A

Dissertation

On

Parallelization of Bat Algorithm using Hadoop MapReduce

Submitted in Partial Fulfillment of the Requirement
For the Award of the Degree of

Master of Technology

in

Computer Science and Engineering

by

Sanchi Girotra
University Roll No. 2K12/CSE/29

Under the Esteemed Guidance of

Dr. Kapil Sharma
Associate Professor, Computer Engineering Department, DTU, Delhi

2012-2015

COMPUTER ENGINEERING DEPARTMENT

DELHI TECHNOLOGICAL UNIVERSITY

DELHI – 110042

ABSTRACT

Bat algorithm is among the most popular meta-heuristic algorithms for optimization. Traditional bat algorithm work on sequential approach which is not scalable for optimization problems dealing with BIG DATA such a scrawled documents, web request logs, commercial transaction information therefore parallelizing meta-heuristics to run on tens, hundreds or even thousands of machine to reduce runtime is required.

In this paper, we introduced two parallel models of BAT algorithm using MapReduce programming model proposed by Google. We used these two models for solving the Software development effort optimization problem.

The experiment is conducted using Apache Hadoop implementation of MapReduce on a cluster of 6 machines. These models can be used to solve problems with large search space, dimension and huge computation by simply adding more hardware resources to the cluster and without changing the proposed model code.

Index Terms - BAT algorithm, COCOMO Model, Effort Estimation, parallel algorithms, Apached Hadoop, MapReduce Model, Scalability, Big data.

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to all the people who have supported and encouraged me during the course of this project without which, this work could not have been accomplished.

First of all, I am very grateful to my project supervisor Dr. Kapil Sharma for providing the opportunity of carrying out this project under his guidance. I am deeply indebted to him for the support, advice and encouragement he provided without which the project could not have been a success. I am also grateful to Dr. OP Verma, HOD, Computer Science, DTU for her immense support. I am also thankful to my parents for being there for me at all times. Last but not the least, I am grateful to Delhi Technological University for providing the right resources and environment for this work to be carried out.

Sanchi Girotra
University Roll no: 2K12/CSE/29
M.Tech (Computer Science & Engineering)
Department of Computer Engineering
Delhi Technological University
Delhi – 110042

CERTIFICATE

This is to certify that the dissertation titled "**Parallelization of Bat Algorithm using Hadoop MapReduce**" is a bonafide record of work done at **Delhi Technological University** by **Sanchi Girotra, Roll No. 2K12/CSE/29** for partial fulfilment of the requirements for the degree of Master of Technology in Computer Science & Engineering. This project was carried out under my supervision and has not been submitted elsewhere, either in part or full, for the award of any other degree or diploma to the best of my knowledge and belief.

(Dr. Kapil Sharma)
Associate Professor & Project Guide
Department of Computer Engineering
Delhi Technological University

Date: _____

Table of Contents

List of Figures

List of Abbreviations

BA	Bat Algorithm
PSO	Particle Swarm Optimization
CS	Cuckoo Search
MPI	Message Passing Interface
PRS	Performance-based Reward Strategy
FBC	Fuzzy Bat Clustering
MOBA	Multi Objective Bat Algorithm
BBA	Binary Bat Algorithm
DLBA	Differential Operator and Lévy Flights Bat Algorithm
BAM	Bat Algorithm with Mutation
IBA	Improved Bat Algorithm
BPNN	Back-Propagation Neural Network
ACO	Ant Colony optimization
GA	Genetic Algorithm
PF	Particle Filter
APF	Annealed Particle Filter
TSP	Travelling Salesman Problem
HFS	Hybrid Flow Shop
RBF	Radial basis function
COCOMO	Constructive Cost Model
MPSO	Map - Particle Swarm Optimization
MRPSO	Map Reduce Particle Swarm Optimization
MRPGA	Map Reduce Parallel Genetic Algorithm
NASA	National Aeronautic and Space Administration
EM	Effort Multiplier
MRBA	Map Reduce Bat Algorithm

MBA	Map Bat Algorithm
KLOC	Thousands of Lines of Code
HDFS	Hadoop Distributed File System
MMRE	Mean Magnitude of Relative Error

CHAPTER 1
INTRODUCTION

Over the last few years, Meta-heuristic (discover solution by trial and error) approximation algorithms are widely used to solve many continuous and combinatorial optimization problems. It often finds good solution with less computation effort than exhaustive, iterative and simple heuristic methods. Some of the meta-heuristic algorithms are particle swam optimization (PSO), genetic algorithm (GA), cuckoo search (CS) etc. These algorithms are problem independent thus suits many optimization problems. Bat Algorithm is one of the swarm intelligence based nature inspired meta heuristic algorithm proposed by Yang [1] in 2010 and it is based on the echolocation behavior of bats to detect its prey, avoid obstacle and to find its dwell in caves. BA has been successfully applied on many continuous optimization problems of topology design, effort estimation, economic dispatch etc. and it has shown better convergence to optimal solution than other meta - heuristic algorithms.

But BA takes reasonable amount of time to solve problems involving large volumes of data (big data), involving huge fitness computation or having large number of dimensions. One solution to handle this would be to parallelize bat algorithm and scale it to large number of processors. But after program successfully parallelize, it must still needs to focus on communication of best bat across all the processors this can be done through shared memory which provides a global address space which parallel tasks can access asynchronously but it lacks locality of reference thus takes much time as compare to message passing interface (MPI) which provides an interface, protocol and semantic specifications to pass messages between parallel processes. These parallelized program still need to handle distribution of data and node failure explicitly. In order to deal with these concerns, Google designed a new abstraction called MapReduce that provides a parallel design pattern for simplifying application development for distributed environments i.e. allows expressing the simple computation in that design model with hiding messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. This abstraction is inspired by map and reduce primitives present in Lisp and many other functional languages. Apache Hadoop implementation of map reduce and HDFS have successfully parallelized many optimization algorithms like PSO, genetic algorithm and have

effectively reduced the running time of algorithm for complex problems like stock trading Strategy (PRS) [2], job shop scheduling problems Huang and Lin [3], k-mean clustering [4] etc.
This thesis makes the following research contributions:

1. We demonstrate a transformation of bat algorithms into the *map* and *reduce* primitives
2. We propose 2 models of parallel bat algorithm.
3. We implement these models on effort estimation problem and demonstrate its scalability to large problem sizes.

From this proposed work, many complex optimization problem based on BA can easily scaled only by increasing the no of hardware resources.
The rest of the thesis is organized as follows. Chapter 2 presents BA and MapReduce in literature. Chapter 3 discusses the research methodology used for proposed work. In chapter 4 we discuss how bat algorithm can be parallelized using the Map Reduce model. Chapter 5 provides implementation, environment details and evaluates these two models on Hadoop MapReduce cluster. Then we conclude in chapter 6.

CHAPTER 2
LITERATURE REVIEW

In 2010, [1] proposed Bat Algorithm (BA), which is a new meta(high level)-heuristic algorithm for continuous problem optimization. From then BA has been employed on various optimization problems and many variants of BA have been proposed.

2.1 VARIANTS OF BAT ALGORITHM

In order to improve the solution of the problem, many variants of Bat Algorithm have been proposed, which resulted in better performance than original algorithm.

1. Fuzzy Bat Clustering (FBC) algorithm : Khan, et al. [5] combined fuzzy logic with bat algorithm and presented a variant of bat algorithm for clustering of ergonomic workspace.

2. Multi Objective Bat Algorithm (MOBA) : In 2011, Xin-She [6] proposed an extended version of bat algorithm to solve multiobjective optimisation problems. It used Pareto optimality and weighted sum to deal with multiple objective functions.

3. K-Means Bat Algorithm (KMBA) : In KMBA, Komarasamy and Wahi [7] used bat algorithm to calculate the cluster center and KM algorithm to form clusters which provides efficient clustering.

4. Chaotic Levy Flight Bat Algorithm : Lin, et al. [8] uses chaotic(map) variables in place of random numbers and levy flight for random walk for estimation of parameters in dynamic biological system.

5. Binary bat algorithm (BBA): Nakamura, et al. [9] developed a discrete version of bat algorithm for solving feature selection problem.

6. Improved Bat Algorithm with Doppler Effect : U., et al. [10] modified BA to take into account the Doppler shift i.e. change in frequency of echoes when bats are approaching their prey. In this case, bats changes their frequency of signal emission to make echoes centered

within a narrow frequency range where they have very sensitive hearing. It has been tested on 4 benchmark functions.

7. Differential Operator and Lévy Flights Bat Algorithm(DLBA): Xie, et al. [11] modified bat algorithm by introducing Differential Operator as a mutator to increase the convergence speed of algorithm and Lévy Flights for random walk to avoid pre mature convergence.

8. Bat algorithm with mutation (BAM) : Zhang and Wang [12] modified bat algorithm to mutate bat in Equation 4 to solve image matching problem.

9. Hybrid Bat Algorithm : Jr., et al. [13] uses differential evolution ”DE/rand/1/bin” strategy to find the neighborhood solution in order to get good results for higher-dimensional problems.

10. Adaptive Bat Algorithm : Wang, et al. [14] improved bat algorithm to avoid premature convergence problem and used dynamic / adaptive strategy to update its speed and direction.

11. Improved Bat Algorithm (IBA): Yilmaz and Kucuksille [15] employed Inertia Weigh Factor, Adaptive Frequency Modification, Scout Bee Modification to balance exploration over exploitation in BA.

12. Enhanced Bat Algorithm : Kaveh and Zakian [16] presents a variant of bat algorithm by limiting the local search by a Θ scale factor thus improve the exploration capability of BA.

13. Modified Bat Algorithm: Yılmaz, et al. [17] balance exploration and exploitation in BA by assigning separate loudness A and pulse rate r for each dimension of the solution.

14. Compact Bat Algorithm: Dao, et al. [18] addressed the problem of limited hardware resources such as memory for solving numerical optimization problem by using probability vector in place of population.

15. Parallelized Bat Algorithm with a Communication Strategy : In 2014, Tsai, et al. [19] defined communication strategy to parallelize original Bat Algorithm.

16. Evolved Bat Algorithm : Dao, et al. [20] proposed a new bat algorithm to solve Economic Dispatch problem whose results are found better than Genetic Algorithm.

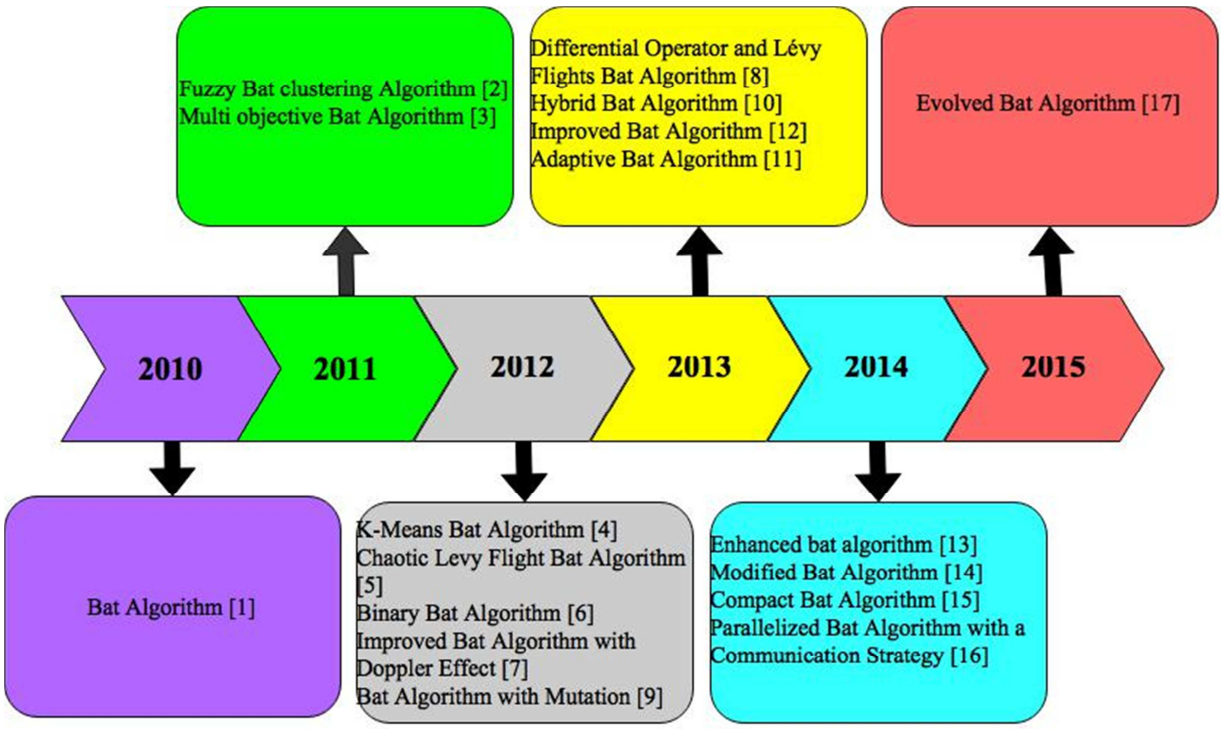

Figure 2.1: Timeline of Bat Algorithm

Many researchers have united the BA with other algorithms to solve optimization problems like, Wang and Guo [21] merged BA with Harmony Search for Global Numerical Optimization, Nawi, et al. [22] combined BA with Back-propagation neural network (BPNN) algorithm to solve the local minima problem in gradient descent trajectory and to increase the convergence rate.

2.2APPLICATIONS OF BAT ALGORITHM

Meta-heuristic technique are problem-independent and provide general purpose strategy for search which makes BA applicable for different continuous optimization problems and in the last 5 years, BA has successfully provided optimized solutions of various problems:

1. Bora, et al. [23] (2012) optimized of 5 design parameters in order to design Brushless DC Wheel Motor having best efficiency η with satisfying 6 constraints.

2. Damodaram and Valarmathi [24] (2012) employed BA to minimize the error rate of predicting the e-banking phishing websites in which BA outperformed ACO and PSO.

3. In 2013, Ramesh, et al. [25] used BA to solve economic dispatch problem by optimizing fuel cost coefficients of 6 generators in order to minimize total power generation cost with satisfying all units and operational constraints of the power system.

4. Marichelvam and Prabaharam [26] (2012) minimized makespan (the maximum completion time) and mean flow time by selecting optimal hybrid flow shop schedule in HFS with multiple machines and BA found more efficient than Genetic Algorithm and PSO.

5. M., et al. [27] (2012) estimated full body human pose by minimizing the distance or maximize the similarity between generic model of human body and the cloud of points. It performed better than PSO, Particle Filter (PF) and Annealed Particle Filter (APF).

6. Yang, et al. [28] (2012) used BA in topology optimization i.e. find best geometric configuration so as to achieve certain objective. For E.g. distribute two different materials so as to maximize the temperature difference.

7. Tamiru and Hashim [29] combined bat algorithm with fuzzy logic for gas turbine model identification after exergy changes.

8. Faritha Banu and Chandrasekar [30] explored Modified Bat algorithm to detect duplicate records in Cora Bibliographic data set in which BA performed better than Genetic Programming.

9. Srivastava, et al. [31] (2014) proposed test effort estimation model using BA.

10. Saji, et al. [32] (2014) redefined Bat Algorithm for solving NP hard TS Problem.

11. Gupta and Sharma [33] estimated software development effort by optimizing COCOMO coefficients.
12. Djelloul, et al. [34] found optimal assignment of colors in the graph subject to certain constraints i.e. solved Graph Coloring Problem.

13. Hassan, et al. [35] detected optimal community of nodes in the network having better internal connectivity than external using discrete bat algorithm.

2.3APPLICATIONS OF MAPREDUCE MODEL

In 2004, Google's Dean and Ghemawat [36] published a white paper to parallelize large data on clusters by using map reduce model. After that MapReduce became widely-used parallel programming model and was employed on various large processing application like relation data processing [37], machine learning [38], scientific analysis [39] etc.

Nowadays, heuristic algorithms have started using Hadoop Map Reduce Model to scale data intensive problems such as:

1. Particle Swarm optimization Algorithm (PSO) : McNabb, et al. [40] successfully parallelized PSO on a MapReduce framework and evaluated it on RBF Network Training function and sphere function then in 2014 Wang, et al. [2] proposed MPSO (Map-PSO) and applied this on Performance-based Reward Strategy in stock markets which took less running time then sequential and MRPSO.

2. Genetic Algorithm (GA) : Jin, et al. [41] used MapReduce model to parallelize GA and proposed MRPGA. Verma, et al. [42] presented parallel model of GA for data-intensive computing and tested it on one max problem. Keco and Subasi [43] proposed Model 2 and compared it with model 1 for same one max problem implementation. Then these models were used for various problems like Job Shop Scheduling Problem [3], automatic generation of JUnit Test Suites [44] etc.

3. Cuckoo Search : Lin, et al. [45] scaled Modified Cuckoo Search using MapReduce Architecture and evaluated it on Griewank function, Rastrigrin function, Rosenbrock function and Sphere function.

CHAPTER 3
RESEARCH METHOLOGY

In this research, we have studied Bat algorithm, Hadoop Map Reduce Model, COCOMO II and have successfully used these methods to propose a new parallel Bat Algorithm using MapReduce Programming Model and have evaluated it on Effort Estimation problem using NASA 63 Dataset.

3.1 BAT ALGORITHM

Bat Algorithms (BA), is swarm intelligence based meta-heuristic optimization algorithm proposed by Xin-She Yang in 2010. BA finds the solution in the search space by exploiting the echolocation behavior of bats in which they locate their prey by emitting a short - loud sound pulse and listen for the echo that bounces back from the surrounding obstacle then they use the time delay from the emission of sound pulse to perception of the echo and the time difference between their two ears and also the loudness variations (signal intensity) of the echo's to get a view about surrounding and prey size as shown in Figure 3.1.

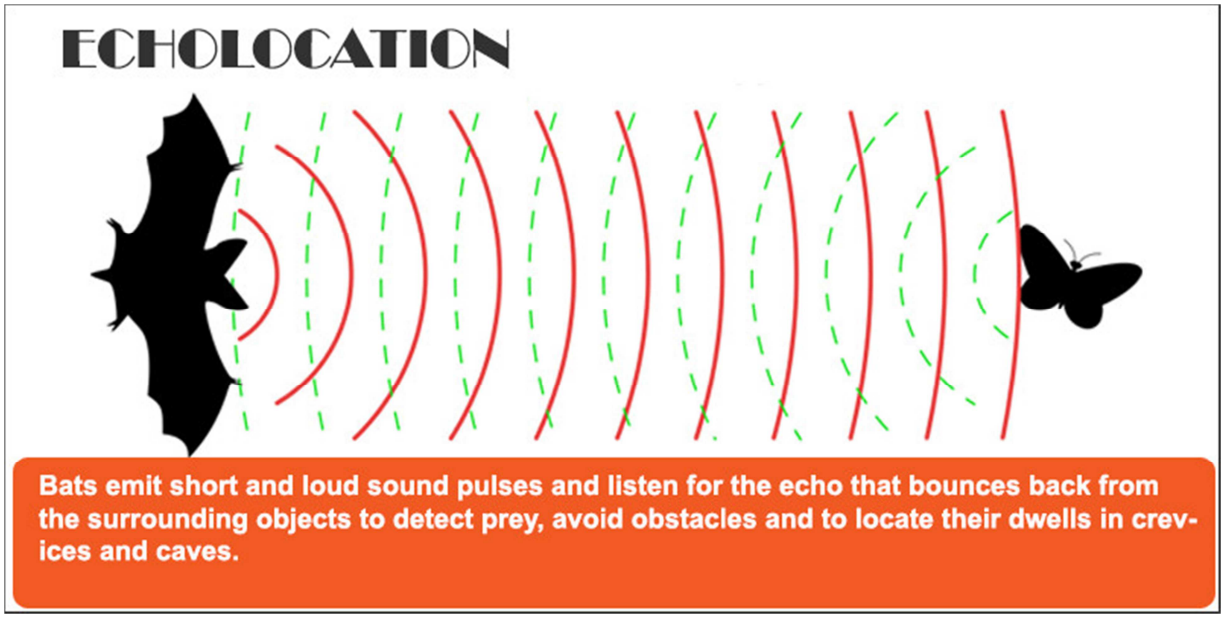

Figure 3.1: Bat using echolocation to locate its prey

BA is a continuous meta-heuristic optimization algorithm used to select best real valued input from defined set of options in order to maximize or minimize a function. It only guarantee approximate (not approximation) solution to the problem.

In addition to this, heuristic Algorithms have two main attributes: exploration (also known as diversification) and exploitation (also known as intensification).

Exploration is investigation of larger portion of search space so as to detect global optimum point (global search ability) while **Exploitation** is investigation of limited (promising) region of the search space in order to improve or refine the promising previous best solutions.

A good response of an algorithm depends on well-balance of these components i.e. in the case of little exploration and massive exploitation, the algorithm can get confined into local optimum points. Whereas too much exploration but less exploitation will make it difficult for algorithm to converge on optimal point thus slow down the search performance. But it is beneficial to first explore and then exploit as we get good solution in the beginning then we can refine them to find local best solutions. And in BA exploration and exploitation is performed by varying the pulse emission rates r and loudness A_0.

Yang used three generalized rules for Bat Algorithm:

1. All bats use echolocation to sense distance and they also know the difference between food/prey and background barriers in some magical way.
2. Bats fly randomly with velocity v_i at position x_i with a frequency f_{min}, varying wavelength and loudness A_0 to search for prey. They can automatically adjust the wavelength (or frequency) of their emitted pulses and adjust the rate of pulse emission r [0, 1] depending on the proximity of their target.
3. Although the loudness can vary in many ways, we assume that the loudness varies from a large (positive) A_0 to a minimum constant value A_{min}.

BAT Algorithm

1: Initialize the population of N bats randomly and each bat corresponds to a potential solution to the given problem using Equation (1)

2: Define loudness A_i, pulse frequency $f_i \in [f_{min}, f_{max}]$

3: Define the initial velocities v_i (i=1, 2, 3...N)

4: Set pulse rate r$_i$.

5: **while** t < Max number of Generations **do**

6: **for** bat **in** population **do**

7: Generate new solution by adjusting frequency, updating velocities using Equation [2-4]

8: **if (**rand > r$_i$**) then**

9: Select a solution among the best solutions.

10: Generate a local solution around the selected best solution using Equation (5)

11: **end if**

12: Generate a new solution by flying randomly

13: **if (**rand < A_i **&** f (x_i) < f (x^*)**) then**

14: Accept the new solution

15: Increase r$_i$ and reduce A_i using Equation [6, 7]

16: **end if**

17: **end for**

18: Rank the bats and find the current best x^*

19: **end while**

20: Post process results and visualization

In this algorithm, the bat behaviour of finding the best location is captured to optimize the fitness function of the problem to be solved. It consists of the following steps:

Initialization of Bat Population (lines 1 - 4) –

In first iteration, N bats are randomly spread over the search space as they have no idea where the prey is located thus population is randomly generated for each dimension d with default values for frequency, velocity, pulse emission rate and loudness.

$$x_{ij} = x_{\min j} + rand\,(0,1)\ (x_{\max j} - x_{\min j}) \qquad (1)$$

Where $i = 1, 2 \ldots N, j = 1, 2 \ldots d$, $x_{\min j}$ and $x_{\max j}$ are lower and upper boundaries for dimension j respectively.

Update Process of Frequency, Velocity and Solution (line 7) -

Each bat emits sound pulses of random frequency dispersed between [f_{min}, f_{max}] and this frequency controls the speed and new position of bat using below equations:

$$f_i = f_{\min} + (f_{\max} - f_{\min})\beta \qquad (2)$$

$$v_i^t = v_i^{t-1} + (x_i^{t-1} - x^*) f_i \qquad (3)$$

$$x_i^t = x_i^{t-1} + v_i^t \qquad (4)$$

Where β denotes a randomly generated number with in the interval [0, 1] and x^* is the current global best solution which is located after comparing all the solutions. Upper and lower bounds of frequency are chosen such that it is comparable to the size of search space of that variable.

Local search (lines 8-11) –

Random walk is used in order to exploit a position around the best solution.

$$x_{new} = x_{old} + \varepsilon A^t \qquad (5)$$

Here x_{old} is selected best solution from current best solutions, ε denotes a random number within the interval [-1, 1], while A^t is the average loudness of all the bats at this time step.

Updating Loudness and Pulse Emission Rate (lines 12-16) -

As the bat approach their prey, loudness of emitted sound pulse decreases and pulse emission rate increases so if new solution is improved than previous solution then factors like loudness and pulse rate needs to be updated. For simplicity, we can also use $A_0 = 1$ and $A_{\min} = 0$, assuming $A_{\min} = 0$ means that a bat has just found the prey and temporarily stop emitting any sound.

$$A_i^{t+1} = \alpha A_i^t \qquad (6)$$

$$r_i^{t+1} = r_i^0 (1 - e^{-\gamma t}) \qquad (7)$$

Where α and γ are constants. r_i^0 & A_i^0 consist of random values and $r_i^0 \in [0, 1]$ and $A_i^0 \in [1, 2]$

Find the current best solution (line 18) –

Find the current best bat by comparing last best fitness value with improved bat fitness if it found better then update the best bat solution.

In bat algorithm, if the new solutions are generated from equation [2-4] then exploration is good and if solution is generated by Equation 5 then exploitation is high. So the pulse rate r controls the balance between exploration and exploitation but the possibility of rand > r$_i$ to be true is during the initial iterations of bat algorithm that means initially exploitation will be good and during following iterations exploration will be good. But it is advantageous to explore first and exploit later to prevent the solution from getting trapped into local minimum. In order to solve this issue, researchers have proposed various variants of bat algorithm. For e.g. FBC redefined velocity v_i^t and position x_i^t in Equation [3, 4] in which each bat is not only considering the best solution found by all the bats but also its own local preference (x^{lbest}).

$$v_i^t = v_i^{t-1} + (x_i^{t-1} - x^*) f_i + (x_i^{t-1} - x^{lbest}) f_i \quad (8)$$

And Chaotic Levy Flight Bat Algorithm used Levy Flight for random walk thus modified neighborhood solution.

$$x_{new} = x_{old} + C_s \otimes Levy(\lambda) \qquad (9)$$

Here, the product $\otimes$ means entry-wise multiplications.

Improved bat algorithm with Doppler Effect introduced a new frequency equation to deal with Doppler Effect of bat's echo

$$f_i^t = \frac{c - v_i}{c + v_i} f_i^{t-1} \qquad (10)$$

Where f is bat frequency and c iss speed of sound which is 340 m/s.

3.2 MAPREDUCE MODEL

Map Reduce is programming model proposed by Google - Dean and Ghemawat [36] MapReduce is a simple model designed for processing large volumes of data in parallel by dividing the program into a set of high-level *map* and *reduce* functions which are inspired by functional languages like Lisp, List etc. This Model exploits parallelism by splitting the input

data into chunks which are processed by the map and reduce tasks in a completely parallel manner.

The input to MapReduce Model is in the form of (key; value) pairs and operation on these set of pairs occur in three stages: the map stage, the shuffle stage and the reduce stage.

A. Map Stage

In the map stage, the map() function takes as input a single (key; value) pair and produces as output any number of intermediate (key; value) pairs. It is crucial that the map operation is stateless - that is, it operates on one pair at a time. This allows for easy parallelization as different inputs for the map can be processed in different machines.

MAP: (K1, V1) => LIST (K2, V2)

B. Shuffle Stage

In shuffle phase, all of the map's output (key; value) pairs are sorted and value lists are created for each individual key which is then send to the reduce() function. This occurs automatically without programmer intervention.

C. Reduce Stage

In the reduce stage, each reduce () function takes all of the values associated with a single key k, and outputs a multi set of (key; value) pairs with the same key k (often "reduced" in length from the original list of values).This highlights one of the sequential aspects of Map/Reduce computation i.e. all of the maps need to finish before the reduce stage can begin.

REDUCE: (K2, list (V2)) => list (V2)

And in the reduce step, the parallelism is exploited by observing that reducers operating on different keys can be executed simultaneously.

Apache Hadoop is an open source software framework for distributed processing which have implemented google MapReduce and Google file system in its modules.

3.2.1 EXAMPLE OF MAPREDUCE MODEL

Example of Map Reduce Model is shown in Figure 3.2. This **WordCount** program counts the number of occurrences of each word in a given input data. The input data file is given to the Map Reduce job which splits the data into several files and sends it to different map tasks(mappers)

for processing. Each mapper processes the file, line by line and generates input key-value pair for map function. The map() function splits each line into words and sends output key-value pair as word:1 where key is the word and 1 is its count. Then sort and shuffle function combines mapper key-value output and generates count list for each word. Finally, the reducer sum the count value of each word and write the final count of occurrence of each word as output.

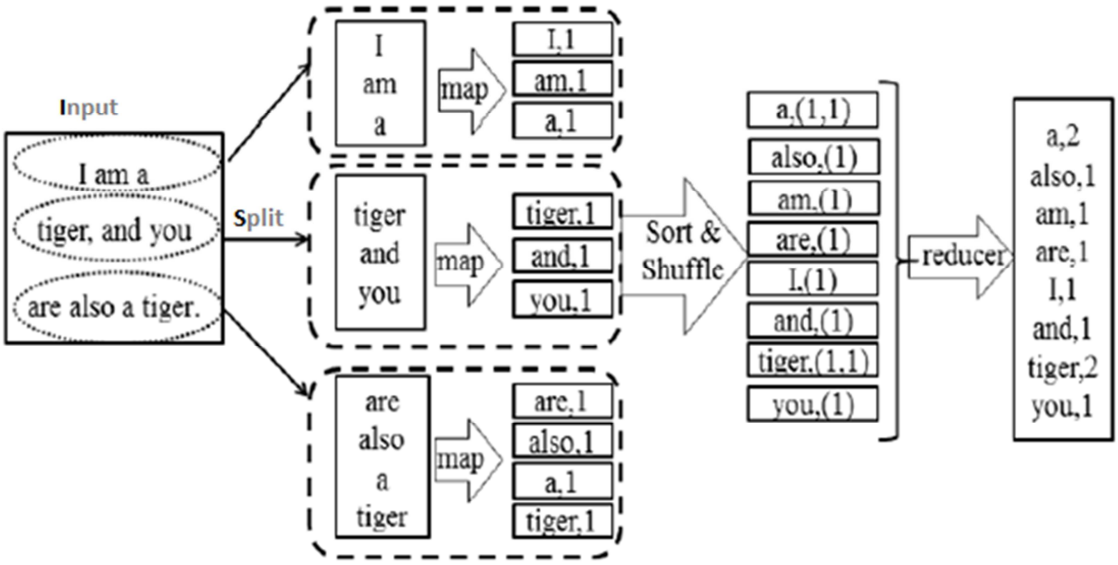

Figure 3.2: Example of MapReduce Model

3.2.2 HADOOP MAPREDUCE ARCHITECTURE

As shown in Figure 3.3, the major components of Hadoop based cluster are client machine, master node, slave node. Each master node performs parallel processing and handling of data in HDFS which have instances of name node (handles data distribution across data nodes), job tracker (handles instances of trasktracker in each slave node) and secondary name node (backup of namenode). Each slave node performs map/reduce task so have data node and task tracker instances.

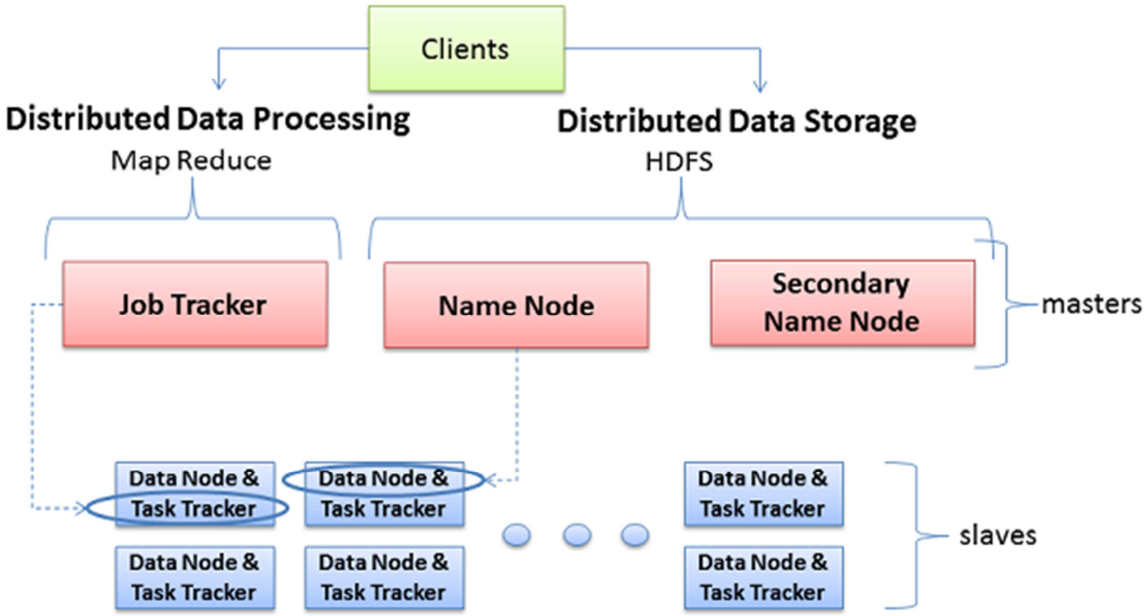

Figure 3.3: Hadoop Server Roles

The basic architectural flow of Hadoop MapReduce is shown in figure 3.4 and can be explained in steps as:

1. Client writes input files into HDFS.
2. Client submits the job consisting of following information: inputformat, outputformat, mapper class, reducer class, partitioner class, combiner class, input path, output path etc.
3. Hadoop Client breaks the files into inputsplits depending upon the block size.
4. Client by consulting the name node store each block in one or more data nodes which is controlled by *dfs.replication parameter*.
5. Job Tracker creates task instances for mappers depending upon no of splits i.e. if 5 split then 5 mapper tasks will be created.
6. Job Tracker distributes the tasks across the data nodes and number of tasks assigned to each node depends on the hardware configuration of the system.
7. Defined Inputformat provides record reader object to read the data in key – value pairs.
8. Record Reader passes each key-value pair in the input split to the user defined map () function.

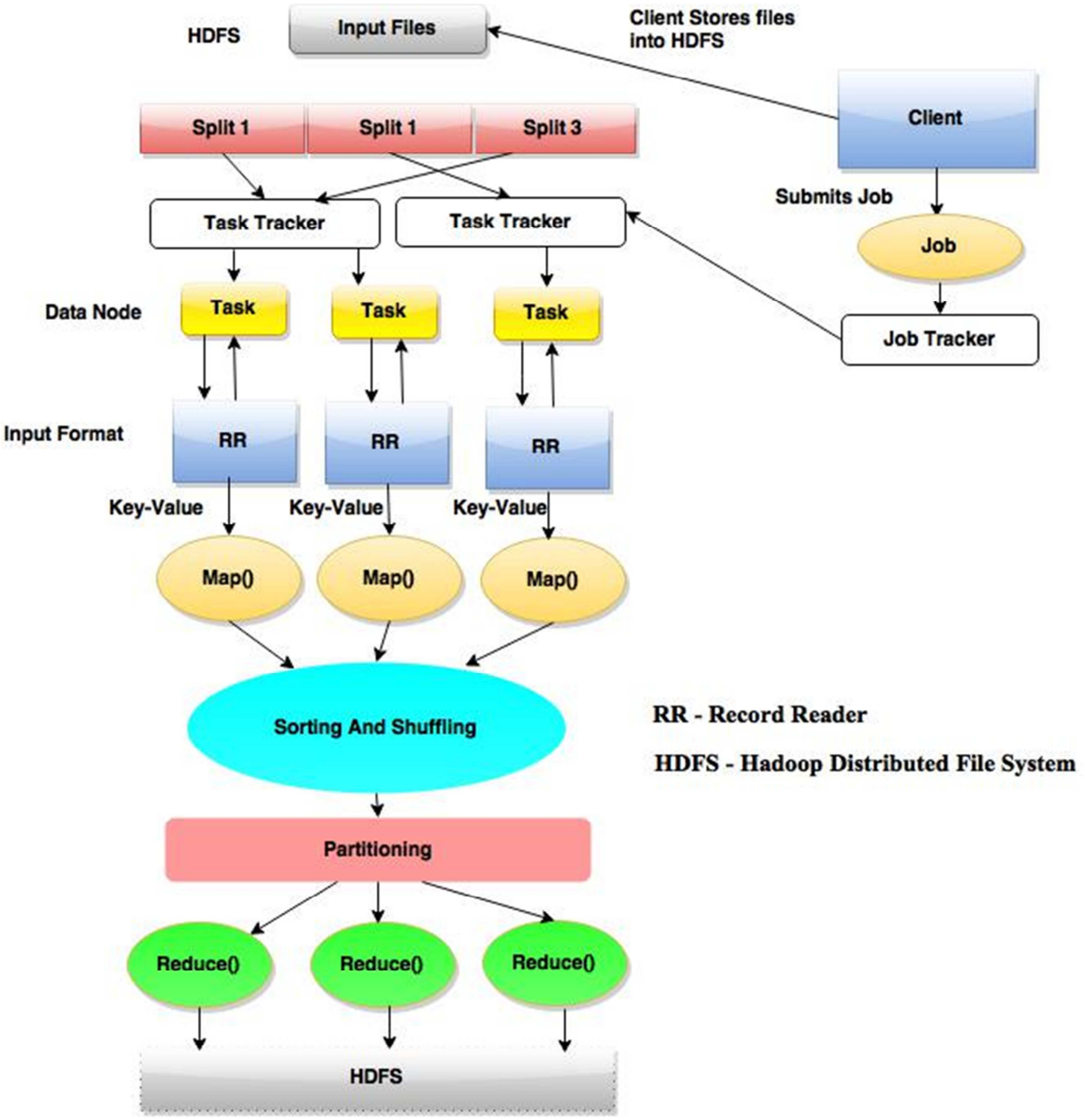

Figure 3.4: Hadoop MapReduce Architecture

9. Map () function synthetize each key-value pair into new object and passes it to the partitioner and shuffler for sorting and grouping them.
10. These new grouped data is again distributed across the data nodes.
11. Job Tracker creates reduce tasks based on mapred.reduce.tasks parameter or by default creates 1 instance of reduce task and deploy it on data nodes.
12. Each key – value list in the partition is assigned to a single Reduce task.

13. Then the user defined reduce () function will write final key value pair as output in HDFS.

3.3 COCOMO II MODEL

The **Constructive Cost Model** (**COCOMO**) is a regression-based software cost estimation model developed by Barry W. Boehm in 1981 which is also called in references as COCOMO 81. COCOMO-81 is said to be the best known, best documented and it reflects most software development practices on that time.

One of the problems with the use of COCOMO I today is that it does not support modern software development processes like desktop development, code reusability, rapid-development, object-oriented approaches etc. Therefore, in 1997, Boehm developed the COCOMO II for estimating modern software development projects.

COCOMO consists of a hierarchy of three increasingly detailed and accurate forms:

1. **Basic COCOMO** computes software development effort (and cost) as a function of program size and it holds up until a certain point, usually for projects that can be reasonably accomplished by small teams of two or three people.
2. **Intermediate COCOMO** provides more accurate estimates by taking into account software development environment through 15 cost drivers.
3. **Detailed COCOMO** computes effort as a function of program size and a set of cost drivers given according to each phase of software life cycle i.e. analysis and design of the software engineering Process.

The COCOMO estimated software effort is given by below equation and is measured in calendar months

$$Effort(Person-month) = a \times [LOC]^b \times (EM1 \times EM2 \times EM3 \times \ldots \times EM15) \quad (11)$$

Here the coefficient "a" is known as productivity coefficient and the coefficient "b" is the scale factor. They are based on the different modes of project as given in table 3.1

Software Project	**Project Size**	**a**	**b**
Organic	Less than 50 KLOC	3	1.1
Semi-detached	50 – 300 KLOC	3	1.1

Embedded	Over 300 KLOC	3	1.2

Table 3.1: Software Project Mode

And EMi are effort multipliers (Cost drivers) which have up to six levels of rating: Very Low, Low, Nominal, High, Very High, and Extra High. Each rating has a corresponding real number based upon the factor and the degree to which the factor can influence productivity as given in Table 3.2

Cost Drivers	**Rating**					
	Very Low	**Low**	**Nominal**	**High**	**Very High**	**Extra High**
acap	1.46	1.19	1	0.86	0.71	-
pcap	1.42.	1.17	1	0.86	0.7	-
aexp	1.29	1.13	1	0.91	0.82	-
modp	1.24.	1.1	1	0.91	0.82	-
tool	1.24	1.1	1	0.91	0.83	-
vexp	1.21	1.1	1	0.9	-	-
lexp	1.14	1.07	1	0.95	-	-
sced	1.23	1.08	1	1.04	1.1	-
stor	-	-	1	1.06	1.21	1.56
data	-	0.94	1	1.08	1.16	-
time	-	-	1	1.11	1.3	1.66
turn	-	0.87	1	1.07	1.15	-
virt	-	0.87	1	1.15	1.3	-
rely	0.75	0.88	1	1.15	1.4	-
cplx	0.7	0.85	1	1.15	1.3	1.65

Table 3.2: Software Cost Drivers

These effort multipliers fall into three groups: those that are positively correlated to more effort, those that are negatively correlated to more effort and the third group containing just schedule information.

3.4 NASA 63 DATA SET

The proposed algorithm is evaluated on 63 NASA projects from different centers from the years of 1971 to 1987. As shown in table 3.3, this dataset consist of development mode(embedded,

organic, semidetached), EAF of 15 cost drivers, size of each project in kilo source line of code and actual effort.

Mode	EAF	LOC	Effort
Embedded	2.28811	113	2040
Embedded	0.53128	6.9	8
Embedded	5.50991	22	1075
Embedded	2.01377	30	423
Embedded	1.73015	29	321
Embedded	1.73015	32	218
Embedded	0.93626	37	201
Embedded	4.94502	3	60
Embedded	3.04353	3.9	61
Embedded	2.37496	6.1	40
Embedded	1.94746	3.6	9
Embedded	3.27117	320	11400
Embedded	3.48791	299	6400
Embedded	0.84607	252	2455
Embedded	0.96816	118	724
Embedded	0.7025	90	453
Embedded	1.1639	38	523
Embedded	0.95249	48	387
Embedded	0.99439	1.98	5.9
Embedded	0.56909	390	702
Embedded	2.30187	42	605
Embedded	1.47674	23	230
Embedded	0.30168	91	156
Embedded	0.3401	6.3	18
Embedded	2.66087	27	958
Embedded	3.30632	17	237
Embedded	1.05362	9.1	38
Organic	0.32046	132	243
Organic	0.99814	60	240
Organic	0.65617	16	33
Organic	1.86504	4	43
Organic	0.85243	25	79
Organic	1.6573	9.4	88
Organic	0.68887	15	55
Organic	0.37224	60	47

Organic	0.3588	15	12
Organic	0.38774	6.2	8
Organic	0.9649	3	8
Organic	0.25445	5.3	6
Organic	0.58734	45.5	45
Organic	1.06981	28.6	83
Organic	1.33662	30.6	87
Organic	0.87268	35	106
Organic	0.82473	73	126
Organic	1.28037	24	176
Organic	2.30456	10	122
Organic	1.15428	5.3	14
Organic	0.77736	4.4	20
Organic	1.08961	25	130
Organic	1.00697	23	70
Organic	2.12549	6.7	57
Organic	0.38613	10	15
Semidetached	0.84227	293	1600
Semidetached	0.67554	1150	6600
Semidetached	0.90842	77	539
Semidetached	2.81069	13	98
Semidetached	0.99439	2.14	7.3
Semidetached	3.43917	62	1063
Semidetached	2.17879	13	82
Semidetached	0.38067	23	36
Semidetached	0.75808	464	1272
Semidetached	1.37602	8.2	41
Semidetached	0.4466	28	50

Table 3.3: NASA 63 Dataset

CHAPTER 4

PARALLEL BAT ALGORITHM USING MAPREDUCE MODEL

In each iteration of Bat algorithm, all bats finds a new location either by flying randomly using equation 5 or by adjusting their frequency and updating their velocity, location using equation [2-4]. Each bat then evaluates the fitness of new solution and accordingly move to new position if found better than the current position.

In this algorithm, steps that can be parallelized i.e. can be executed independently are:

- Each bat updating its position.
- Iterations can be executed in parallel.

But iterations can't be parallelized as population of bat should improve from generation to generation and initial iteration results should be utilized in the next iterations.

Based on above idea, to transform BA into map and reduce primitives following issues need to be considered:

1. Determine the input to MapReduce Model.
2. Determine the jobs of mapper and reducer tasks.
3. Exchange information between parallel map tasks.

4.1 INPUT REPRESENTATION

We provide large initial random population of bats as input to MapReduce framework which then split them into chunks and distribute them across the map tasks. Each bat in input population is represented by key/value pairs:

K1: Set of string representing individual bat

V1: fitness

Here the key consist of set of attributes representing bat like position, frequency and velocity whereas value is the fitness of that bat. For e.g. in our software effort optimization problem we have:

Coefficient a; Coefficient b; Frequency; Velocity a; Velocity b**:** Fitness

And gbest bat of population is of following form:

Coefficient a; Coefficient b: Fitness

4.2 EXCHANGE OF INFORMATION BETWEEN BATS

In bat algorithm, bats use the shared current best bat in the updating process but in Parallel Bat Algorithm we share generation best bat which can be implemented by two approaches:

1. Map Reduce Bat Algorithm (MRBA) :

In MRBA, initially we share the gbest across all the map tasks and each bat uses this gbest as x^* in Equation [3, 5]. Output of map tasks are combined and sent to a single Reducer (as used same key for all the improved bats). Reduce task sorts all the improved bats and output gbest bat from them which is used for following iterations as shown in figure 4.1.

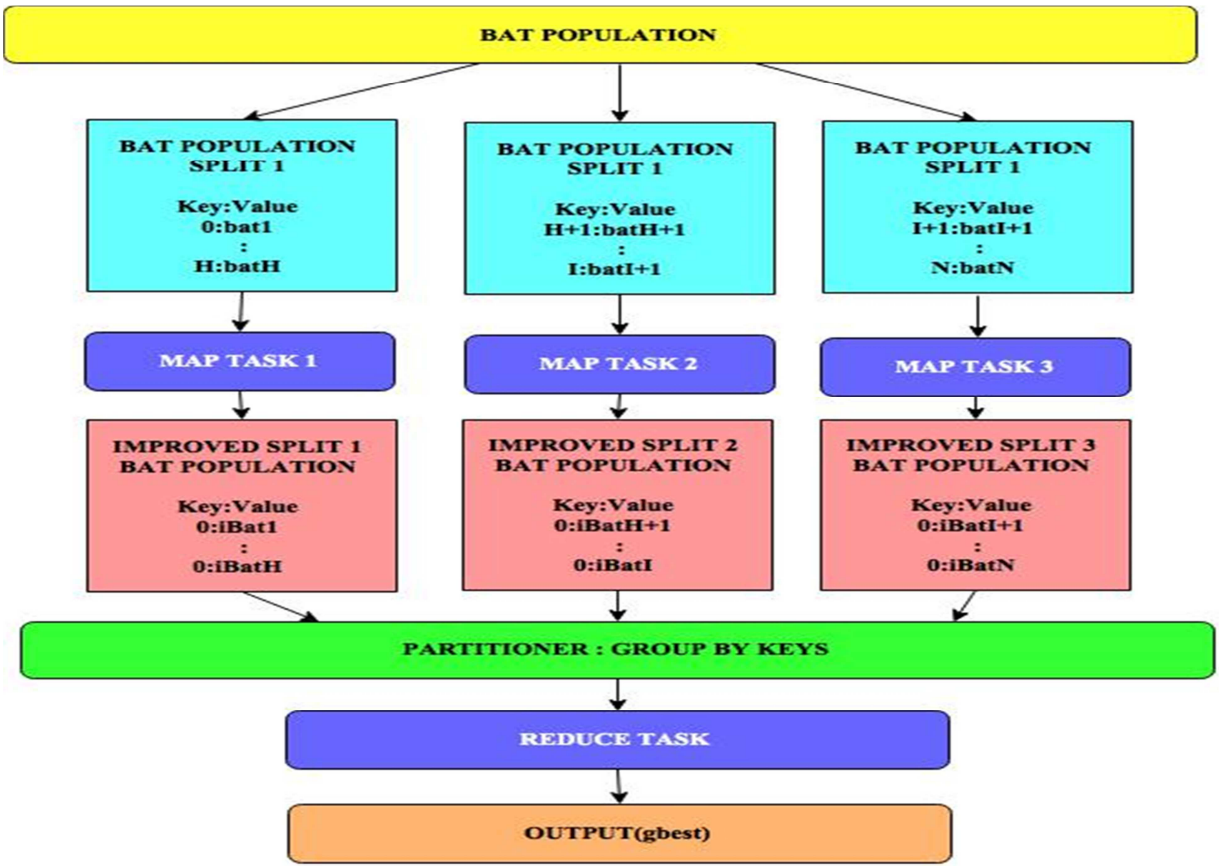

Figure 4.1: Operation phases in Map Reduce Bat Algorithm (MRBA)

2. Map Bat Algorithm (MBA)

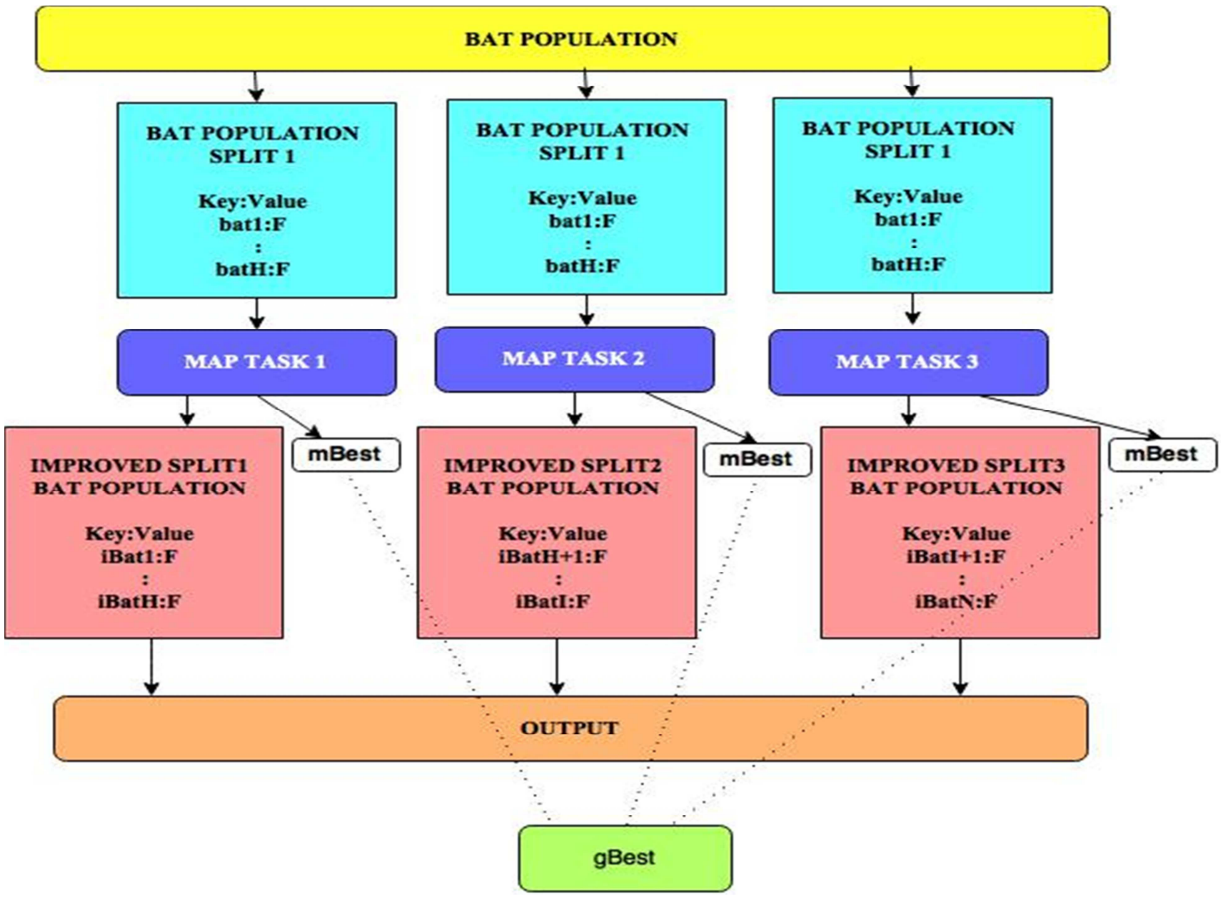

Figure 4.2: Operation phases in Mapper Bat Algorithm (MBA)

In MBA i.e. only map approach, initially we share the population gbest across all the map tasks. Each map task has a mapper level best bat which is initialized with this gbest. This mbest is updated in map () if a better bat is found in map task specific input split. At the end of each map task this mapper level best bat is stored in distributed file systems. In this we take zero reduce task so mapper output will not be sorted or shuffled and is directly sent for output. After every iteration, gbest is evaluated from all mapper level mbest bats as shown in figure 4.2. Thus MBA reduces the no of comparison than MRBA for finding the gbest. Therefore in our proposed algorithm we have used MBA.

4.3 MAPPER IN BAT ALGORITHM

Hadoop job tracker set up instances of map task equal to the no of population split and each map task calls the map () function for each bat in the population split.

Map Task Set Up:

This method is called once at the beginning of each map task in which we set mapper level best as last generation gbest.

Algorithm 1: setup (context)

```
mBest = gBest;
```

Here mBest is Mapper level best bat and gBest is Global best bat

Map Function:

In this thesis, we present two models of parallel bat algorithm. First model uses one map phase for one generation of bat algorithm i.e. population is evolving generation by generation while the second model uses one map phase for all the generations i.e. each bat is evolving for N generations. That means in Model 1, N Map Reduce cycle execute whereas in Model 2 only one map reduce cycle will generate the optimum output. So we have two map algorithms corresponding to these two models.

Model 1 map () function explores and exploits new solution for given key bat using mBest as x^*. And the outputs improved bat which is sent as input for next generation as shown in figure 4.3.

Algorithm 2 for Model 1: map (key, value, context)

1. bat ← BATREPRESENTATION(key)
2. fitness ← value
3. # Generate new solution by adjusting frequency and updating velocities and locations/solutions.
4. newBat ← NEWBAT (bat, mBest)
5. **if (**rand > r_i**)**
6. newBat ← LOCALBAT (mBest)
7. **end if**
8. newFitness ← CALCULATEFITNESS (newBat)
9. **if (**rand < A_i **&** newFitness < fitness**)**
10. # Accept the new solution.
11. bat = newBat
12. fitness = newFitness
13. **end if**
14. **if (**fitness < mBest.fitness**)**

15. mBest = bat
16. **end if**
17. EMIT (bat, fitness)

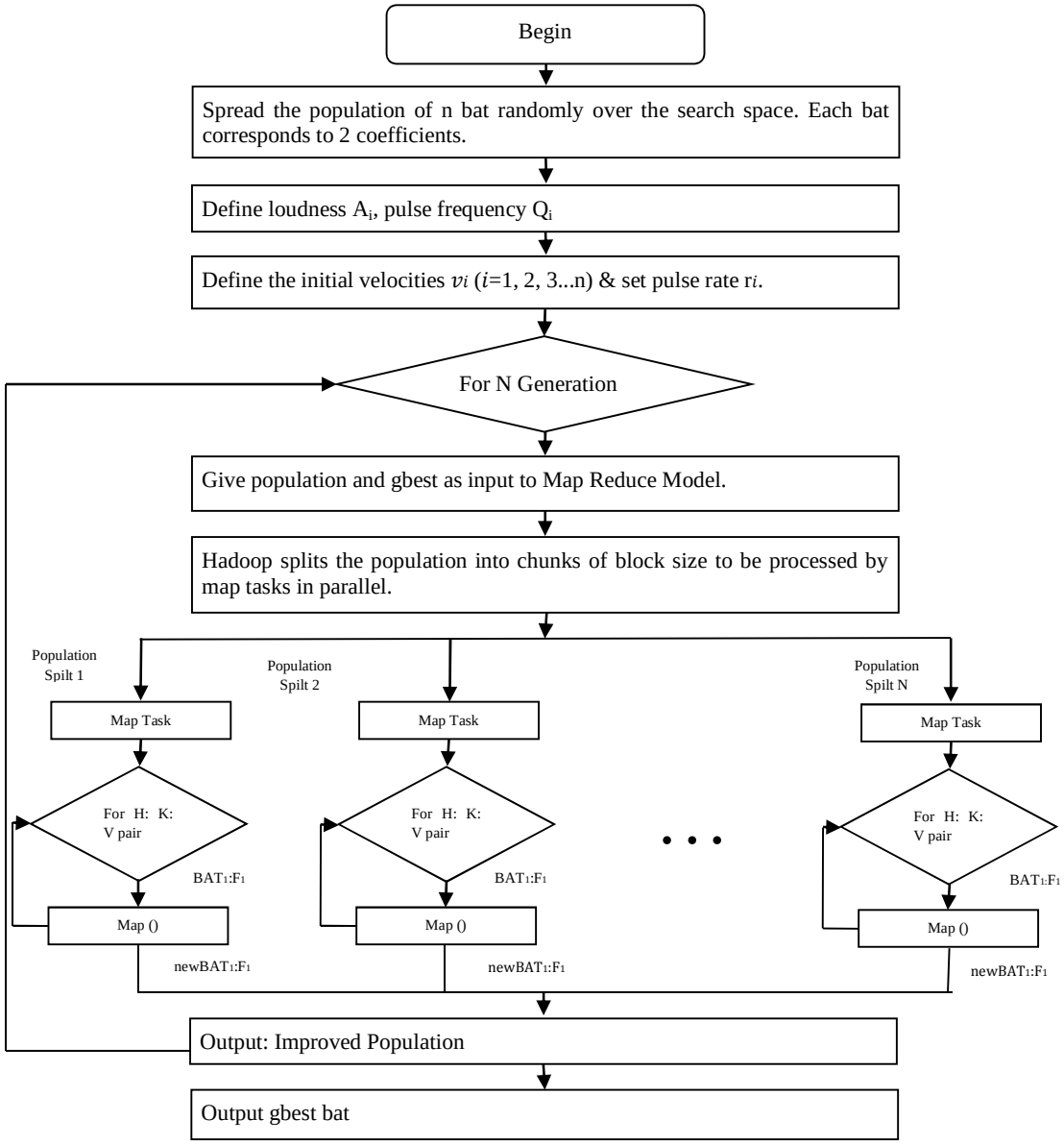

Figure 4.3: Flow chart of Model 1

Model 2 map () function evolves the given key-bat for N generations using mBest as x*.In this no intermediate data are emitted by any Mapper and each iteration evaluated gbest is considered as the output of the job as shown figure 4.4

Algorithm 3 for Model 2: map (key, value, context)

```
1.  bat ← BATREPRESENTATION(key)
2.  fitness ← value
3.  while t <Max number of Generations do
4.    newBat ← NEWBAT(bat, mBest)
5.    if (rand >ri)
6.      newBat ← LOCALBAT(mBest)
7.    end if
8.    newFitness ← CALCULATEFITNESS(newBat)
9.    if (rand <Ai & newFitness < fitness)
10.     # Accept the new solution.
11.     bat = newBat
12.     fitness = newFitness
13.   end if
14.   if (fitness < mBest.fitness)
15.     mBest = bat
16.   end if
17. end while
```

Map Task Clean Up:

This method is called ones at the end of each map task in which improved mbest is written to the distributed file system. These mbest's are then evaluated at the end of each iteration for gbest.

Algorithm 4: cleanup (context)

```
Write(mBest, mBest.fitness)
```

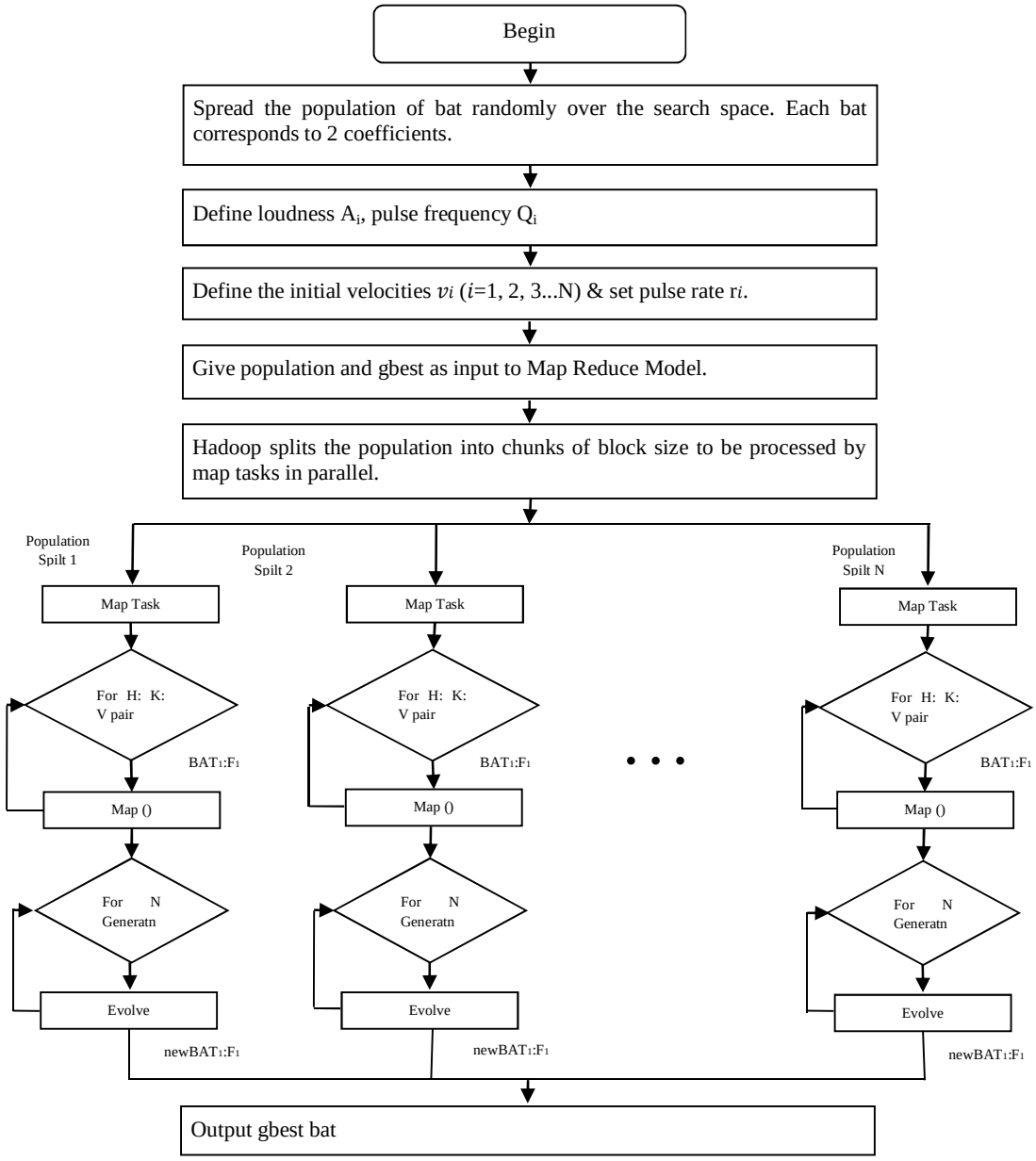

Figure 4.4: Flow chart of Model 2

CHAPTER 5
EXPERIMENTAL RESULT & ANALYSIS

5.1 IMPLEMENTATION

We have implemented these proposed models on optimization COCOMO II parameters (a, b) such that calculated efforts approximate to actual efforts for NASA 63 project Dataset.

Formally, this problem can be framed as finding parameter $X = \{x_1, x_2\}$ where $x_i \in \{0, 5\}$, that minimize the following equation:

$MMRE = [Actual - x_1(KLOC)^{x_2}] / Actual$

Here MMRE is Mean Magnitude of Relative Error which is used as evaluation criteria for assessment of optimized parameters. And since parameter x_1 and x_2 are specific to project mode therefore we execute program for each mode separately.

For this implementation, we have taken Bat Algorithm Specific Parameter as d = 2, r = 0.5, A = 0.5, $x_{max\ j}$=4, $x_{min\ j}$ = 0 and initial frequency/velocity as 0 and have considered constant pulse rate and loudness over the iterations.

5.2 ENVIRONMENT

We implemented these models on Apache Hadoop (0.19) and ran it on our 6 node cluster in which one node act as master and other acts as slaves. Each node runs a Intel 5 dual core, 4GB RAM and 250 GB hard disks. The nodes are integrated with Hadoop Distributed File System (HDFS) yielding a potential single image storage space of 2 * 250/3 = 166 GB (since the replication factor of HDFS is set to 3). By default each node can run 2 mappers and 1 reducers in parallel else it depends on hardware configuration.

5.3 EXPERIMENTAL RESULTS

We have performed following analysis on both the models:

1. **Scalability of MBA for effort estimation problem with increasing the number of mappers**:

In this experiment, we have taken the population size of 2 lakh which generated the input of 13 MB and have set the number of iterations as 5. Figure 5.1 compares the run time of MBA (Model 1) by using different number of map task i.e. setting different block size. On taking block size as 1 MB i.e. 13 map task, execution time is 16.84 which decreased with increase in block size due to less no of task distribution among the nodes. But on further decreasing the number of tasks (< no of nodes), run time became almost constant due to decrease in communication overhead.

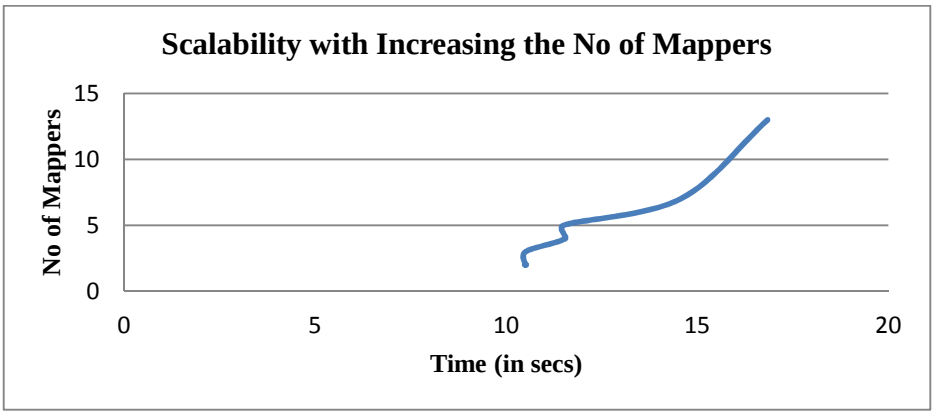

Figure 5.1

2. **Comparison between MRBA and MBA**:

In this experiment, we have taken population size as 2 lakh, number of iterations as 5, number of nodes as 6 and block size of 5MB and compared the running time of Map Reduce Bat Algorithm (MRBA) and Map Bat Algorithm (MBA) for Model 1. As shown in Figure 5.2, MBA takes less time as compare to MRBA by saving time in intermediate bat's sorting with no initialization of Reducer tasks. Moreover MRBA requires whole population sorting to get the gbest whereas MBA reads at most 13 mbest from the HDFS to find the gbest.

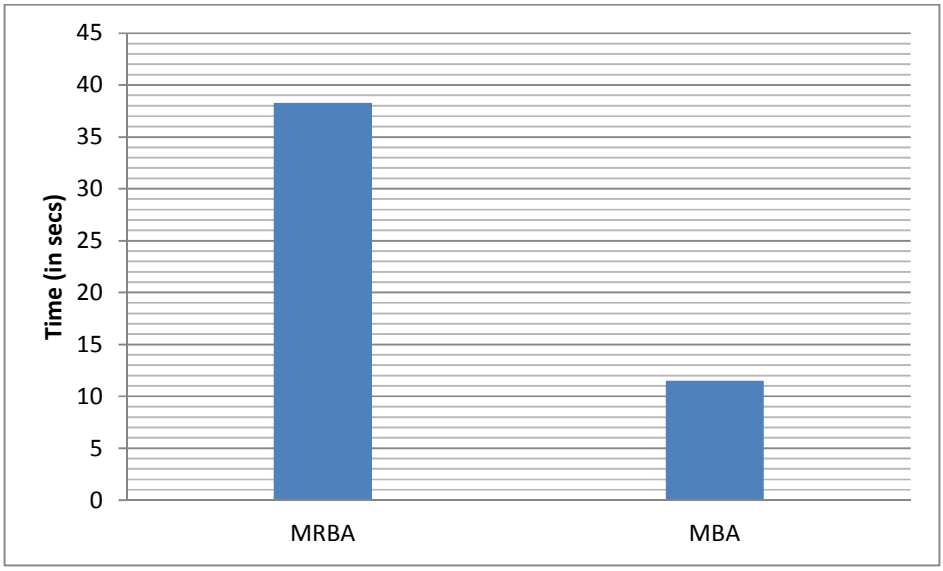

Figure 5.2

3. **Scalability of MBA for effort estimation problem with increasing the number of nodes**:

   In this experiment, we have taken population size as 2 lakh, number of iterations as 5, block size of 5MB and compared the run time of both MBA Models by increasing the number of nodes in the cluster. As shown in Figure 5.3, Model 2 took less time than Model 1 due to less no of map cycles and no intermediate data handling. For model 2, single node cluster took less time than 3 node cluster as there was no communication/task distribution overhead but on further increasing the number of nodes, the execution time reduced and became almost constant.

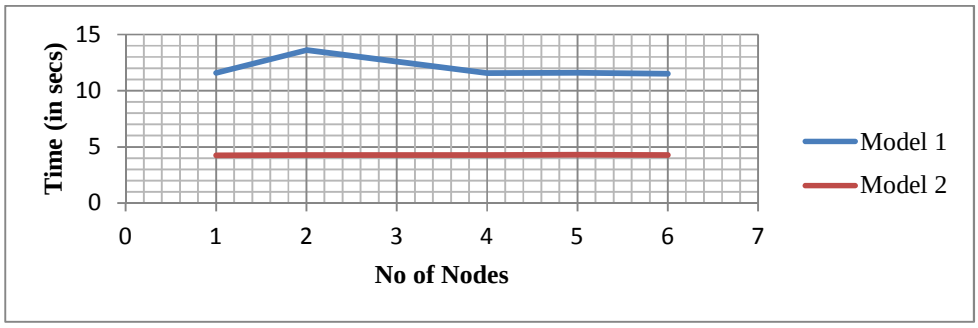

Figure 5.3

4. **Performance tuning with increase in population size**:

   In this experiment, we have taken number of iterations as 5 and block size of 5MB and compared the MMRE of embedded projects for both models by increasing the size of

population. As shown in Figure 5.4 MMRE reduced with increase in population and became constant after 1500.

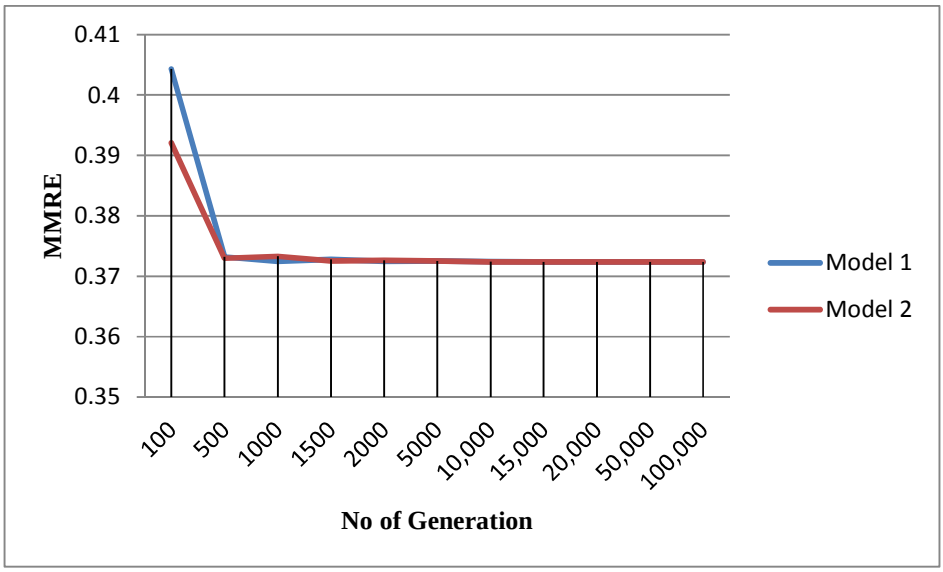

Figure 5.4

5. **Performance tuning with increase in number of Generations** :

In this experiment, we have taken population size as 10,000 and block size of 5MB and compared the MMRE of embedded projects for both models by increasing the number of iterations. As shown in Figure 5.5 both models give better MMRE than COCOMO Model (0.3921) and MMRE decreases with increase in generation due to more refining of output.

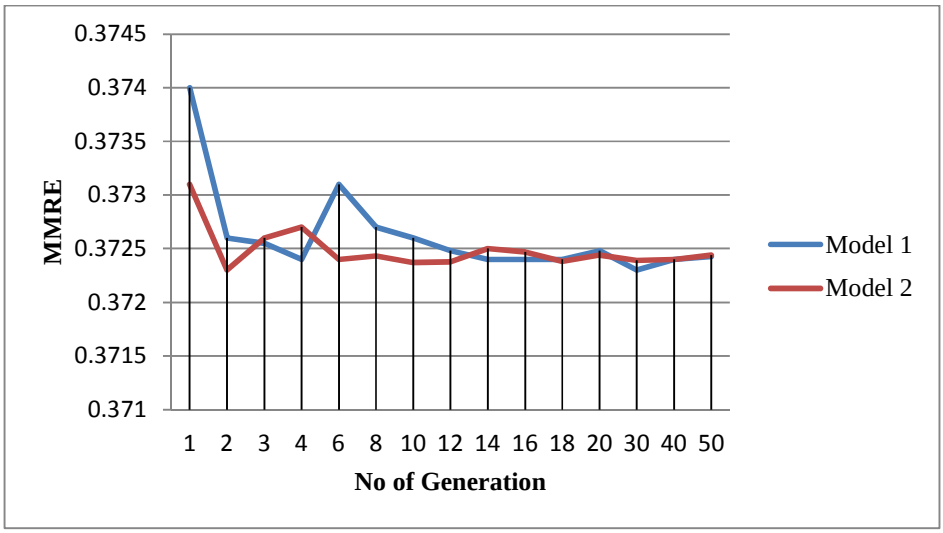

Figure 5.5

CHAPTER 6

CONCLUSION AND FUTURE WORK

The models proposed in this thesis can easily parallelize BA which could be used to solve problems involving large search space by simply adding more hardware resources to the cluster and without changing the proposed model code. And according to experimental results model 2 shows better convergence than model 1 and also take less time for execution.

It has been seen that lots of communication, task start up overhead is associated with Hadoop Map Reduce Architecture thus is not suitable for problems involving small search space, few dimension and less computation.

Due the update process in BA Parallel BA Models these proposed models can be used for large population but can't be used for given large dataset in order to find the optimal result from them. So further study on BA modification is required to find best results from given dataset for e.g. getting best quotation from large dataset of quotations.

In future work both models should be used for solving problems involving large search space, big computation, large no of dimensions like in stock market strategies. And these models running time can also be further improved by examining other features of MapReduce architecture like partitioner, combiner, shuffler etc. which may reduce the processing. We can also compare these models with existing MPI-based implementation.

References

[1] X.-S. Yang, "A New Metaheuristic Bat-Inspired Algorithm," *Studies in Computational Intelligence, Springer Berlin*, pp. 65-74, 2010.

[2] F. Wang, P. L. H. Yu, and D. W. Cheung, "Combining Technical Trading Rules Using Parallel Particle Swarm Optimization based on Hadoop," presented at the International Joint Conference on Neural Networks (IJCNN), Beijing, China, 2014.

[3] D.-W. Huang and J. Lin, "Scaling Populations of a Genetic Algorithm for Job Shop Scheduling Problems using MapReduce," presented at the 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), Indianapolis, IN, 2010.

[4] W. Zhao, H. Ma, and Q. He, "Parallel k-means clustering based on mapreduce," vol. 5931, pp. 674-679, 2009.

[5] K. Khan, A. Nikov, and A. Sahai, "A Fuzzy Bat Clustering Method for Ergonomic Screening of Office Workplaces," in *Third International Conference on Software, Services and Semantic Technologies S3T 2011*. vol. 101, ed: Springer Berlin Heidelberg, 2011, pp. 59-66.

[6] Y. Xin-She "Bat algorithm for multi-objective optimisation," *Internal Journal of Bio-Inspired Computation,* vol. 3, pp. 267-274, 2011.

[7] G. Komarasamy and A. Wahi, "An optimized K-means clustering technique using bat algorithm," *European Journal Scientific Research,* vol. 84, pp. 263-273, 2012.

[8] J.-H. Lin, C.-W. Chou, C.-H. Yang, and H.-L. Tsai, "A chaotic Levy flight bat algorithm for parameter estimation in nonlinear dynamic biological systems," *Journal of Computer and Information Technology,* vol. 2, 2012.

[9] R. Y. M. Nakamura, L. A. M. Pereira, K. A. Costa, D. Rodrigues, J. P. Papa, and X. S. Yang, "BBA: A binary bat algorithm for feature selection," presented at the 25th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI),, 2012.

[10] G. L. I. U., H. Huang, S. Wang, and Z. Chen, "An Improved Bat Algorithm with Doppler Effect for Stochastic Optimization," *International Journal of Digital Content Technology and its Applications,* vol. 6, pp. 326-336, 2012.

[11] J. Xie, Y. Q. Zhou, and H. Chen, "A novel bat algorithm based on differential operator and L´evy flights trajectory," *Computational Intelligence and Neuroscience,* vol. 2013, p. 13, 2013.

[12] J. W. Zhang and G. G. Wang, "Image Matching Using a Bat Algorithm with Mutation," in *Applied Mechanics and Materials*, Z. Du and B. Liu, Eds., ed, 2012, pp. 88-93.

[13] I. F. Jr., D. s. Fister, and X.-S. Yang, "A Hybrid Bat Algorithm," 2013.

[14] X. Wang, W. Wang, and Y. Wang, "An Adaptive Bat Algorithm," in *Intelligent Computing Theories and Technology,* ed: Springer Berlin Heidelberg, 2013.

[15] S. Yilmaz and E. U. Kucuksille, "Improved Bat Algorithm (IBA) on Continuous Optimization Problems," *Lecture Notes on Software Engineering,* pp. 279-283, 2013.

[16] A. Kaveh and P. Zakian, "Enhanced bat algorithm for optimal design of skeletal structures," *Asian Journal of civil engineering,* vol. 15, pp. 179-212, 2014.

[17] S. Yılmaz, E. U. Kucuksille, and Y. Cengiz, "Modified Bat Algorithm," *Electronics and Electrical Engineering,* vol. 20, 2014.

[18] T.-K. Dao, J.-S. Pan, T.-T. Nguyen, S.-C. Chu, and C.-S. Shieh, "Compact Bat Algorithm," in *Intelligent Data analysis and its Applications*, ed: Springer International Publishing, 2014.
[19] C.-F. Tsai, T.-K. Dao, W.-J. Yang, T.-T. Nguyen, and T.-S. Pan, "Parallelized Bat Algorithm with a Communication Strategy," in *Modern Advances in Applied Intelligence*, ed: Springer International Publishing, 2014.
[20] T.-K. Dao, T.-S. Pan, T.-T. Nguyen, and S.-C. Chu, "Evolved Bat Algorithm for Solving the Economic Load Dispatch Problem," in *Genetic and Evolutionary Computing*, ed: Springer International Publishing, 2015.
[21] G. Wang and L. Guo, "A novel hybrid bat algorithm with harmony search for global numerical optimization," *Journal of Applied Mathematics*, vol. 2013, pp. 1–21, 2013.
[22] N. M. Nawi, M. Z. Rehman, and A. Khan, "A New Bat Based Back-Propagation (BAT-BP) Algorithm," in *Advances in Systems Science*, ed, 2014.
[23] T. C. Bora, L. S. Coelho, and L. Lebensztajn, "Bat-inspired optimization approach for the brushless DC wheel motor problem," presented at the IEEE Transactions on Magnetics, 2012.
[24] R. Damodaram and M. L. Valarmathi, "Phishing website detection and optimization using modified bat algorithm," *nternational Journal of Engineering Research and Applications*, vol. 2, pp. 870–876, Jan-Feb 2012 2012.
[25] B. Ramesh, V. C. J. Mohan, and V. C. V. Reddy, "Application of bat algorithm for combined economic load and emission dispatch," *International Journal of Electricl Engineering and Telecommunications*, vol. 2, pp. 1–9, 2013.
[26] M. K. Marichelvam and T. Prabaharam, "A bat algorithm for realistic hybrid flowshop schedulihng problems to minimize makespan and mean flow time," *ICTACT Journal on Soft Computing*, vol. 3, pp. 428–433, October 2012 2012.
[27] A.-R. E. M., A. A. R., and S. Akhtar, "A metaheurisic bat- inspired algorithm for full body human pose estimation," presented at the 2012 Ninth Conference on Computer and Robot Vision, Toronto, ON, 2012.
[28] X. S. Yang, M. Karamanoglu, and S. Fong, "Bat aglorithm for topology optimization in microelectronic applications," presented at the IEEE International Conference on Future Generation Communication Technology (FGCT2012) London, 2012.
[29] A. L. Tamiru and F. M. Hashim, *Application of bat algorithm and fuzzy systems to model exergy changes in a gas turbine*. Springer, Heidelberg: Springer Berlin Heidelberg, 2013.
[30] A. Faritha Banu and C. Chandrasekar, "An optimized appraoch of modified bat algorithm to record deduplication," *International Journal of Computer Applications*, vol. 62, pp. 10–15, 2013.
[31] P. R. Srivastava, A. Bidwai, A. Khan, K. Rathore, R. Sharma, and X. S. Yang, "An empirical study of test effort estimation based on bat algorithm," *International Journal of Bio-Inspired Computation*, vol. 6, pp. 57-70, 2014.
[32] Y. Saji, M. E. Riffi, and B. Ahiod, "Discrete bat-inspired algorithm for travelling salesman problem " presented at the 2014 Second World Conference Complex Systems (WCCS), Agadir, 2014.
[33] N. Gupta and K. Sharma, "Optimizing intermediate COCOMO model using BAT algorithm," presented at the 2015 2nd International Conference on “Computing for Sustainable Global Development” (INDIACom), New Delhi, India 2015.

[34] H. Djelloul, S. Sabba, and S. Chikhi, "Binary bat algorithm for graph coloring problem," presented at the 2nd World Conference on Complex Systems (WCCS14), Agadir, 2014.
[35] E. A. Hassan, A. I. Hafez, A. E. Hassanien, and A. A. Fahmy, "A Discrete Bat Algorithm for the Community Detection Problem," in *Lecture Notes in Computer Science*, ed: Springer International Publishing, 2015.
[36] J. Dean and S. Ghemawat, "Mapreduce simplified data processing on large clusters," *Sixth Symposium on Operating System Design and Implementation,* vol. 51, pp. 107-113, 2004.
[37] H. Chih, A. Dasdan, R. L. Hsiao, and D. S. Parker, "Map-reducemerge: Simplified relational data processing on large clusters," in *Proceeding ACM SIGMOD International Conference on Management of data (SIGMOD 2007)*, 2007, pp. 1029-1040.
[38] C. Chu, S. Kim, Y. Lin, Y. Yu, G. Bradski, A. Ng, *et al.*, "Map-reduce for machine learning on multicore," in *Advances in Neural Information Processing Systems*, B. Scholkopf, J. Platt, and T. Hoffman, Eds., ed: MIT Press, 2007, pp. 281-288.
[39] J. Ekanayake, S. Pallickara, and G. Fox., "Mapreduce for data intensive scientific analyses," in *eScience '08. IEEE Fourth International Conference on eScience, 2008,* Indianapolis, IN, 2008, pp. 277–284.
[40] A. W. McNabb, C. K. Monson, and K. D. Seppi, "Parallel PSO Using MapReduce," presented at the IEEE Congress on Evolutionary Computation, 2007. CEC 2007, Singapore, 2007.
[41] C. Jin, C. Vecchiola, and R. Buyya, "MRPGA An Extension of MapReduce for Parallelizing Genetic Algorithms," presented at the IEEE Fourth International Conference on eScience, 2008., Indianapolis, IN 2008.
[42] A. Verma, X. Llorà, D. E. Goldberg, and R. H. Campbell, "Scaling Genetic Algorithms Using MapReduce," presented at the ISDA '09. Ninth International Conference on Intelligent Systems Design and Applications, 2009, Pisa 2009.
[43] D. Keco and A. Subasi, "Parallelization of genetic algorithms using Hadoop Map/Reduce," *SouthEast Europe Journal of Soft Computing*, vol. 1, 2012.
[44] L. Di Geronimo, F. Ferrucci, A. Murolo, and F. Sarro, "A Parallel Genetic Algorithm Based on Hadoop MapReduce for the Automatic Generation of JUnit Test Suites " presented at the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST), Montreal, QC, 2012.
[45] C.-Y. Lin, Y.-M. Pai, K.-H. Tsai, C. H.-P. Wen, and L.-C. Wang, "Parallelizing Modified Cuckoo Search on MapReduce Architecture," *Journal of Electronic Science and Technology,* vol. 11, 2013.