computing Toolbox.**

I. Introduction

Volumes of data originating from variety of source documents have shown a significant growth. For big data, there is need to have various techniques to enable one process and store it as it grows over a period of time considering its characteristics[1]. Inspite of 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 overtime[3].

Hadoop MapReduce is a computing technology used in processing and storing big data[4]. It has capabilities of carrying out parallel and distributed processing and storage on different clusters in a manner which ensures scalability and fault tolerance. Even with Hadoop's more than 190 configuration parameters which need to be set in order to facilitate execution of jobs, the main challenge is that most users are not even aware of these parameters. Due to 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 a 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 several experiments, our results showed a significant improvement in the execution time of MapReduce jobs as compared to the use of the default settings.

Section II of this paper gives us a conceptual overview of Hadoop and MapReduce, section III provides a discussion on other related work while section IV to VI shows our contribution. Section VII and VIII is the discussions together with recommendations and section IX provides the conclusion and the future work.

II. 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 tasks which are distinct and they are map and reduce tasks done by Hadoop jobs. 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 and every output of the job is then moved into the reducer in order to generate ultimate outputs.

A. Hadoop Architecture

HDFS plays a main role in storage due to its reliability and fault tolerance feature. It has the ability 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 tacker together with task tracker. The job tracker is a master and its main role is to ensure user jobs which 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. "Fig. 1" shows the Hadoop architecture.

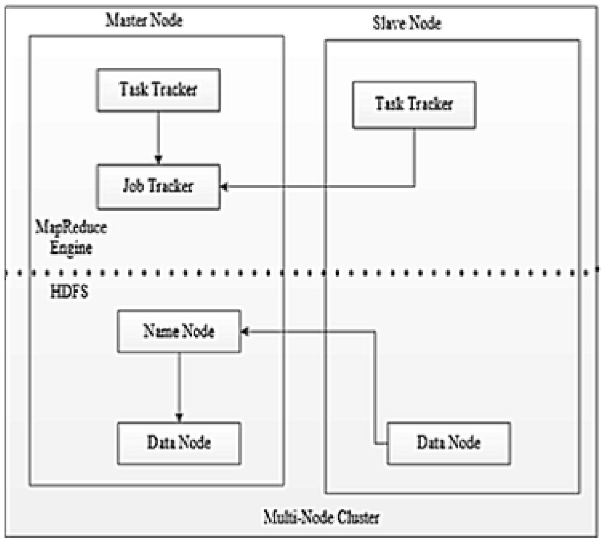


Fig. 1. Hadoop Architecture

B. 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 are returned as a complete output as shown in "Fig. 2" .

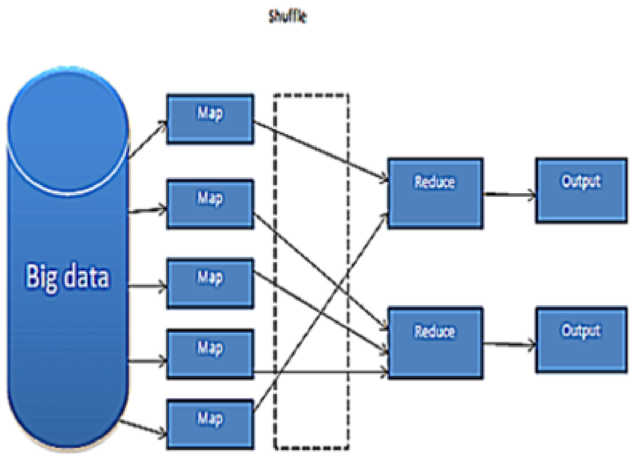


Fig. 2. MapReduce Architecture

III. Analysis of existing models used for parameter optimization

In recent years, research has been done for the purpose of 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 focus has been put on.

Mansaf Alam[6]did research on Hadoop framework for big data analytics in the cloud environment. They outlined that even if the rate of growth in big data is fast growing, management of the same data is challenging due to its characteristics which are varying. Their work involved 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 final 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 Hadoop framework can work well when the key/value pairs are used. The data which is less structured is able to handle quite a number of challenges in different problem domains. Their results showed that there are many forms of data which can be successfully analyzed after being transformed to key/value pairs.

Guangdeng Liao[11]performed an evaluation on 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 Gunther algorithm for the purpose of 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 which 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 objective function in order 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.

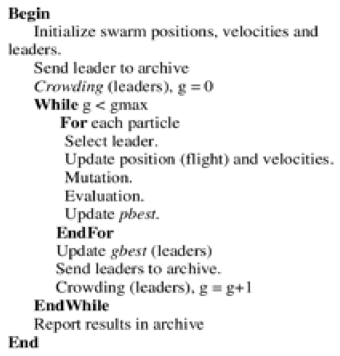
All the existing work has shown that changes in parameter values are done either before or after Hadoop MapReduce jobs are executed. Also the MOPSO concept has never been used for parameter tuning in big data processing. For this reason, we proposed PITCH, an algorithm which used the MOPSO concept to develop a multi objective function in order to optimize parameter settings while MapReduce jobs are running. MOPSO is emerging due to its usefulness in optimizing more than one objective function at the same time. In MOPSO instead of providing a single solution, a set of solutions known as pareto optimal sets are determined[19].

IV. Enhancing MOPSO

This section describes how PITCH employed the MOPSO concept in MapReduce parameter settings optimization. MOPSO, which is used to solve Multi Objective Problems was proposed by Coello Coello in 2004[15]. It is an extension of the PSO algorithm as introduced by Kennedy and Eberhart in 1995[15]. PSO was inspired by a social complex behavior of a flock of birds or a school of fish through simple actions of each member of the flock flying and performing repetitive tasks hence achieving a higher level of intelligence[18]. Swarm intelligence has been utilized through multi dimensional search spaces in order to find an optimum solution therefore effectively solving single objective problems[19]. However, PSO can only get a global optimum solution since the particles in a swarm have the same flying experience [24]. The algorithm's inability to solve multi objective problems is the reason why it was modified to a standard MOPSO.

A. PITCH algorithm

Multi Objective Optimization Problems (MOOP) have multiple objectives containing constraints which need to be satisfied using feasible solutions[18]found using the pareto optimality theory[12]. (a) shows PITCH algorithm in a 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
```

(a)

MOOP problems having a number of objectives together with a number of equality and inequality constraints are formulated in "(1)":

$$\begin{aligned} &\text{minimize } fn(X) && n = 0,1,2,3,\ldots,N \\ &\text{subject to} && h_k(x) = 0, \quad k = 0,1,2,3,\ldots,K, \\ & && g_l(x) <= 0, \quad j = 0,1,2,3,\ldots,L \end{aligned} \tag{1}$$

Where: fn(X) is the Objective function, x is a decision vector that represents solutions, N are a number of objectives, K is the number of equality constraints and L is number of inequality constraints.

In such a situation, the goal is to optimize n objective functions at the same time with the end result being good compromise of the solutions which represent the better trade-off between objectives[27]. The choice of parameters can highly impact the optimization performance therefore in order to implement PITCH, we first have to randomly initialize the positions of the particles in the search space to ensure uniform coverage as well as initializing velocity to zero as shown in "(2)".

$$V_i^{t+1} = wvi^t + c1r1(pbesti^t - Xi^t)c2r2(gbesti^t - Xi^t) \tag{2}$$

In every iteration, PITCH is updating the current position vector of the swarm by adding velocity 'V' using "(3)":

$$X_i^{t+1} = X_i^t + V_i^{t+1} \tag{3}$$

Where: r1 and r2 are cognitive and social randomization parameters which have random values between 0 and 1. c1 and c2 are local and global weights respectively which are acceleration constants. v denotes particle i's velocity and position. w is the inertia weight and t is an absolute time index. pbest is a personal archive which has the most recent non-dominated positions encountered by a particle in the past. gbestti the global best swarm flying experience is selected from an external archive. vit is the current external archive which stores non-dominated solutions known as pareto optimal sets. vit+1 is a new external archive which is then obtained from the current external archive vit and the population Pt.

The constriction factors K and L eliminate clamp speed therefore promoting convergence and these factors are expressed using the "(4)" and "(5)"

$$V_i^{t+1} = k{*}wvi^t + c1r1(pbesti^t - Xi^t) + c2r2(gbesti^t - Xi^t) \tag{4}$$

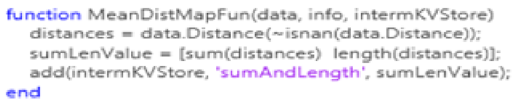
$$V_i^{t+1} = l{*}wvi^t + c1r1(pbesti^t - Xi^t) + c2r2(gbesti^t - Xi^t) \tag{5}$$

Just like the PSO algorithm, particles in PITCH share information and move towards the global best as well as their personal best memory. What makes it different from PSO is the use of more than one condition to define and conclude the best solution as either global or local[21]. In this case, all the non-dominated particles in the swarm are grouped together into a repository which is geographically based in order to maintain diversity[22]. It is in the repository that every particle selects its global best objective amongst its members[23]. A personal best particle is selected based on the level of domination as well as probabilistic rules since each particle has got its own swarm flying experience.

B. 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 makes use of a data store to process data in small blocks that fit into memory individually. Every block of code goes through the map phase, whereby data is formatted for processing then the intermediary data blocks go through the reduce phase, which puts together all the intermediate results in order to produce a final output[31]. The MapReduce functions are automatically called at the execution time[10] and for the functions to run properly, they must be able to accomplish some specific requirements. In the map phase, three input functions used are data, info, and intermKVStore. data and info are the output of the call which reads the function from the input datastore, whereby MapReduce execution is automatically done before every call to the map function. The intermKVStore object, the Intermediate Key Value Store is where the map function adds the key-value pairs. A simple example of a map function in (b) is:

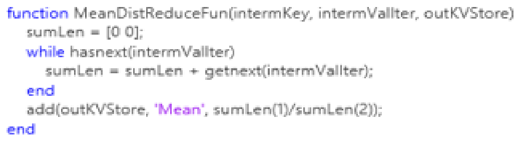
(b)



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

The first line is used to filter all the NaN values in a block of data. The second line's role is to create a vector which has two elements having a total distance besides the count of the block. The third line adds the vector of all the values in to the intermKVStore with the key, 'sumAndLength'. After executing this map function on all the data block d in ds, then the intermKVStore object will contain the total count for every block of data. Inputs of the reduce functions are the intermKey, intermValIter, and outKVStore. intermKey is used for the active key which is added by the map function. Every call made by MapReduce to the reduce function is used to specify a new unique key from all the keys in the intermediate KeyValueStore object. The intermValIter is the ValueIterator which associates with the intermKey. The object also contains all the values in the active key. Finally, the outKVStore, the final KeyValueStore is the object which the reduce function uses to add the key-value pairs. The output is then returned to the output datastore. A simple example of a reduce function in (c) :

(c)



```
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 runs as a loop through every distance beside count values of the intermValIter. It keeps the running total distance together with count after every pass[16]. When the loop is complete, the reduce function then calculates the overall mean using a simple division, then finally adds a single key to outKVStore.

One main advantage MapReduce is its ability to be extended for it to run in a number of computing environments[24]. However, the ability of MapReduce to perform calculations on large collections of data makes it not suitable for performing calculations on data sets which can be loaded directly into computer memory and analyzed with traditional techniques[5]. For this reason, it is important to use MapReduce for performing statistical or analytical calculations on datasets which do not fit in memory[33]. The mapreducer configuration function is used to change the execution environment using a Parallel Computing Toolbox, the MATLAB Parallel Server as well as the MATLAB Compiler[31]. This enables one to first start small by verifying the map and reduce functions before scaling up to run in larger computations.

C. *Running MapReduce application in Parallel*

The Parallel Computing Toolbox helps in speeding up MapReduce job execution using the full processing power of multicore computers in order to execute applications using a parallel pool of workers. The MATLAB Parallel Server enables one to run the same applications on remote computer clusters such as Hadoop using tall arrays which allow one to run big data applications that do not fit into the computer memory[24]. MATLAB compiler on the other hand enables one to create standalone MapReduce applications or deployable archives, which can be shared between applications or moved to production in Hadoop systems[28].

V. 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 PITCH algorithm.

A. *Environment*

Experiments were run on a HP super computer, an Intel(R) Core(TM) i5-8250U CPU @ 1.60 GHz 1.80 GHz with 8 GB of memory. In a serial environment, we run 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.

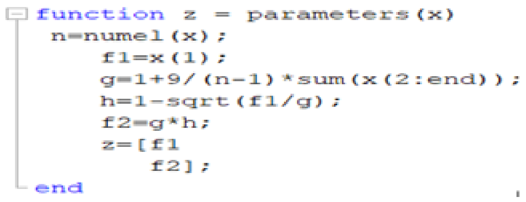
B. *performance evaluation*

For performance evaluation of the algorithm, we used the ZDT benchmark function[30], a test suite which got its name from its authors Zitzler, Deb and Thiele[29].

This algorithm shared the code for the particles movement and for performing evaluation in order to determine a tradeoff between two of our objectives which are reducing the quantity of the Random Access Memory(RAM) allocated to every task and attaining a better Input/Output(I/O) performance hence improving the execution time of every MapReduce job.

Starting with problem definition, we came up with the objective function together with the constraints to be minimized. The constraints in our case are the MapReduce configuration parameters which describe the number of decision variables. These are the I/O location of jobs, the I/O format of data as well as the classes which contain map and reduce functions. The motivation for the selected parameter values is based on the running records of MapReduce jobs, that can be either memory intensive or I/O intensive. In order to optimize the problem, we determined the tradeoff between two objectives which in or 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 satisfy the maximum memory constraint.
- We set I/O performance to be maximum and also 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 tradeoff between the two objectives using the minimization function in (d).

(d) 

```
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
```

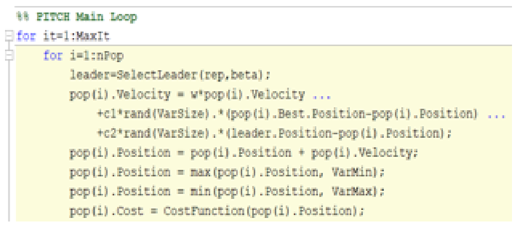
We equated the decision variables to 10, with a matrix size of 1 having lower bound and upper bound variables of 0 and 1 respectively. We also set 100 as the particle swarm size, 200 for the number of iterations and a repository size of 50. The values of c1 and c2 were set to 1.4269, the value of w was set to 0.49, and the values of r1 and r2 were selected randomly between 0 and 1 in every iteration.

The parameters used in PITH algorithm are presented in "Table I"

TABLE I
PARAMETERS USED IN PITCH ALGORITHM

Parameters	Value
Number of decision variables	10
Matrix size	1
Lower bound decision variables	0
Upper bound decision variables	1
Number of particles in the swarm	100
Number of iterations	200
Repository size	50
C1 and C2	1.4269
Inertia weight	1.49
r1 and r2	0 and 1

In every iteration, the new position of a particle was calculated using the number of unknown variables. 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 in (e) is the algorithm's main loop.

(e) 

```
%% 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);
```

C. Movement of particles in a pareto search

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

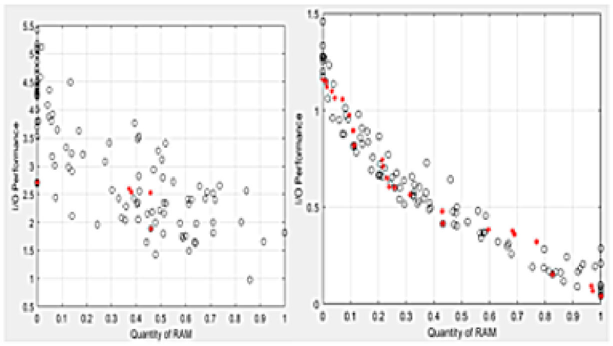


Fig. 3. Iteration 5 and Iteration 25

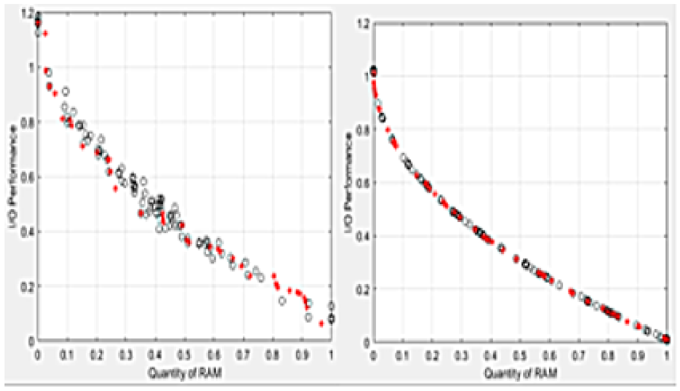


Fig. 4. Iteration 50 and Iteration 200

It is well known that multi objective algorithms have some limitations in relation to convergence and diversity[21]. In order to avoid these problems, we used the archiving process which introduced more convergence and the leader's selection method which provided diversity to the search. It used the procedure that limited the velocity of each particle by equality and inequality constraints k and l [26] 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].

VI. Experimental Results of PITCH

A. experiment 1

In the first experiment, 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.

B. 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.

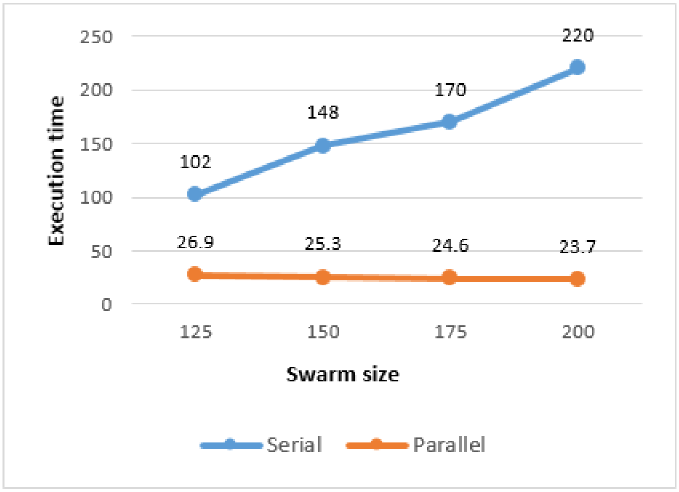


Fig. 5. Increase in swarm size

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

C. 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.

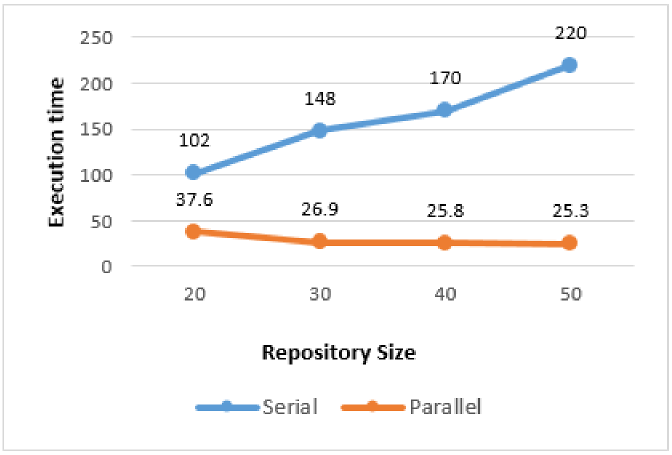


Fig. 6. Reducing the repository size

It can be seen from experiment 3 that there is still a gradual reduction of the execution time from 37.6 to 25.3 seconds.

D. 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.

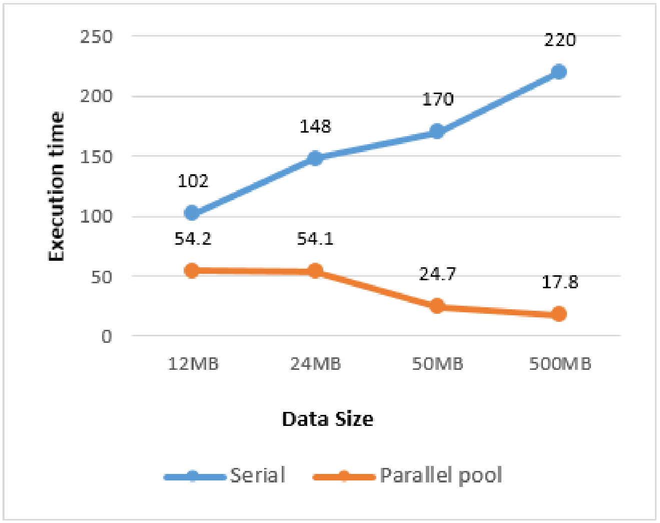


Fig. 7. Increase in the Data Size

From experiment 4, the execution time of the 12MB dataset was at 54.2 while 500MB was at 17.8 seconds respectively.

VII. DISCUSSION

During experiments, we noted that both serial and parallel implementations can be scaled easily and be used for larger swarm sizes in order 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. When we increased the number of particles in the swarm and reduced the repository size in every iteration, the performance of the algorithm showed a gradual improvement. Better still, when we scaled up and increased the data size, there was a significant improvement in the execution time.In all the experiments we kept the decision variables and the number of iterations constant. For the different data sizes which we used, we observed that execution time of a MapReduce job in a smaller dataset was of no great significance to the performance of the algorithm as compared to when we increased the data size. 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.

VIII. Recommendations

Based on the range of values tested, there is 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:

A. Number of particles in the swarm:

This is the population size of the algorithm and we recommend the population size to be from 100 to 200 particles.

B. 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. .

C. Repository size:

These 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.

D. Inertia weight and damping rate:

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

IX. conclusion

MOPSO concept is emerging due to its usefulness in optimizing more than one objective function at the same time. In PITCH, instead of using a single solution, a set of solutions known as pareto optimal sets are determined. Most real world problems including those in big data processing are multi objective hence involving simultaneous optimization in solving these problems is important. This can only be achieved by the use of algorithms. It is also significant to note that the way objective functions are measured is always competing and conflicting[21]. Therefore having competing objective functions is what gives rise to a set of multiple solutions instead of a single optimal solution. 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), 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 for a more diversified estimates of the pareto front, without losing convergence. Also, PITCH will be analyzed in other benchmarking problems which involve maximizing an objective function. There is more to do in areas which emphasize on PITCH parameters' self-adaptation, efficiency as well as application work in theoretical development.

X. 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 direcly to Thomas F. Coleman following his contribution towards constrained minimization functions and the use of the Optimization Toolbox™. Last but not least, Jomo Kenyatta University of Agriculture and Technology for permitting the authors to spend time and resources in the University.

References

[1] J. M. Cavanillas, E. Curry, and W. Wahlster, "New Horizons for a Data-Driven 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 Scale-out 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 multi-objective 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-to-MapReduce 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.