CHAPTER 1

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 proposed meta-heuristic algorithm are particle swam optimization PSO, genetic algorithm, cuckoo search etc. These algorithms are problem independent thus suits many optimization problem. Artificial Bee Colony (ABC) Algorithm is one of the swarm intelligence based nature inspired meta heuristic algorithm proposed by Karaboga in 2005 (KARABOGA). ABC has been successfully applied on many continuous optimization problems of minimum routing cost spanning tree, solving dynamic optimization problem, graph coloring problem etc. and it has shown better convergence to optimal solution than other meta heuristic algorithms. But ABC 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 would be to parallelize ABC algorithm and scale it to large number of processors.

But after program successfully parallelize, it must still focus on various issues of how to parallelize the computation, distribute the data and handle failures. 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 reduces primitives present in Lisp and many other functional languages. Hadoop map reduce 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), job shop scheduling problems, k-mean clustering etc. This thesis makes the following research contributions:

1. Demonstrates a transformation of artificial bee colony algorithms into the map and reduce primitives

2. Proposes a model of parallel artificial bee colony algorithm.

3. Implements the model on effort estimation problem and demonstrate its scalability to large problem sizes.

From this proposed work, many complex optimization problem based on ABC can easily scaled only by increasing the number of hardware resources.

1.1 Motivation

Big Data has become one of the buzzwords in IT industry during the last couple of years. Initially it was shaped by organizations which had to deal with speedy growth rates of data like web data, data resulting from scientific or business simulations or other data sources. The Google File System and MapReduce Architecture were resulted from the pressure to handle the expanding data amount on the web by Google (Ghemawat, Dean, & Sanjay, 2008). These technologies were tried to rebuild as open source software. This lead to Apache Hadoop and the Hadoop File System and laid the foundation for technologies under the umbrella of 'big data'.

Evolutionary Computing (EC) is a field of Computer science that adapts the theory of Darwinian evolution to optimize Computing problems (Eiben & Smith, 2003)**.** The basis of EC revolves around the process of natural evolution such as competition, random variation, and reproduction. The process of EC is globally applicable for wide range of problems due to the fact that evolution itself is an optimization process. Two of its most popular algorithms, Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) have been used ubiquitously to optimize problems in numerous fields of study.

Despite the robust nature of EC, expensive computing and storing resources are required due to its parallel nature. Thus when EC gets parallelly executed it performs better as compared to that of its normal serial execution. Therefore, EC has since become a popular subject of parallelization using various parallel processing techniques and architectures.

Optimization is the process of search for the maximum or minimum of a given objective function. Artificial Bee Colony (ABC) is a simple and effective evolutionary algorithm, but it may take hours or days to optimize difficult objective functions which are deceptive or expensive. Deceptive functions may be highly multimodal and multidimensional, and ABC requires extensive exploration to avoid being trapped in local optima. Expensive functions whose computational complexity may arise from dependence on detailed simulations or large datasets, takes a long time to evaluate. For deceptive or expensive objective functions, ABC must be parallelized to use multiprocessor systems and clusters efficiently.

ABC can be expressed naturally in Google's MapReduce framework (Ghemawat, Dean, & Sanjay, 2008) to develop a robust and simple parallel implementation that includes communication, load balancing, and fault tolerance. This flexible implementation makes it easy to apply modifications to the algorithm, such as those that improve optimization of difficult objective functions and improve parallel performance.

1.2 Objective

The objective of the thesis is to propose and realize an approach to implement Artificial Bee Colony Algorithm in a parallel manner and use it to compute the effort of COCOMO Model. The population of the bee here are in the form of big data are stored in Hadoop Distributed File System (HDFS) on Hadoop to have better fault tolerance and to store large amount of data in distributed manner.

The machine learning technique used in our model is Artificial Bee Colony. ABC is a fairly recent addition to the family of non-gradient based, probabilistic search approach that is based on a simple social model and is closely related to swarming theory. Although ABC algorithm presents several attractive properties to the designer, they are plagued by high computational cost as measured by elapsed time.

Thus, this evolutionary algorithm is implemented in MapReduce architecture on Hadoop to overcome its limitations of large scale serial implementation for computationally intensive functions. MapReduce ABC is simple, flexible, scalable and robust because it is designed in the MapReduce parallel programming model.

The benefits that can be obtained using this model are remarkable: the unique property of Hadoop of replication factor provides us at different level of abstraction with the availability of documentation that is beneficial for anyone who wants to access the records. Hadoop also improves response time and is fault tolerant due to which storing database on it is very advantageous.

Thus, this model overall proves to be beneficial for both data owner as well as end-users. Using this model an optimal value of effort required could be computed for a given project within a small span of time which could save both computation time and resources when there are reasonable numbers of projects or complex computations.

1.3 Thesis Outline

The thesis is divided into 6 chapters:

Chapter 1 is the introduction part. It describes the motivation of the work, objective of the thesis and also the structure of the thesis.

Chapter 2 is the literature survey. It includes the description of the work and contribution of various people in the same field and their findings.

Chapter 3 is basically the research background. This chapter discusses optimization, overview of artificial bee colony and its application. Also a brief overview of big data, its characteristics, challenges faced by big data and its benefits. Workflow of MapReduce architecture is also discussed in this chapter.

Chapter 4 presents the detailed explanation of Hadoop and its components used in the work and their working.

Chapter 5 presents the implementation or proposed work and the results obtained after running the code on our setup.

Conclusion and future work are presented in **chapter 6**

CHAPTER 2

2. LITERATURE REVIEW

1. Graph coloring is a challenging problem of coloring the nodes of any graph by using the least possible number of colors and on the same time we have to ensure that no two adjacent nodes gain same color. In this paper, Ranjeet singh tomar, Sonali Singh, shekhar Verma, Geetam Singh Tomar (Tomar, Singh, Verma, & Tomar, A Novel ABC Optimization Algorithm for Graph Coloring Problem, 2013) have proposed a novel Artificial Bee Colony (ABC) optimization algorithm for graph coloring problem. They analyzed the proposed algorithm and compared the algorithm with three other graph coloring algorithms.

   Problem of finding chromatic number of any graph is an NP-complete problem and hence is unsolvable by any existing polynomial time algorithm reliably In many problems we have to deal with the conflicts arising. Graph coloring model proved itself that it works well as a conflict resolver. Graph coloring problem has been even so often used for solving the problems in important applications like school time table scheduling, register allocating in computer, problem of bandwidth allocation, and lots of other applications. The results indicate that ABC algorithm converges in few iterations and is able to optimally allocate colors to vertices of graph.

2. In the paper, "An Artificial Bee Colony Algorithm for Solving Dynamic Optimization Problems" authors, (Kojima, Nakano, & Miyauchi, 2013) proposed an ABC algorithm for solving dynamic optimization problems with the procedure which is very simple. For various dynamic optimization problems, the proposed method provides fast solution search. And this method is also very effective in the numerical simulations.

3. In the paper Alok Singh and Shyam Sundar (Singh & Sundar, 2011) have proposed an Artificial Bee Colony Algorithm (ABC)-based approach for the minimum routing cost spanning tree (MRCST) problem. They used four best methods- two genetic algorithm, a

stochastic hill climber and a perturbation-based local search and compared all these methods against their approach. And ABC proves its superiority over other approaches by obtaining better quality solution.

MRCST problem can be described as finding a spanning tree with minimum routing cost among all spanning trees of the graph. It is a NP-Hard problem (Johnson et al. 1978). The ABC approach is a combination of an ABC algorithm and a local search heuristic. The best solution we obtain through the ABC algorithm is improved further by using the local search. The MRCST problem has many important application areas such as in communication network design, in multiple sequence alignment problems in computational biology (Wu and Chao 2004).

4. In the paper, "A Novel Parallel Approach of Cuckoo Search using MapReduce ", authors (Xu, Ji, Yuan, & Liu, 2014) invent a novel approach to accommodate cuckoo search to the analysis of big data by using MapReduce paradigm which is a latest cloud computing technique. As the backend MapReduce engine, they use HADOOP platform during the implementation of their method. Processing large dataset their approach has much better runtime performance.

5. In the paper," Parallelization of the Artificial Bee Colony (ABC) Algorithm", author (SUBOTIC, TUBA, & STANAREVIC)proposed some modifications to the Artificial Bee Colony algorithm that introduced parallelization of the standard ABC algorithm. Since multiple processors can be better utilized parallelization allow faster execution. This approach has been used in many optimization problems.

6. In the paper," A MapReduce Based Ant Colony Optimization Approach to Combinatorial optimization Problems", author (Wu, Wu, & Yang, 2012)proposed a MapReduce based Ant Colony Optimization approach and they also showed how ACO algorithm can be modeled into MapReduce framework. They also described the implementation and algorithm design of ACO on Hadoop.

7. In the paper," Parallel K-PSO Based on MapReduce", (Wang, Yuan, & Jiang, 2012)parallel K-PSO which is an improved method of Particle Swarm Optimization

(PSO) with K-means based on MapReduce is proposed. To improve the global search ability of K-means the approach takes an advantage of PSO and then to enhance its capability of processing massive data it takes K-means parallel with MapReduce.

8. In the paper," A MapReduce-based approach for shortest path problem in large-scale networks", authors (Aridhi, Lacomme, Ren, & Vincent, 2015)proposed a MapReduce-based approach for the shortest path problem in large-scale real-road networks. Such problem is a cornerstone of any real-world routing problem. In this proposed approach the original graph is partitioned into a set of sub-graphs, then solution for original graph is obtained by solving the shortest path on each sub-graph in a parallel way.

9. The work proposed by Dino Kečo, Abdulhamit Subasi (Kečo & Subasi, 2012), presents parallel implementation of Genetic Algorithm. This parallel implementation of Genetic Algorithm is compared with its serial implementation in solving One Max (Bit Counting) problem. The comparison criteria used in this work are fitness convergence, quality of final solution, algorithm scalability and cloud resource utilization. The proposed model shows better performance and fitness convergence than serial model, but has lower quality of solution because of species problem.

10. In the work proposed by Andrew W. McNabb, Christopher K. Monson, and Kevin D. Seppi (McNabb, Monson, & D., 2012), they have stated that a problem using web content, commercial transaction information or bioinformatics data may involve large amounts of data and requires minutes and hours for each function evaluation. And PSO must be parallelized to optimize such functions. They have proposed MapReduce Particle Swarm Optimization (MRPSO) that is intended for computationally intensive functions and have used problem of training a Radial Basis Function (RBF) Network as representative of optimization problems which uses large amount of data. Once a program successfully scales, it must still address the issue of failing nodes. Google too faced the same problem in large scale parallelization. It is demonstrated that MRPSO scales to 256 processors on moderately difficult problems and tolerates node failures.

CHAPTER 3

3. RESEARCH BACKGROUND

3.1 Artificial Bee Colony Algorithm

The Artificial Bee colony algorithm is a population-based newly invented metaheuristic approach which is motivated by the intelligent foraging behavior of honey bee swarm. This algorithm is introduced by Karaboga (2005) (KARABOGA) and further developed by Karaboga and Basturk (Basturk and Karaboga 2006; Karaboga and Basturk 2007a, b, 2008, 2009) and Singh (2009).

3.1.1 Overview

In ABC, there are three classes of the foraging bees—employed, onlookers and scouts.

- **Employed bees** - Employed bees are those bees that are presently tapping a rich artificial food source. The employed bees are responsible for bringing loads of nectar from the food sources to the hive and sharing the information about food sources with onlooker bees.
- **Onlookers bees** – Onlookers are those bees that are waiting in the hive for the information to be shared by the employed bees about their food sources.
- **Scouts bees** - scouts are those bees that are presently looking for new food sources in the vicinity of the hive.

Figure 3.1: Honey Bee

Employed bees share information about food sources by dancing in a common area in the hive called dance area. The nature and duration of a dance depends on the nectar content of the food source currently being tapped by the dancing bee. Onlooker bees observe numerous dances before choosing a food source. The onlookers have a tendency to choose a food source with a probability proportional to the nectar content of that food source. Therefore, good food sources attract more bees than the bad ones. Whenever a bee, whether it is scout or onlooker, finds a food source it becomes employed. Whenever a food source is completely exhausted, all the employed bees associated with it leave it, and can again become scouts or onlookers. So in a way scout bees can be perceived as performing the job of exploration, whereas employed and onlooker bees.

Figure 3.2: Food Sources

3.1.2 The ABC Algorithm

The ABC algorithm is an SI algorithm that mimics the behavior of bees. As compared with GA and PSO, the ABC is more effective for high-dimensional and multi-modal problems with complex nonlinearity.

In ABC, food sources (search points) have position vectors in search space and their evaluation values for an objective function. The food sources are updated by three groups of bees, called Employed Bees, Onlooker Bees and Scout Bees. The following roles and search methods are assigned to them.

1) ***Initialization Phase-*** All vector of the population of food source are initialized and control parameter are set.
2) ***Employed Bee Phase-*** The extensive global search phase in which each Employed Bee searches each assigned food source.
3) ***Onlooker Bee Phase-*** The intensive local search phase in which each Onlooker Bee searches solutions around food sources with high fitness values.
4) ***Scout Bee Phase-*** The stochastic search phase in which each Scout Bee discards unimproved food sources and relocates them randomly in search space.

The general scheme of ABC algorithm is as follows:

```
Initialization Phase
REPEAT
  Employed Bees Phase
  Onlooker Bees Phase
  Scout Bees Phase
  Memorize the best solution achieved so far
UNTIL(Cycle=Maximum Cycle Number or a Maximum CPU time)
```

In ABC algorithm, the colony of artificial bees consists of three groups of bees: employed bees, onlookers and scouts. First half of the colony consists of the employed artificial bees and the second half includes the onlookers. For Artificial Bee Colony (ABC) Optimization Algorithm 791 every food source, there is only one employed bee. In other words, the number of employed bees is equal to the number of food sources around the hive. The employed bee whose the food source has been abandoned by the bees becomes a scout.

In ABC algorithm, the position of a food source represents a possible solution to the optimization problem and the nectar amount of a food source corresponds to the quality (fitness) of the associated solution. The number of the employed bees or the onlooker bees is equal to the number of solutions in the population.

At the first step, the ABC generates a randomly distributed initial population $P(G = 0)$ of SN solutions (food source positions), where SN denotes the size of population. Each solution xi ($i = 1, 2, ..., SN$) is a D-dimensional vector. Here, D is the number of optimization parameters. After initialization, the population of the positions (solutions) is subjected to repeated cycles, $C = 1, 2, ...,MCN$, of the search processes of the employed bees, the onlooker bees and scout bees. An employed bee produces a modification on the position (solution) in her memory depending on the local information (visual information) and tests the nectar amount (fitness value) of the new source (new solution). Provided that the nectar amount of the new one is higher than that of the previous one, the bee memorizes the new position and forgets the old one. Otherwise she keeps the position of the previous one in her memory.

After all employed bees complete the search process, they share the nectar information of the food sources and their position information with the onlooker bees on the dance area. An onlooker bee evaluates the nectar information taken from all employed bees and chooses a food source with a probability related to its nectar amount. As in the case of the employed bee, she produces a modification on the position in her memory and checks the nectar amount of the candidate source. Providing that its nectar is higher than that of the previous one, the bee memorizes the new position and forgets the old one.
An artificial onlooker bee chooses a food source depending on the probability value associated with that food source, p_i , calculated by the following expression (1):

$$p_i = \frac{fit_i}{\sum_{n=1}^{SN} fit_n} \qquad (1)$$

where fit_i is the fitness value of the solution i which is proportional to the nectar amount of the food source in the position i and SN is the number of food sources which is equal to the number of employed bees (BN).
In order to produce a candidate food position from the old one in memory, the ABC uses the following expression (2):

$$v_{ij} = x_{ij} + \phi_{ij}\left(x_{ij} - x_{kj}\right) \qquad (2)$$

where $k \in \{1, 2,..., SN\}$ and $j \in \{1, 2,...,D\}$ are randomly chosen indexes. Although k is determined randomly, it has to be different from i. $\phi_{i,j}$ is a random number between [-1, 1]. It controls the production of neighbours food sources around $x_{i,j}$ and represents the comparison of two food positions visually by a bee. As can be seen from (2), as the difference between the parameters of the $x_{i,j}$ and $x_{k,j}$ decreases, the perturbation on the position $x_{i,j}$ gets decrease, too.

Thus, as the search approaches to the optimum solution in the search space, the step length is adaptively reduced. If a parameter value produced by this operation exceeds its predetermined limit, the parameter can be set to an acceptable value. In this work, the value of the parameter exceeding its limit is set to its limit value. The food source of which the nectar is abandoned by the bees is replaced with a new food source by the scouts. In ABC, this is simulated by producing a position randomly and replacing it with the abandoned one. In ABC, providing that a position cannot be improved further through a predetermined number of cycles, then that food source is assumed to be abandoned. The value of predetermined number of cycles is an important control parameter of the ABC algorithm, which is called "*limit*" for abandonment. Assume that the abandoned source is xi and $j \in \{1, 2,...,D\}$, then the scout discovers a new food source to
be replaced with xi. This operation can be defined as in (3)

$$x_i^j = x_{\min}^j + rand(0,1)\left(x_{\max}^i - x_{\min}^j\right) \tag{3}$$

After each candidate source position vi,j is produced and then evaluated by the artificial bee, its performance is compared with that of its old one. If the new food has an equal or better nectar than the old source, it is replaced with the old one in the memory. Otherwise, the old one is retained in the memory. In other words, a greedy selection mechanism is employed as the selection operation between the old and the candidate one. It is clear from the above explanation that there are four control parameters used in the ABC: The number of food sources which is equal to the number of employed or onlooker bees (*SN*), the value of *limit*, the maximum cycle number (*MCN*).

3.1.3 Pseudo Code

Detailed pseudo-code of the ABC algorithm is given below:

1: Initialize the population of solutions $x_{i,j}$, $i = 1...SN$, $j = 1...D$

2: Evaluate the population

3: cycle=1

4: **repeat**

5: Produce new solutions $v_{i,j}$ for the employed bees by using (2) and evaluate them

6: Apply the greedy selection process

7: Calculate the probability values $P_{i,j}$ for the solutions $x_{i,j}$ by (1)

8: Produce the new solutions $v_{i,j}$ for the onlookers from the solutions xi,j selected depending on $P_{i,j}$ and evaluate them

9: Apply the greedy selection process

10: Determine the abandoned solution for the scout, if exists, and replace it with a new randomly produced solution $x_{i,j}$ by (3)

11: Memorize the best solution achieved so far

12: cycle=cycle+1

13: **until**

14: cycle=MCN

3.1.4 Applications

Swarm intelligence algorithms are playing very important roles in many disciplines today. In this section, we have discussed about different areas where artificial bee colony (ABC) has implemented.

The applications of the Artificial Bee Colony Algorithm can be summarized as:

I. General Assignment Problem

In this paper the performance of ABC was investigated, with the integration shift neighborhood structures and greedy randomized adaptive heuristic search. ABC was extended through integration of the employed and onlooker phases with shift neighborhood structures applied sequentially. The result shows the purposed algorithm can solve small to medium size generalized assignment problems effectively.

II. Cluster Analysis

Dervis Karaboga and Celal Ozturk used ABC in forming clusters of the benchmark classification problems for classification purpose. Abdulsalam et al. were presented a cluster deviation detection task using the ABC algorithm. ABC was tested on three UCI benchmark datasets. The outcome is that ABC performed well. Xin Sui et al. have presented the Cooperative Article Bee Colony CABC, which significantly improved the original ABC in solving clustering problems and some benchmark function.

III. Constrained Optimization Problem

In this paper ABC algorithm was extended to handle constrained optimization problems. A set of 13 benchmark optimization problems was examined, and the results were compared with Differential Evolution (DE) and PSO algorithms. Optimization functions, with the dimension of each function varying from 10 to 30. Now, ABC is give efficient result than ACO (ant colony optimization) and PSO (particle swarm optimization) as it never suffered from uncertain convergence time and partial optimization.

IV. Structural Optimization

This paper presented optimization of different skeletal structural using artificial bee colony algorithm. The performance was compared with other optimization algorithm from the literature. The results showed that ABC can efficiently be used in structural design problems.

V. Software Testing

In this paper a novel search technique, based on ABC, for automatic generation of structural software test has purposed. Fitness of individual are measured with the help of branch distance based objective function to generate test cases. Experiments are performed on ten real world problems. This technique has performed satisfactorily for most of the program, but not for programs with large input domains and many equality based path constraints.

VI. Numerical Assignment Problem

In this paper an Interactive Artificial Bee Colony (IABC) optimization is purposed for numerical optimization problems. The (IABC) introduces the concept of universal gravitation into the consideration of the affection between employed bees and the onlooker bees. Five benchmark functions are simulated in the experiments in order to compare the accuracy/quality of the IABC. The results got from the simulations indicate that the IABC performs better than the original ABC.

VII. MR Brain Image Classification

In this paper author implemented Fuzzy C- Means algorithm and hybrid ABC algorithm to detect the tumor and its size using real Brain MR images. ABC algorithm is used to optimize fuzzy clustering process. The result shows that ABC algorithm improves the efficiency of FCM segment process.

VIII. Face Poses Estimation

In this paper a novel pose estimation method was described, using a six-point template as the correspondence between the object space and the image space. Besides, we proposed a novel CABC based on the traditional ABC algorithm and Rossler attractor to solve the six-point template. The experiments showed that this proposed CABC algorithm is superior to both GA and PSO with respect to the errors of three rotational angles in both no noise condition and noisy conditions.

IX. Wireless Sensor

In this paper an improved ABC algorithm has been proposed to match the different characteristics of wireless sensor network deployment process, which will be optimum for real time dynamic network functioning. Artificial Bee Colony algorithm has opened up a vast stage for WSN protocol suite design. Like any swarm intelligent scheme, they are compatible to any stage of a WSN design and implementation, which makes them an attractive choice as the base.

3.2 Big Data

Today is the era of social media. Establishing new connections, social networking, online shopping, web postings, online lectures, blogging and much more. 'Daily data' as comments on Facebook, likes, video and pictures posts, tweets, millions of videos on YouTube are just common examples of the sources of millions and trillions of data that is being stored and uploaded/downloaded every day over the internet. The exponential growth of data is challenging for Facebook, Yahoo, Google, Amazon and Microsoft. The term 'Big Data' is used to refer the collection of data sets that are so large and complex to handle and process using traditional data processing applications. (Verma, Patel, & Patel, 2015)

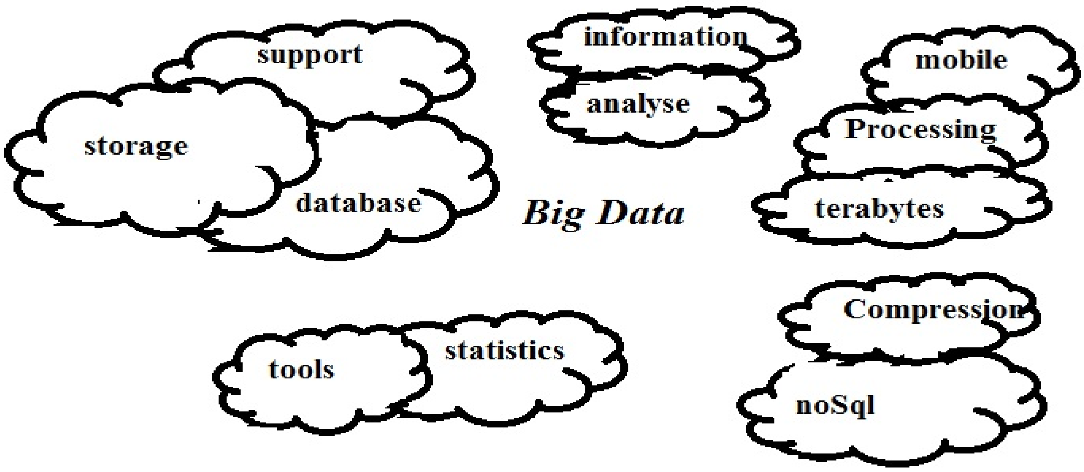


Figure 3.3: Big Data Characteristics

The term itself is being more formally defined by IBM as the combination of 3 V's is velocity, variety and volume. These are the generic big data properties. However, the acquired properties depicted after entering the system includes value, veracity, variability and visualization. Thus, the 7 V's correctly describes the big data. (Armour, Kaisler, & Espinosa, 2015)

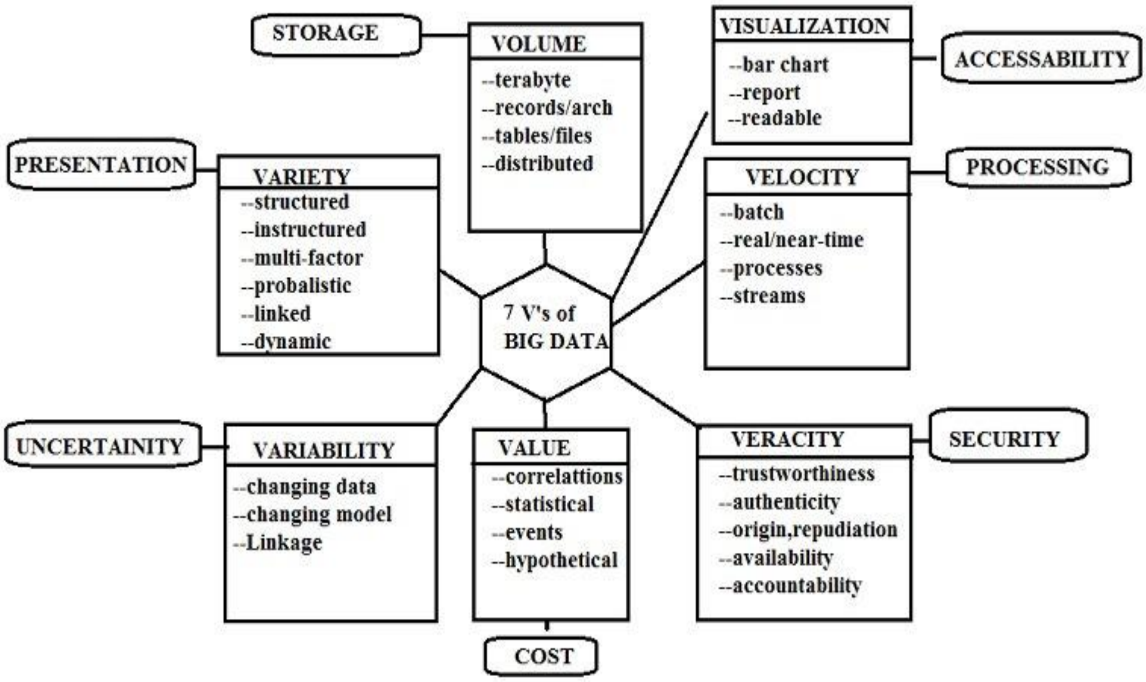


Figure 3.4: 7 V's of Big Data

Computational intelligence (CI) provides exceptional tools for addressing big data challenges. These techniques include evolutionary computation, neural computation and fuzzy systems which are inherently capable of handling uncertainty.

A. Volume

Millions of data is uploaded everyday on Facebook, twitter and other online platforms. Akamai analysis 75 million events a day that primarily targets online ads, Wal-Mart handles 1 million customer transactions per hour contributing to the exponential growth of data online as a part of big data. The system is generating terabytes, petabytes and zeta bytes of data. This data may be also handled using computationally intelligent biologically inspired techniques say bacteria colony optimization. This huge chunk of data is handled through CI technique of Data Mining expanding its scope to cover big data analytics.

B. Velocity

The system generates streams of data and multiple sources that require that data. There is an exponential growth in data every hour. For instance Walmart's data warehouse stored 1,000 terabytes of data in 1999 which surpassed over 2.5 petabytes in 2012. Every minute the data is flooded with thousands of online uploads. The widely accepted machine learning databases have increased to millions requiring features selection as a vital requirement. Various CI techniques are used for time domain astronomy (TDA).

C. Variety

Both structures and unstructured data which include blogs, images, audio, and videos are a part of big data. These data may be analyzed for sentiment and content. Earlier may be the days when companies dealt with only a single data format but today big data provides a platform for all data formats. Various CI techniques even biologically inspired swarm intelligent techniques can be used for the dealing with versatile data. Various data mining techniques are used for performing analysis by using neural networks, fuzzy logic and graphs and trees.

D. Variability

Big data allows handling uncertainty in data with changing data helping in prediction of future behavior of various customers, entrepreneurs, etc. Basically the meaning of data is constantly changing and the data relies mainly on language processing.

E. Veracity

In order to ensure the accuracy of big data various security tools are provided for ensuring potential value of the data. This involves automated decision making or feeding data into an unsupervised machine learning algorithm. This ensures the authenticity, availability and accountability of the data.

F. Visualization

The CI techniques involved in making the data readable and easily accessible contribute to the 5th V of the Big data. The data needs to be easily understood and the CI techniques such as the various optimization algorithms provide an advantage of providing an optimal review of the data analyzed.

G. Value

The value of big data is huge. It enables sentiment analysis, prediction and recommendation. It is massive and rapidly expanding, but it loses its worth when dealt without analysis and visualization that encounters noisy, messy and rapidly changing data. This value of the big data may be extracted only when various CI techniques are applied to big data enabling easy analysis and maximum profit.

The Big data can be analyzed by using through swarm intelligent approach like bacteria colony optimization**.** The bacteria colony gives a huge problem space and hence giving a big data problem space domain for performing analysis and optimization for speedy decision making activities.

3.2.1 Definition

Since 2011 interest in an area known as big data has increased exponentially. The term big data has become ubiquitous. Owing to a shared origin between academia, industry and the media there is no single unified definition, and various stakeholders provide diverse and often contradictory definitions.

The McKinsey Global Institute has defined this term as – *"Big Data refers to data sets whose size is beyond the ability of typical database software tools to capture, store, manage and analyze."* The process of research into massive amounts of data to reveal hidden patterns and secret correlations named as big data analytics (SINANC, SAGIROGLU, & Duygu, 2013).

3.2.2 What Comes Under Big Data?

Big data involves the data produced by different devices and applications. Given below are some of the fields that come under the umbrella of Big Data:

a. **Black Box Data:** It is a component of helicopter, airplanes, and jets, etc. It captures voices of the flight crew, recordings of microphones and earphones, and the performance information of the aircraft.

b. **Social Media Data:** Social media such as Facebook and Twitter hold information and the views posted by millions of people across the globe.

c. **Stock Exchange Data:** The stock exchange data holds information about the 'buy' and 'sell' decisions made on a share of different companies made by the customers.

d. **Power Grid Data:** The power grid data holds information consumed by a particular node with respect to a base station.

e. **Transport Data:** Transport data includes model, capacity, distance and availability of a vehicle.

f. **Search Engine Data:** Search engines retrieve lots of data from different databases.

3.2.3 Big Data and Hadoop

Traditionally it may be feasible to analyze the data limited data stored over the server with was stored over the file systems. The data intensive companies (Google, Yahoo, Amazon, and Microsoft) required figuring out the on-demand books, websites, and popular people and thus deciding what kind ads actually appealed the audience. The existing tools and SQL based query analysis tools are not sufficient enough for meeting the growing data analysis demands failing at tackling multiplatform, storage of data requiring multiplatform codes. (Gupta, 2015) Hadoop is a distributive open source framework for writing and running distributed applications that process large amounts of data. The fey features offered by Hadoop are:-

• ***Accessibility-*** Hadoop runs on large clusters of commodity machines and provides easy access to all the systems overcoming the barriers of distance.

• ***Robust-*** Hadoop can easily overcome the frequent machine malfunctions since it runs on commodity hardware.

• ***Scalable-*** Hadoop scales linearly to handle larger data by adding more nodes to the cluster.

• ***Simple-*** The simplicity of Hadoop lies in writing quick efficient parallel programs supporting giving the programmer the advantage of using programs in any language (Java, Python).

• ***Cost effective-*** Hadoop proves to be cost effective in using commodity hardware and not expensive servers. The working environment in Hadoop is given by its Hadoop ecosystem. Hadoop provides various analysis tools, data warehousing, data querying and data mining tools inclusive of machine learning algorithms such that Hadoop may be used for the analysis of big data.

3.2.4 Benefits of Big Data Analytics

Google, eBay and LinkedIn were among the first to experiment with big data. They developed proof of concept and small-scale projects to learn if their analytical models could be improved with new data sources. In many cases, the results of these experiments were positive.

Today, big data analytics is no longer just an experimental tool. Many companies have begun to achieve real results with the approach, and are expanding their efforts to encompass more data and models. Three major benefits of big data analytics are:

1. Cost reduction

Big data technologies like Hadoop and cloud-based analytics can provide substantial cost advantages. While comparisons between big data technology and traditional architectures (data warehouses and marts in particular) are difficult because of differences in functionality, a price comparison alone can suggest order-of-magnitude improvements.

Virtually every large company, however, is employing big data technologies not to replace existing architectures, but to augment them. Rather than processing and storing vast quantities of new data in a data warehouse, for example, companies are using Hadoop clusters for that purpose, and moving data to enterprise warehouses as needed for production analytical applications.

Well-established firms like Citi, Wells Fargo and USAA all have substantial Hadoop projects underway that exist alongside existing storage and processing capabilities for analytics. While the long-term role of these technologies in enterprise architecture is unclear, it's likely that they will play a permanent and important role in helping companies manage big data.

2. Faster, better decision making

Analytics has always involved attempts to improve decision making, and big data doesn't change that. Large organizations are seeking both faster and better decisions with big data, and they're finding them. Driven by the speed of Hadoop and in-memory analytics, several companies focus on speeding up existing decisions.

For example, Caesars, a leading gaming company that has long embraced analytics, is now embracing big data analytics for faster decisions. The company has data about its customers from its Total Rewards loyalty program, web click streams, and real-time play in slot machines. It has traditionally used all those data sources to understand customers, but it has been difficult to integrate and act on them in real time, while the customer is still playing at a slot machine or in the resort.

Caesars has found that if a new customer to its loyalty program has a run of bad luck at the slots; it's likely that customer will never come back. But if it can present, say, a free meal coupon to that customer while he's still at the slot machine, he is much more likely to return to the casino later. The key, however, is to do the necessary analysis in real time and present the offer before the customer turns away in disgust with his luck and the machines at which he's been playing.

In pursuit of this objective, Caesars has acquired Hadoop clusters and commercial analytics software. It has also added some data scientists to its analytics group.

Some firms are more focused on making better decisions analyzing new sources of data. For example, health insurance giant United Healthcare is using "natural language processing" tools from SAS to better understand customer satisfaction and when to intervene to improve it. It starts by converting records of customer voice calls to its call center into text and searching for indications that the customer is dissatisfied. The company has already found that the text analysis improves its predictive capability for customer attrition models.

3. New products and services

Perhaps the most interesting use of big data analytics is to create new products and services for customers. Online companies have done this for a decade or so, but now predominantly offline firms are doing it too. GE, for example, has made a major investment in new service models for its industrial products using big data analytics.

Verizon Wireless is also pursuing new offerings based on its extensive mobile device data. In a business unit called Precision Market Insights, Verizon is selling information about how often mobile phone users are in certain locations, their activities and backgrounds. Customers thus far have included malls, stadium owners and billboard firms.

3.3 Map Reduce Architecture

MapReduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster. Conceptually similar approaches have been very well known since 1995 with the Message Passing Interface standard having reduce and scatter operations.

A MapReduce program is composed of a **Map( )** procedure that performs filtering and sorting (such as sorting students by first name into queues, one queue for each name) and a **Reduce( )** procedure that performs a summary operation (such as counting the number of students in each queue, yielding name frequencies). The "MapReduce System" (also called "infrastructure" or "framework") orchestrates the processing by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance.

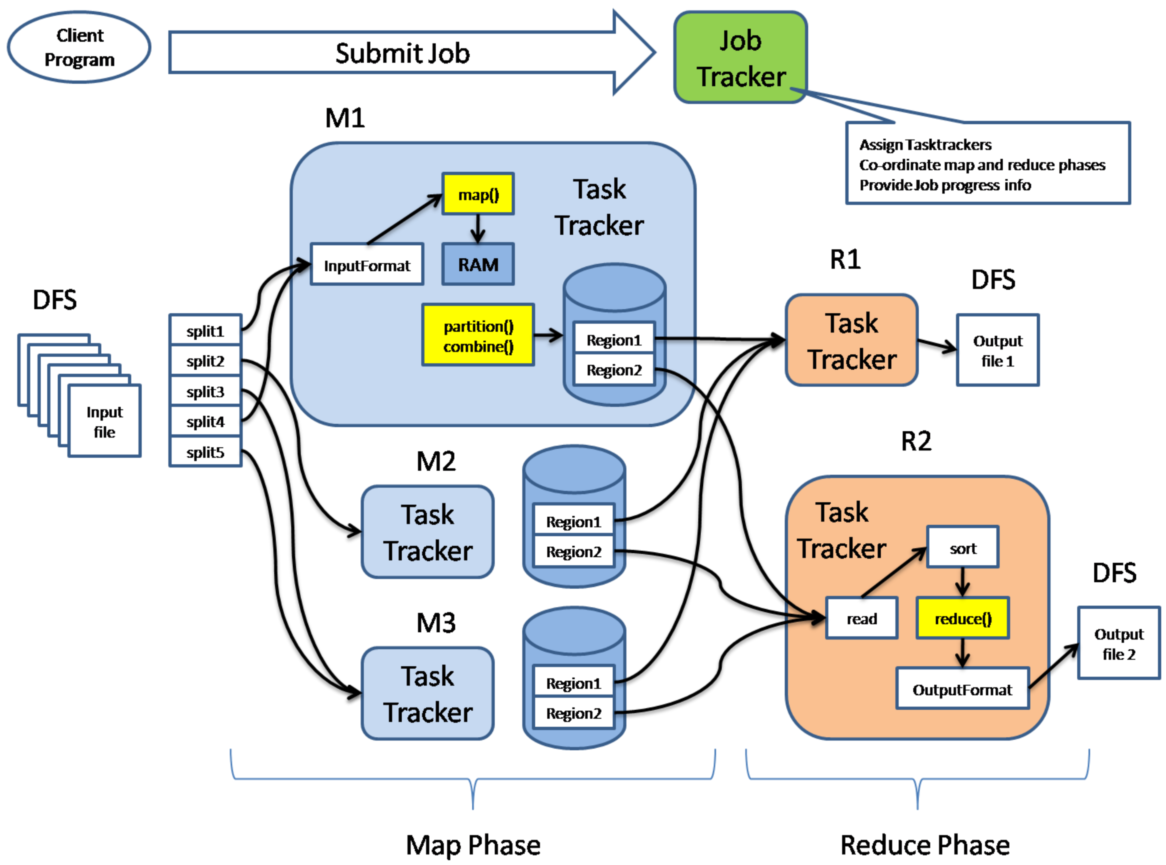


Figure 3.5: Map Reduce Architecture

The model is inspired by the map and reduce functions commonly used in functional programming, although their purpose in the MapReduce framework is not the same as in their original forms. The key contributions of the MapReduce framework are not the actual map and reduce functions, but the scalability and fault-tolerance achieved for a variety of applications by optimizing the execution engine once. As such, a single-threaded implementation of MapReduce (such as MongoDB) will usually not be faster than a traditional (non-MapReduce) implementation; any gains are usually only seen with multi-threaded implementations. The use of this model is beneficial only when the optimized distributed shuffle operation (which reduces network communication cost) and fault tolerance features of the MapReduce framework come into play. Optimizing the communication cost is essential to a good MapReduce algorithm. (Aridhi, Lacomme, Ren, & Vincent, 2015)

MapReduce libraries have been written in many programming languages, with different levels of optimization. A popular open-source implementation that has support for distributed shuffles is part of Apache Hadoop. The name MapReduce originally referred to the proprietary Google technology, but has since been genericized. MapReduce as a big data processing model is considered dead by many domain experts, as development has moved on to more capable and less disk-oriented mechanism that incorporate full map and reduce capabilities.

3.3.1 Inputs and Outputs

The MapReduce framework operates exclusively on **<key, value>** pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types.

The key and value classes have to be serializable by the framework and hence need to implement the Writable interface. Additionally, the key classes have to implement the WritableComparable interface to facilitate sorting by the framework.

Input and Output types of a MapReduce job:

Map takes one pair of data with a type in one data domain, and returns a list of pairs in a different domain:

Map (k1,v1) → list (k2,v2)

The ***Reduce*** function is then applied in parallel to each group, which in turn produces a collection of values in the same domain:

Reduce (k2, list (v2)) → list (v3)

3.3.2 Workflow

MapReduce is implemented in a master/worker configuration, with one master serving as the coordinator of many workers. A worker may be assigned a role of either a *map worker* or a *reduce worker*.

Step 1. Split input

The first step, and the key to massive parallelization in the next step, is to split the input into multiple pieces. Each piece is called a split, or shard. For *M* map workers, we want to have *M* shards, so that each worker will have something to work on. The number of workers is mostly a function of the amount of machines we have at our disposal.

The MapReduce library of the user program performs this split. The actual form of the split may be specific to the location and form of the data. MapReduce allows the use of custom readers to split a collection of inputs into shards, based on specific format of the files.

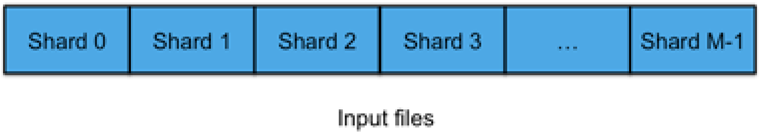


Figure 3.6: Split input into shards

Step 2. Fork processes

The next step is to create the master and the workers. The master is responsible for dispatching jobs to workers, keeping track of progress, and returning results. The master picks idle workers and assigns them either a map task or a reduce task. A map task works on a single shard of the original data. A reduce task works on intermediate data generated by the map tasks. In all, there will be *M* map tasks and *R* reduce tasks. The number of reduce tasks is the number of partitions defined by the user. A worker is sent a message by the master identifying the program (map or reduce) it has to load and the data it has to read.

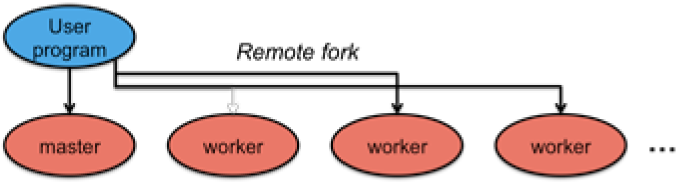


Figure 3.7: Remotely execute worker processes

Step 3. Map

Each *map* task reads from the input shard that is assigned to it. It parses the data and generates *(key, value)* pairs for data of interest. In parsing the input, the *map* function is likely to get rid of a lot of data that is of no interest. By having many map workers do this in parallel, we can linearly scale the performance of the task of extracting data.

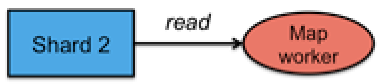


Figure 3.8: Map task

Step 4: Map worker: Partition

The stream of *(key, value)* pairs that each worker generates is buffered in memory and periodically stored on the local disk of the map worker. This data is partitioned into *R* regions by a partitioning function.

The partitioning function is responsible for deciding which of the *R* reduce workers will work on a specific key. The default partitioning function is simply a hash of *key* modulo *R* but a user can replace this with a custom partition function if there is a need to have certain keys processed by a specific reduce worker.

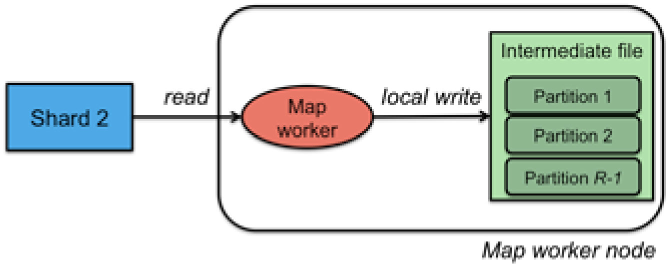


Figure 3.9: Create intermediate files

Step 5: Reduce: Sort (Shuffle)

When all the map workers have completed their work, the master notifies the reduce workers to start working. The first thing a reduce worker needs to is to get the data that it needs to present to the user's *reduce* function. The reduce worker contacts every map worker via remote procedure calls to get the *(key, value)* data that was targeted for its partition. This data is then sorted by the keys. Sorting is needed since it will usually be the case that there are many occurrences of the same key and many keys will map to the same reduce worker (same partition). After sorting, all occurrences of the same key are grouped together so that it is easy to grab all the data that is associated with a single key.

This phase is sometimes called the shuffle phase.

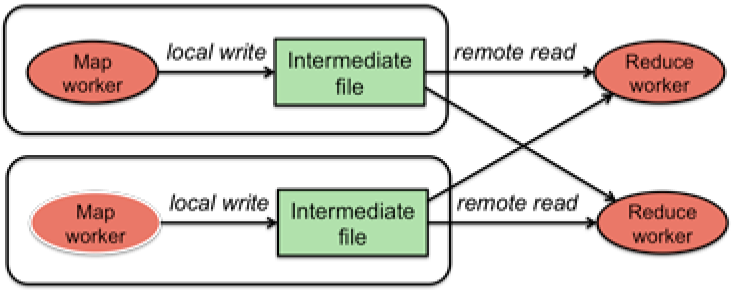


Figure 3.10: Sort and merge partitioned data

Step 6: Reduce function

With data sorted by keys, the user's *Reduce* function can now be called. The reduce worker calls the *Reduce* function once for each unique key. The function is passed two parameters: the key and the list of intermediate values that are associated with the key.

The *Reduce* function writes output sent to file.

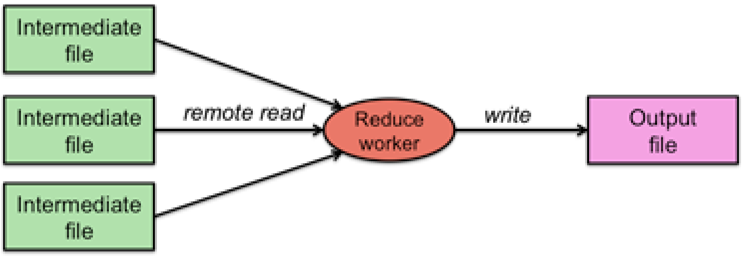


Figure 3.11: Reduce function writes output

Step 7: Done!

When all the reduce workers have completed execution, the master passes control back to the user program. Output of MapReduce is stored in the *R* output files that the *R* reduce workers created. (Marozzo, Talia, & Trunfio, 2011)

The big picture

Figure 7 illustrates the entire MapReduce process. The client library initializes the shards and creates map workers, reduce workers, and a master. Map workers are assigned a shard to process. If there are more shards than map workers, a map worker will be assigned another shard when it is done. Map workers invoke the user's *Map* function to parse the data and write intermediate *(key, value)* results onto their local disks. This intermediate data is partitioned into *R* partitions according to a partioning function. Each of *R* reduce workers contacts all of the map workers and gets the set of *(key, value)* intermediate data that was targeted to its partition. It then calls the user's *Reduce* function once for each unique key and gives it a list of all values that were generated for that key. The *Reduce* function writes its final output to a file that the user's program can access once MapReduce has completed.

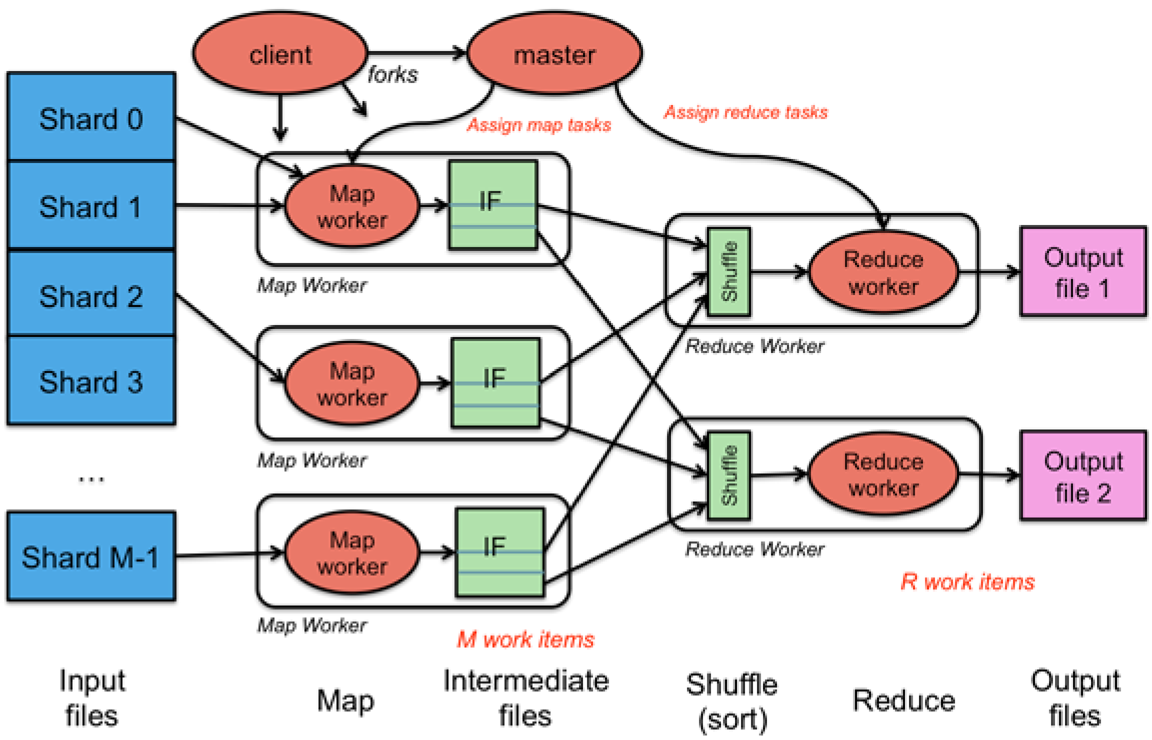


Figure 3.12: MapReduce

3.3.3 Uses

MapReduce is useful in a wide range of applications, including distributed pattern-based searching, distributed sorting, web link-graph reversal, Singular Value Decomposition, web access log stats, inverted index construction, document clustering, machine learning, and statistical machine translation. Moreover, the MapReduce model has been adapted to several computing environments like multi-core and many-core systems, desktop grids, volunteer computing environments, dynamic cloud environments, and mobile environments.

At Google, MapReduce was used to completely regenerate Google's index of the World Wide Web. It replaced the old *ad hoc* programs that updated the index and ran the various analyses. Development at Google has since moved on to technologies such as Percolator, Flume and MillWheel that offer streaming operation and updates instead of batch processing, to allow integrating "live" search results without rebuilding the complete index. MapReduce's stable inputs and outputs are usually stored in a distributed file system. The transient data is usually stored on local disk and fetched remotely by the reducers.

3.3.4 Benefits

The following table describes some of MapReduce's key benefits:

Table 1: Benefits of Map Reduce

Benefit	Description
Simplicity	Developers can write applications in their language of choice, such as Java, C++ or Python, and MapReduce jobs are easy to run
Scalability	MapReduce can process petabytes of data, stored in HDFS on one cluster
Speed	Parallel processing means that MapReduce can take problems that used to take days to solve and solve them in hours or minutes
Recovery	MapReduce takes care of failures. If a machine with one copy of the data is unavailable, another machine has a copy of the same key/value pair, which can be used to solve the same sub-task. The JobTracker keeps track of it all.
Minimal data motion	MapReduce moves compute processes to the data on HDFS and not the other way around. Processing tasks can occur on the physical node where the data resides. This significantly reduces the network I/O patterns and contributes to Hadoop's processing speed.

CHAPTER 4

4. APACHE HADOOP

Apache Hadoop is an open-source software framework written in Java for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures (of individual machines or racks of machines) are commonplace and thus should be automatically handled in software by the framework. (Yang & Li, 2013)

The core of Apache Hadoop consists of a storage part (Hadoop Distributed File System (HDFS)) and a processing part (MapReduce). Hadoop splits files into large blocks and distributes them amongst the nodes in the cluster. To process the data, Hadoop MapReduce transfers packaged code for nodes to process in parallel, based on the data each node needs to process. This approach takes advantage of data locality—nodes manipulating the data that they have on hand—to allow the data to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are connected via high-speed networking.

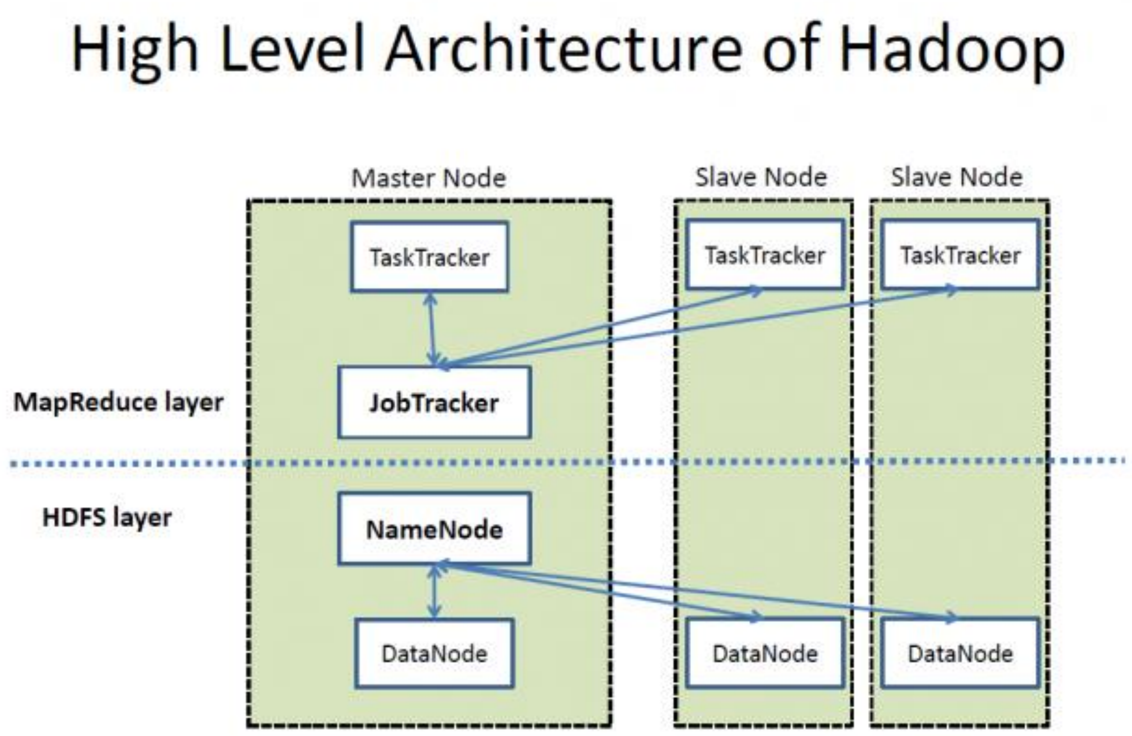


Figure 4.1: Architecture of Hadoop

The base Apache Hadoop framework is composed of the following modules:

- *Hadoop Common* – contains libraries and utilities needed by other Hadoop modules;
- *Hadoop Distributed File System (HDFS)* – a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster;
- *Hadoop YARN* – a resource-management platform responsible for managing computing resources in clusters and using them for scheduling of users' applications; and
- *Hadoop MapReduce* – a programming model for large scale data processing.

The term "Hadoop" has come to refer not just to the base modules above, but also to the "ecosystem", or collection of additional software packages that can be installed on top of or alongside Hadoop, such as Apache Pig, Apache Hive, Apache HBase, Apache Spark, and others.

Apache Hadoop's MapReduce and HDFS components were inspired by Google papers on their MapReduce and Google File System. Hadoop is supported by its own list of operating systems - Red Hat Enterprise, CentOS, Oracle Linux, Ubuntu, SUSE Linux Enterprise Server.

4.1 Characteristics of Hadoop

1. Scale-Out rather than Scale-Up means Hadoop requires more machines or nodes to be added to the existing distributed system which is easier instead of adding more RAM or CPU for scaling up which is more difficult.

2. It brings code to data, in data to code data is loaded to the processor from storage device located remotely and results are sent back to storage device as done traditionally whereas to bring code to data means both processor and storage are located on same machine and processors run code and access underlying database.

3. **Deal with failures** – they are common while working with large number of machines but Hadoop is designed to cope up with failures as data is replicated on various nodes and tasks are retired.

4. **Abstract complexity** of distributed and concurrent applications, allows developers to focus on application development and business logic and frees developer from worrying about processing Big Data on clusters of commodity hardware and system level challenges

5. Vibrant open-source community

6. Many tools and products reside on top of Hadoop

7. Hadoop consists of the Hadoop Common, which provides access to the file systems supported by Hadoop.

8. Hadoop has published APIs

4.2 Hadoop Cluster

Hadoop Cluster is a set of "cheap" commodity hardware networked together which resides in the same location i.e. set of servers resides in set of racks which are in data centre. “Cheap” Commodity Server Hardware means that there is no need for super-computers, and can use commodity unreliable hardware. The hardware used are not desktops but servers. Hadoop Cluster is a collection of Hadoop nodes where each node consists of a Processor and Storage as shown in figure 4.2. In Hadoop cluster, processors access underlying local storage and execute code.

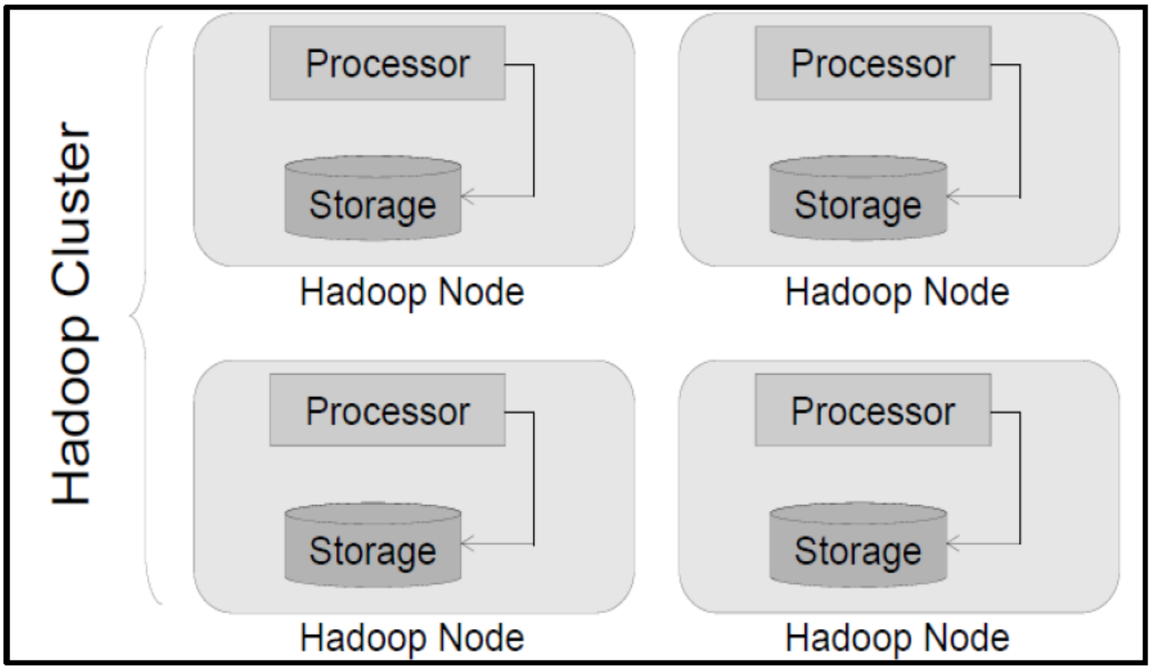


Figure 4.2: Hadoop Cluster

4.3 Hadoop Ecosystem

1. Hadoop Distributed File System

2. MapReduce: a distributed data processing framework

3. HBase: Hadoop column database; supports random reads and limited queries and batch

4. Zookeeper: Highly-Available Coordination Service

5. Oozie: Hadoop workflow scheduler and manager

6. Pig: Data processing language and execution environment

7. Hive: Data warehouse with SQL interface

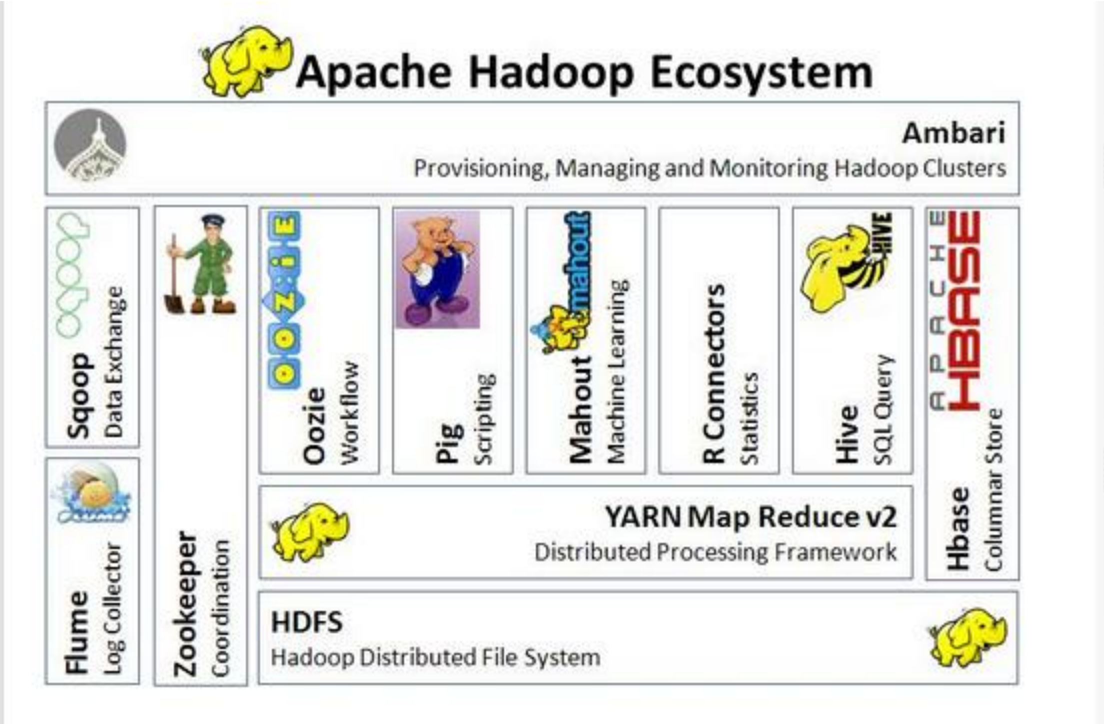


Figure 4.3: Apache Hadoop Ecosystem

4.4 How is Hadoop Different from Past Techniques?

- **Hadoop can handle data in a very fluid way.** Hadoop is more than just a faster, cheaper database and analytics tool. Unlike databases, Hadoop doesn't insist that you structure your data. Data may be unstructured and schemaless. Users can dump their data into the framework without needing to reformat it. By contrast, relational databases require that data be structured and schemas be defined before storing the data.

- **Hadoop has a simplified programming model**. Hadoop's simplified programming model allows users to quickly write and test distributed systems. Performing computation on large volumes of data has been done before, usually in a distributed setting but writing distributed systems is notoriously hard. By trading away some programming flexibility, Hadoop makes it much easier to write distributed programs.

- **Hadoop is easy to administer.** Alternative high performance computing (HPC) systems allow programs to run on large collections of computers, but they typically require rigid program configuration and generally require that data be stored on a separate storage area network (SAN) system. Schedulers on HPC clusters require careful administration and since program execution is sensitive to node failure, administration of a Hadoop cluster is much easier.

Hadoop invisibly handles job control issues such as node failure. If a node fails, Hadoop makes sure the computations are run on other nodes and that data stored on that node are recovered from other nodes.

- **Hadoop is agile.** Relational databases are good at storing and processing data sets with predefined and rigid data models. For unstructured data, relational databases lack the agility and scalability that is needed. Apache Hadoop makes it possible to cheaply process and analyze huge amounts of both structured and unstructured data together, and to process data without defining all structure ahead of time.

4.5 Comparison between Hadoop and Distributed Databases

Table 2: Comparison between Hadoop and Distributed Databases

S.No.	**HADOOP**	**Distributed Databases**
1.	Used with relational database for batch processing	Until recently used for batch processing in various applications
2.	Scale out using more machines	Scale up using CPUs and RAM
3.	Cheap commodity is used to scale out	Expensive to scale for larger installations
4.	Works best with unstructured or semi-structured data	Works well with structured data tables that conform to a specified schema
5.	For offline batch processing	For Online Transactions and low-latency queries

6.	Is designed to stream large amounts of data and large files	It works best with small records
7.	Supports JSON, XML, images, etc.	Does not have support for JSON, images, XML etc.

4.6 Benefits

While large Web 2.0 companies such as Google and Facebook use Hadoop to store and manage their huge data sets, Hadoop has also proven valuable for many other more traditional enterprises based on its five big advantages.

1. Scalable

Hadoop is a highly scalable storage platform, because it can store and distribute very large data sets across hundreds of inexpensive servers that operate in parallel. Unlike traditional relational database systems (RDBMS) that can't scale to process large amounts of data, Hadoop enables businesses to run applications on thousands of nodes involving thousands of terabytes of data.

2. Cost effective

Hadoop also offers a cost effective storage solution for businesses' exploding data sets. The problem with traditional relational database management systems is that it is extremely cost prohibitive to scale to such a degree in order to process such massive volumes of data. In an effort to reduce costs, many companies in the past would have had to down-sample data and classify it based on certain assumptions as to which data was the most valuable. The raw data would be deleted, as it would be too cost-prohibitive to keep. While this approach may have worked in the short term, this meant that when business priorities changed, the complete raw data set was not available, as it was too expensive to store. Hadoop, on the other hand, is designed as a scale-out architecture that can affordably store all of a company's data for later use. The cost savings are staggering: instead of costing thousands to tens of thousands of pounds per terabyte, Hadoop offers computing and storage capabilities for hundreds of pounds per terabyte.

3. Flexible

Hadoop enables businesses to easily access new data sources and tap into different types of data (both structured and unstructured) to generate value from that data. This means businesses can use Hadoop to derive valuable business insights from data sources such as social media, email conversations or clickstream data. In addition, Hadoop can be used for a wide variety of purposes, such as log processing, recommendation systems, data warehousing, market campaign analysis and fraud detection.

4. Fast

Hadoop's unique storage method is based on a distributed file system that basically 'maps' data wherever it is located on a cluster. The tools for data processing are often on the same servers where the data is located, resulting in much faster data processing. If you're dealing with large volumes of unstructured data, Hadoop is able to efficiently process terabytes of data in just minutes, and petabytes in hours.

5. Resilient to failure

A key advantage of using Hadoop is its fault tolerance. When data is sent to an individual node, that data is also replicated to other nodes in the cluster, which means that in the event of failure, there is another copy available for use.

The MapR distribution goes beyond that by eliminating the NameNode and replacing it with a distributed No NameNode architecture that provides true high availability. Our architecture provides protection from both single and multiple failures.

When it comes to handling large data sets in a safe and cost-effective manner, Hadoop has the advantage over relational database management systems, and its value for any size business will continue to increase as unstructured data continues to grow.

4.7 Hadoop Distributed File System (HDFS)

Hadoop Distributed File System is a file system that runs on top of native file system like Ext3, Ext4 and others, and is based on Google file system. It gives user appearance of a single disk. It is highly fault tolerant in a way that it can handle disk crashes, machine crashes, etc. It is built upon cheap commodity hardware which reduces the overall cost of installation of Hadoop

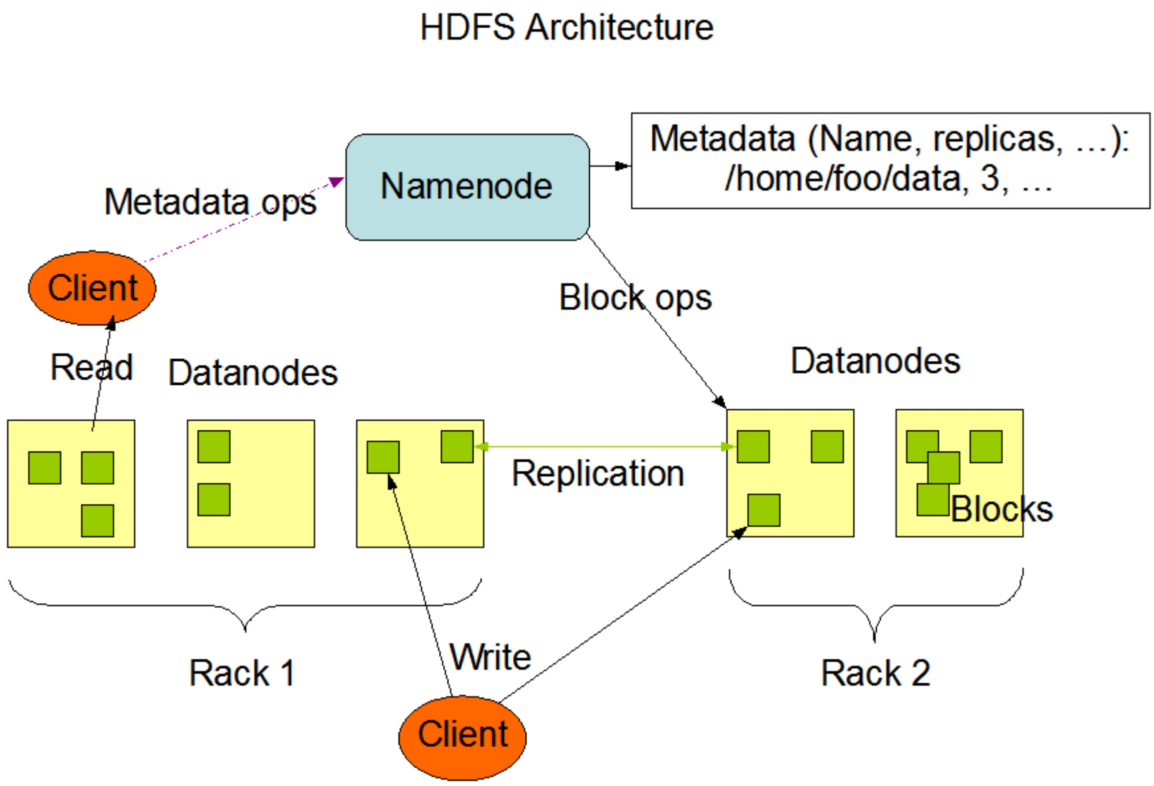


Figure 4.4: HDFS Architecture

4.7.1 HDFS Daemons

File system cluster is being managed by three types of processes namely, NameNode, DataNode and Secondary NameNode.

a. **NameNode:** It manages the file systems namespace, meta-data and file blocks. It runs on one machine and manages several machines. All DataNodes report to NameNode about their presence and according to the number of available DataNodes it manages degree of replication as decided by the Administrator. For fast access NameNode keeps all block meta-data in memory. The other role is to serve the client queries, it allows clients to add/copy/move/delete a file, it will records the actions into a transaction log. For the performance, it save the whole file structure tree in RAM and hard drive. A HDFS only allow one running NameNode, that's why it is a single point of failure, if the NameNode failed or goes down, the whole file system will goes offline too. So, for the NameNode machine, we need to take special cares on it, such as adding more RAM to it, this will increase the file system capacity, and do not make it as DataNode, JobTracker and other optional roles.

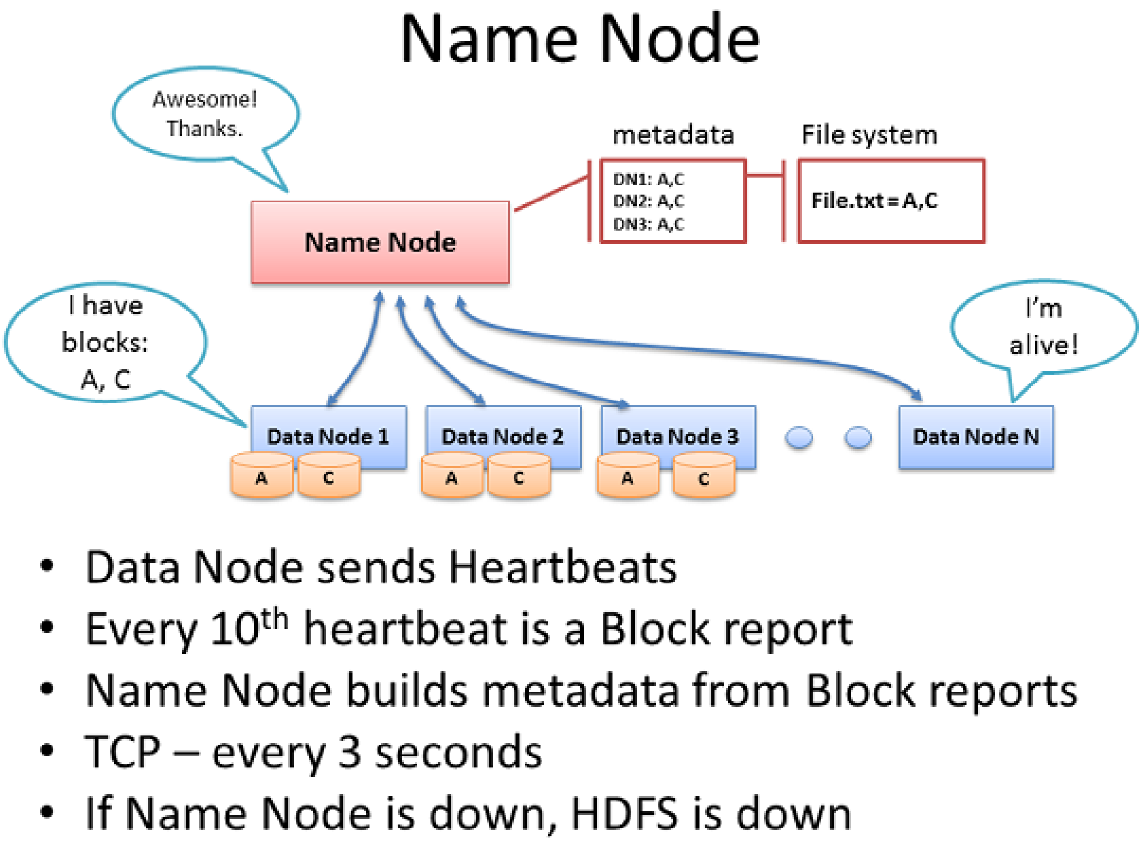


Figure 4.5: NameNode

b. **DataNode:** It stores and retrieves data blocks according to the request after it has reported to NameNode about its health. It runs on many machines and forms the cluster. On startup, DataNode will connect to the NameNode and get ready to respond to the operations from NameNode. After the NameNode telling the position of a file to the client, the client will directly talk to the DataNode to access the files. DataNodes could also talk to each other when they replicating data. The DataNode will also periodically send a report of all existing blocks to the NameNode and validates the data block checksums

c. **Secondary NameNode:** It performs the house keeping work so that NameNode doesn't have to do it and reduces the load of NameNode. It requires similar hardware as NameNode machine and is not used for high-availability – not a backup for NameNode. Its works is to back-up the metadata and store it to the hard disk, this may helping to reduce the restarting time of NameNode. In HDFS, the recent actions on HDFS will be stored in to a file called EditLog on the NameNode, after restarting HDFS; the NameNode will replay according to the Editlog. Secondary NameNode will periodically combines the content of EditLog into a checkpoint and clear the EditLog File, after that, the NameNode will replay start from the latest checkpoint, the restarting time of NameNode will be reduced.

4.7.2 HDFS File Read and Write

In the Hadoop Cluster, NameNode accepts the request but does not directly read or write data to HDFS which is one of the reasons for HDFS‟s scalability. Initially, client interacts with the NameNode to update the HDFS namespace of NameNode and client retrieves block locations for reading and writing then it directly interacts with Datanode to read/write data. The Read and Write operations on the file are explained below.

4.7.2.1 HDFS Write

The write operation in HDFS is done in seven steps as shown in figure 4.6.

1. Create new file in the NameNode's Namespace and calculate block topology
2. Stream data to the first DataNode
3. Stream data to the second DataNode in the pipeline
4. Stream data to the third DataNode
5. Success/Failure acknowledgement
6. Success/Failure acknowledgement
7. Success/Failure acknowledgement

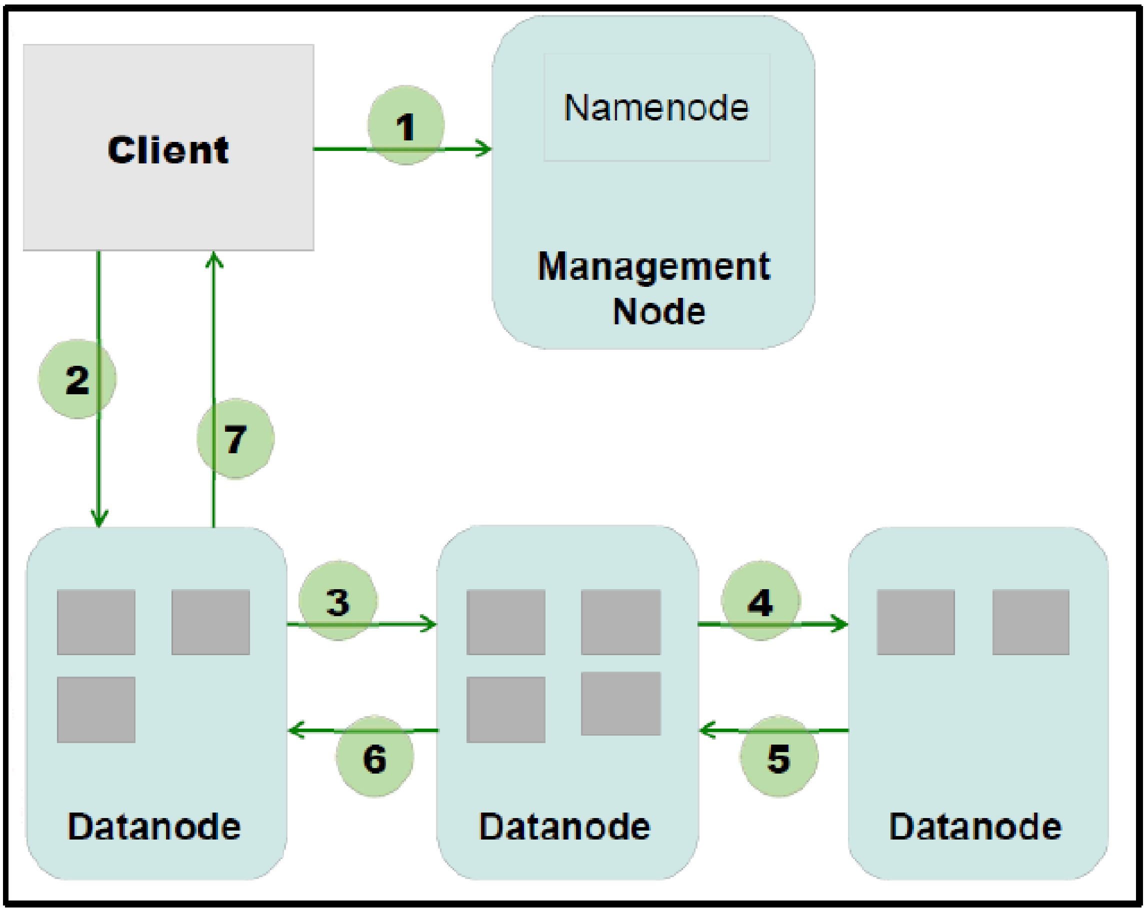


Figure 4.6: HDFS Write

4.7.2.2 HDFS Read

The read operation in HDFS is done in three steps as shown in figure 4.7.

1. Client retrieves block location from NameNode

2. Client read blocks to re-assemble the file

3. Client read blocks to re-assemble the file

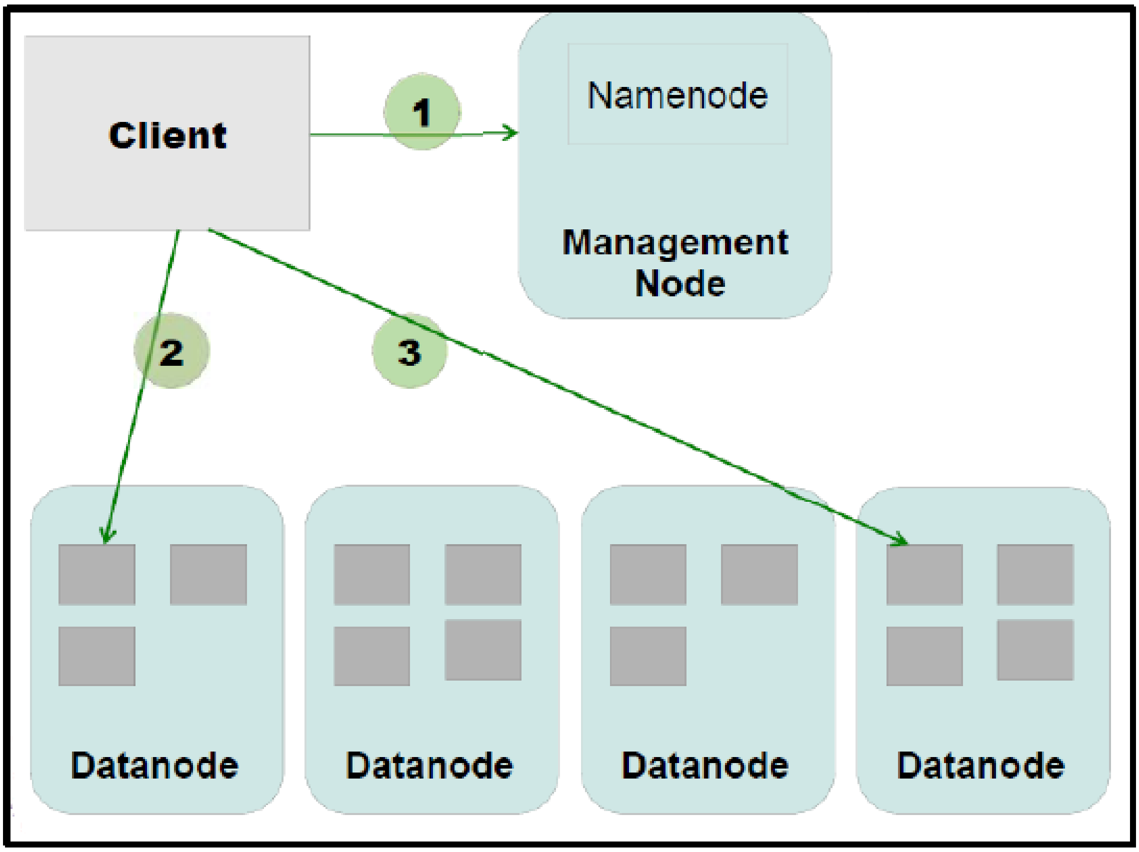


Figure 4.7: HDFS Read

CHAPTER 5

5. IMPLEMENTATION AND RESULTS

This section demonstrates the parallel implementation of Artificial Bee Colony algorithm with its application in finding the effort of a project using COCOMO Model. ABC has a bottleneck for mathematically expensive functions and thus needs to be parallelized for more efficiency.

The steps of the algorithm that can be parallelized are:

i) Each iteration of ABC algorithm can be executed in parallel
ii) Position of food source can be updated parallely

But the iterations in ABC can't be parallelized as there is a issue of message passing among the particles in each iteration. The results of previous iteration must be preserved and utilized in next iteration. Thus to transform ABC in parallel fashion following issues needs to be considered:

i) Determine the input to MapReduce architecture
ii) Determine the jobs of mapper and reducer tasks
iii) Exchanging information between mapper tasks

A large number of initial random population is provided as input to the MapReduce framework, which is then splitted into chunks and distributed across various mappers. Each food source is represented as a key-value pair as:

Key K1 : bee_id

Value V1 : set of attributes representing the food source like position, fitness, probability, etc.

❖ Creating initial population

A large number of initial random population is created, which is provided as input to the MapReduce framework. The population created is in the form of tuples containing following information: bee_id, position of food source, fitness, probaability.

Here, position of the food source is in the form of (a,b), where a and b are the coefficients of the COCOMO model used to compute effort as:

Effort (E) = a $(LOC)^b$ …………………………….**(e)**

1. Generating food source

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

   Where , i = 1,2,3,……..FS

   J =1,2,…..….D

   $x_{\max}$ and $x_{\min}$ denote the maximum and the minimum of each design variable.

2. Calculate the fitness fit_i based on the objective function value in each food source.

   $$fit_i = \begin{cases} 1/(1+f(x_i)) & f(x_i) \geq 0 \\ 1+|f(x_i)| & otherwise \end{cases}$$

   Assuming initial probabilities as 0

3. Writing these attribute using string representation into a file as:
   Food source, fitness, probability

- **Map Function**

In Parallel ABC Algorithm, the map function is called once for each particle. The key is the offset of the tuple which represents the food source and the value is the state string representation of the food source (particle).

map (key, value, context)

a) Employed_Bee_Face()

1. Calculate candidate positions vector V_i to each current food source as:

$$V_{i,j} = \begin{cases} x_{i,d} + \phi(x_{i,d} - x_{k,d}) ... if\ (j = d) \\ x_{i,j} \end{cases}$$

2. Calculate the fitness as

$$fit_i = \begin{cases} 1/(1+f(v_i)) & f(V_i) \geq 0 \\ 1+|f(v_i)| & otherwise \end{cases}$$

3. Apply greedy selection by comparing fit and fit'.

4. Update the food source as

$$x_{i,j} = \begin{cases} V_{i,j} ....if .. f(V_i) \geq 0 \\ 1+|f(V_i)|..otherwise \end{cases} \quad prob_i = fit_i / \sum_{l=1}^{NS} fit_i$$

b) Onlooker_Bee_Phase()

1. Determine the relative probabilities of each food source

$$prob_i = fit_i / \sum_{l=1}^{NS} fit_i$$

❖ Reduce Function

The reduce function in this model receives a key and a list of all associated values. Here, we have explicitly defined the key as 1 so as to run a single reducer to reduce overhead. In reduce phase the global best of all the particles is calculated and the position at which global best is obtained is stored.

reduce (key, value_list, context)

a) Scoute_Bee_Phase()

1. Compare relative probability of food source with a random probability.
2. Select the food source if its probability if greater than random probability

   Else make it abundant.

3. Check if the food source is abundant for a fixed number of cycle, then update the food source using initialization phase.
4. Check for best food source.

Repeat this cycle of map-reduce task for a fixed number of iterations or till we have converged to an optimal solution.

The best solution obtained will represent the optimal value of the coefficients a and b.

❖ Results

Here, we have optimized Basic COCOMO model parameters (a,b), such that calculated effort approximates to the actual effort for NASA 63 project dataset.

Formally, this problem can be described as finding parameter X= $\{x1, x2\}$, with $xi \in \{0,5\}$ that minimize the following equation:

MMRE = [Actual – x1(KLOC)^{x2}] /Actual…………………………………..**(f)**

Here MMRE - Mean Magnitude of Relative Error used as evaluation criteria for assessment of optimized parameters. And since parameter's x1 and x2 are specific to project mode therefore we execute program for each mode separately.

5.1.1 Environment

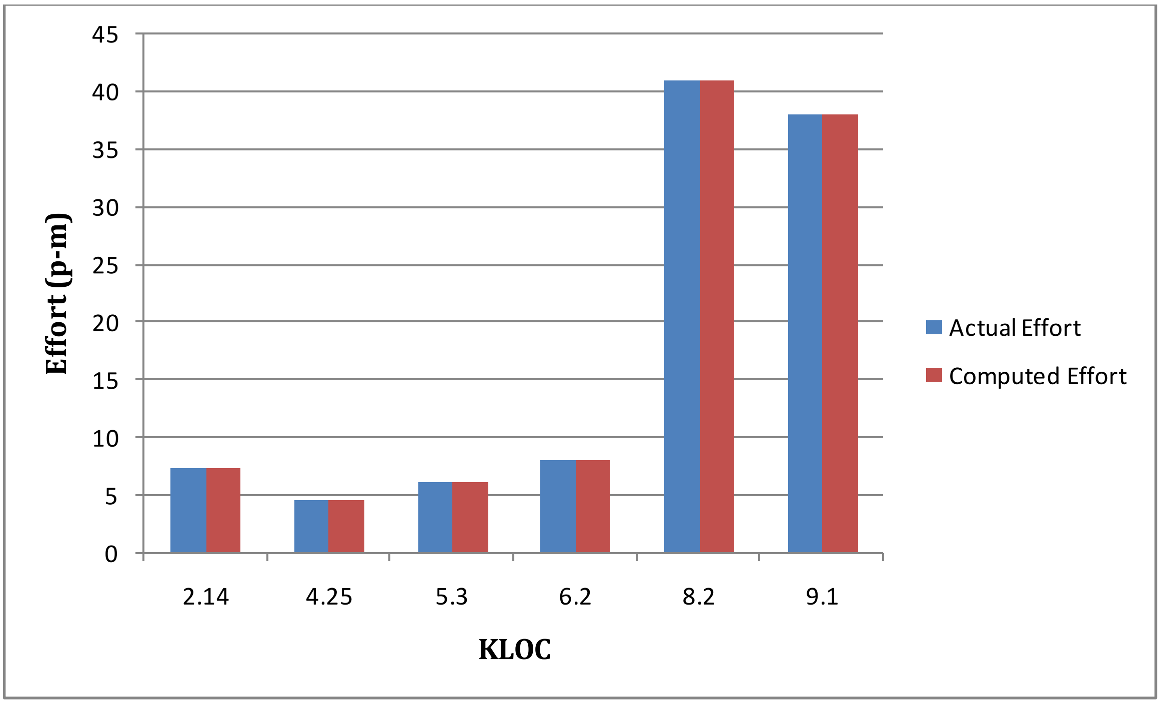
We have implemented the model on Hadoop (2.6) and ran it on our Hadoop cluster with three nodes. Each node runs a two dual Intel Quad cores, 4GB RAM and 250 GB hard disks. The nodes are integrated with Hadoop Distributed File System (HDFS) yielding a potential single image storage space of 2 *52/3 = 34.6TB (since the replication factor of HDFS is set to 3). Each node can run 5 mappers and 3 reducers in parallel.

5.1.2 Tests

We have performed three tests and obtained the following results:

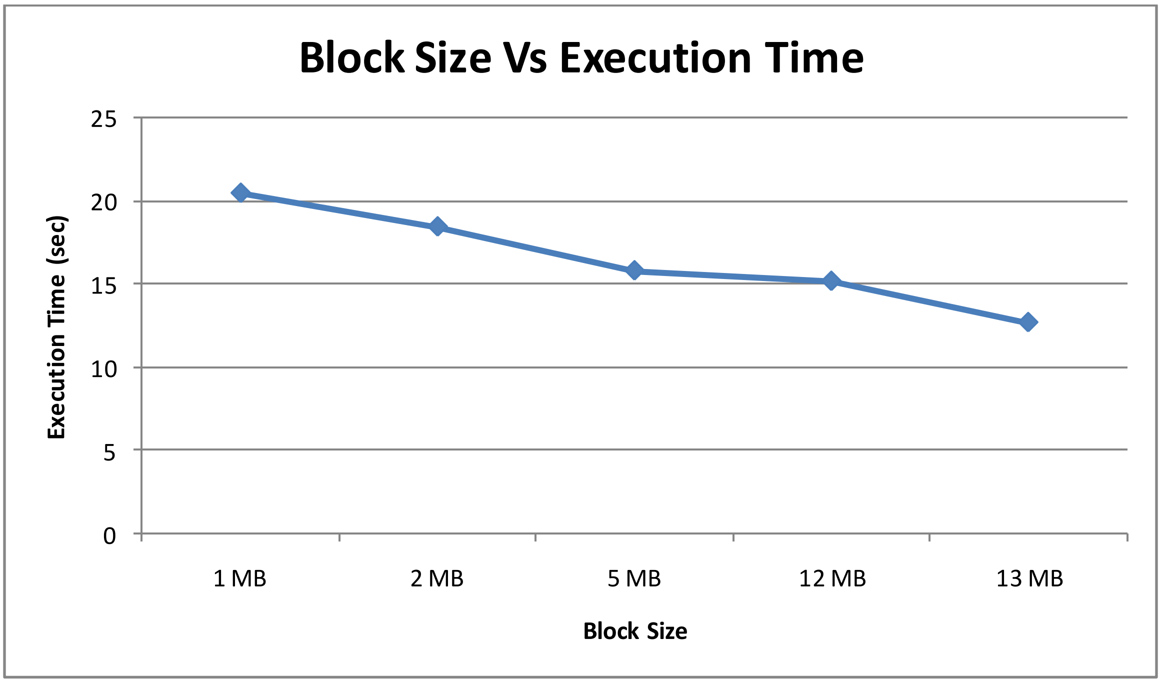
I. Comparison between actual effort and computed effort

In this experiment we have taken the population size of 2 lakhs and performed 3 iterations to obtain the result in each case.



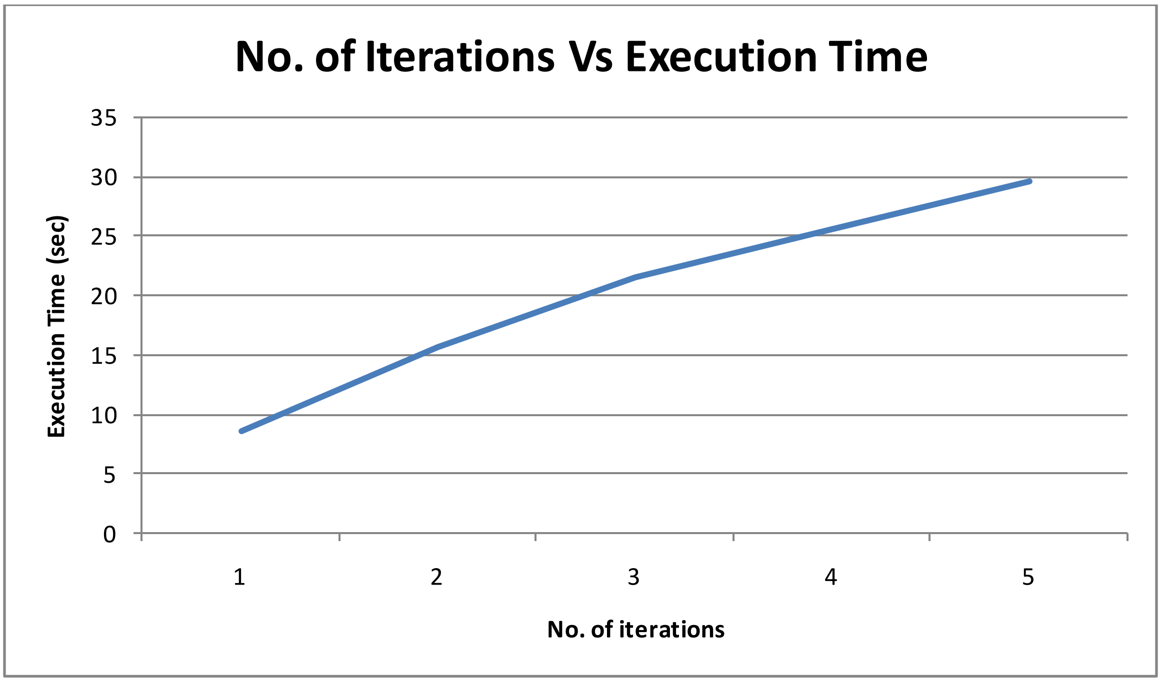
II. Variation of execution time with different block size

Taking the population size of 2 lakhs and performing 3 iterations we observed a measurable variation in execution time while changing the block size of Hadoop resulting in different number of mapper tasks in each iteration.



III. Variation of execution time with number of iterations

The execution time also varies greatly with different number of iteration. Again while performing this experiment we have kept population size of 2 lakhs and block size of 3MB.



CHAPTER 6

1. CONCLUSION AND FUTURE WORK

The Artificial Bee Colony algorithm can easily be parallelized using MapReduce architecture to solve the optimization problems involving large search space. The problem of large search space could be easily tackled by generating a large number of populations so that each particle in the population needs to search a comparatively smaller search space and can thus find the solution more efficiently in less time.

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 having small search space with less computation.

Experimental results shows that the proposed model can have better convergence than its serial implementation for intensively expensive computation functions. This model would be really efficient if we deal with solving the problem involving a large number of dimensions. In such case it would be beneficial to use this parallel implementation for faster convergence to an optimal solution.

The proposed model for parallel ABC can use a large population but could not be applied to a big dataset due to the fact that the particles keep on updating themselves in each iteration. So, a further detailed study is required to modify the algorithm in such a way so that it could be applied to big data to conclude with some meaningful information out of it.

In future work the proposed model should be modified such that it could work upon a big dataset involving large number of dimensions. Also some work could be done in order to improve the execution time of the algorithm examining other features of MapReduce architecture like partitioner, combiner etc. which may reduce the processing.