PARALLELIZATION OF PARTICLE SWARM OPTIMIZATION ALGORITHM USING HADOOP MAPREDUCE

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Priyanka Singh Ghosh

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

March 2016

Fargo, North Dakota

North Dakota State University
Graduate School

Title

Parallelization of Particle Swarm Optimization Algorithm using MapReduce

By

Priyanka Singh Ghosh

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Simone Ludwig
Chair

Dr. Saeed Salem

Dr. Canan Bilen Green

Approved:

03/21/2016
Date

Dr. Brian Slator
Department Chair

ABSTRACT

Particle Swarm Optimization (PSO) has received attention in many research fields and real-world applications for solving optimization problems in the areas of intelligent transportation systems, wireless sensor networks, finance, and engineering. Factor that affects the performance of PSO is its ability of the exploration in a multi-dimensional search space, which can increase the execution time quite significantly. The parallel implementation of PSO is a way to address this. In this paper, we implement and compare the parallel implementation of PSO using two different parallelization techniques using MapReduce programming, 1) all nodes in the cluster work on the same population, and 2) each node in cluster has its own population. Both of the parallel implementations are compared based on performance and speedup. Parallel implementation of the PSO algorithm makes the algorithm faster and scalable in order to find best solutions while working with large datasets in high dimensional search spaces.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my adviser Dr. Simone Ludwig for introducing me to the field of parallel computing and big data processing and her instrumental guidance throughout the progress of my work. I would specially like to thank her for uncomplainingly addressing to all my undeveloped questions and explaining things with great enthusiasm; it encouraged me to explore further is this field. Working with her was a great learning experience.

I am thankful to Nate Olson for always being available to support the Hadoop cluster that I used during my research.

I am grateful to my committee members Dr. Saeed Salem and Dr. Canan Bilen-Green for their invaluable time.

I would also like to mention my friends at North Dakota State University for making my stay so memorable.

DEDICATION

This paper is dedicated to my husband, Siddharth, who has been a constant source of support and encouragement during the challenges of graduate school and life. This work is also dedicated to my parents, who have loved me unconditionally and whose good examples have taught me to work hard for the things that I aspire to achieve in my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Data analytics is attracting more and more attention. Technological advancements has enabled us to capture very high volumes of data since space is not such a vital problem anymore, however, now analyzing and processing the very large amount of data (big data) is the biggest challenge. Big data is defined as the dataset whose size is beyond the processing ability of conventional database and computers. There are four main objects involved: capturing, storing, managing, and analyzing the data. Researchers have proposed many data mining algorithms to address the main objective of data analysis. However, high dimensionality of data also affects the performance of algorithms on a very high magnitude [1]. Many methods suffer from the curse of dimensionality, which implies that their performance deteriorates quickly as the dimension of the search space increases due to the computational complexity

Swarm intelligence is a set of search and optimization techniques. These algorithms are capable of exploring multi-dimensional search spaces to find optimal solutions. In order to search for the minimum or maximum in a problem domain, a swarm intelligence algorithm processes a population of individuals [2] [3]. These algorithms are population-based algorithms, which consists of a population of individuals. Each individual represents a potential solution of the problem being optimized. The population of individuals is expected to have high tendency to move in high dimensional search spaces in order to find better solutions from iteration to iteration through cooperation or competition among themselves [4]. Figure 1 shows the conceptual diagram depicting the workings of a population of individuals in a swarm. In the representation, the algorithm is initialized with random particles within a problem space and the particles are iteratively moving to find the optimum.

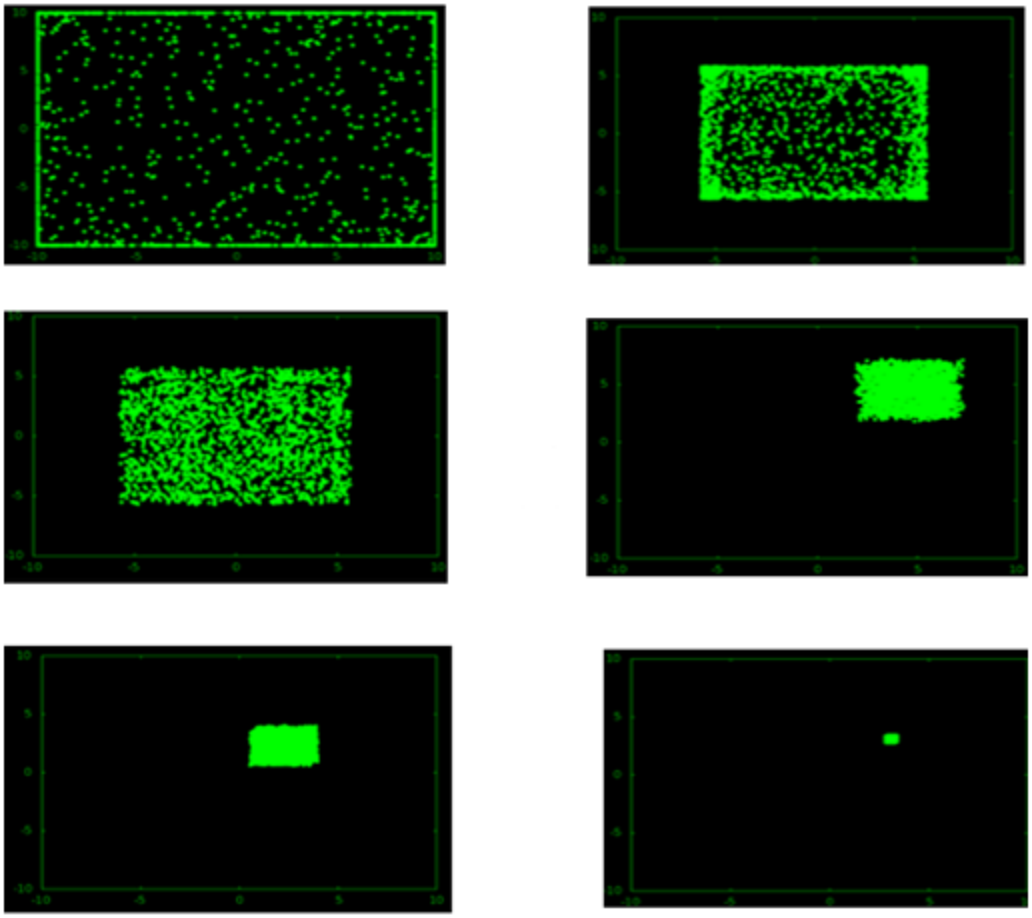

Figure 1. Swarm Intelligence [5]

However, the solution space of the problem often increases exponentially with the problem dimension and more efficient search strategies are required to explore all promising regions within a given period. The search performance of most algorithms is based upon the previous search experience. Considering the limitation of computational resources, the performance of the algorithm is affected by increasing of problem dimensions [1]. This paper concentrates on the parallelization of the Particle Swarm Optimization algorithm to optimize and compare the performance of the algorithm using two different parallelizing techniques.

1.1. Big Data

With the advancement of technology, the amount of data that is being created and stored on a global level is almost inconceivable and keeps growing. Data volumes processed by many applications crosses the peta-scale threshold and this massive volume of data is termed as big data. Figure 2 shows that the data is growing at a 40 percent compound annual rate, reaching nearly 45 Zettabytes by 2020.

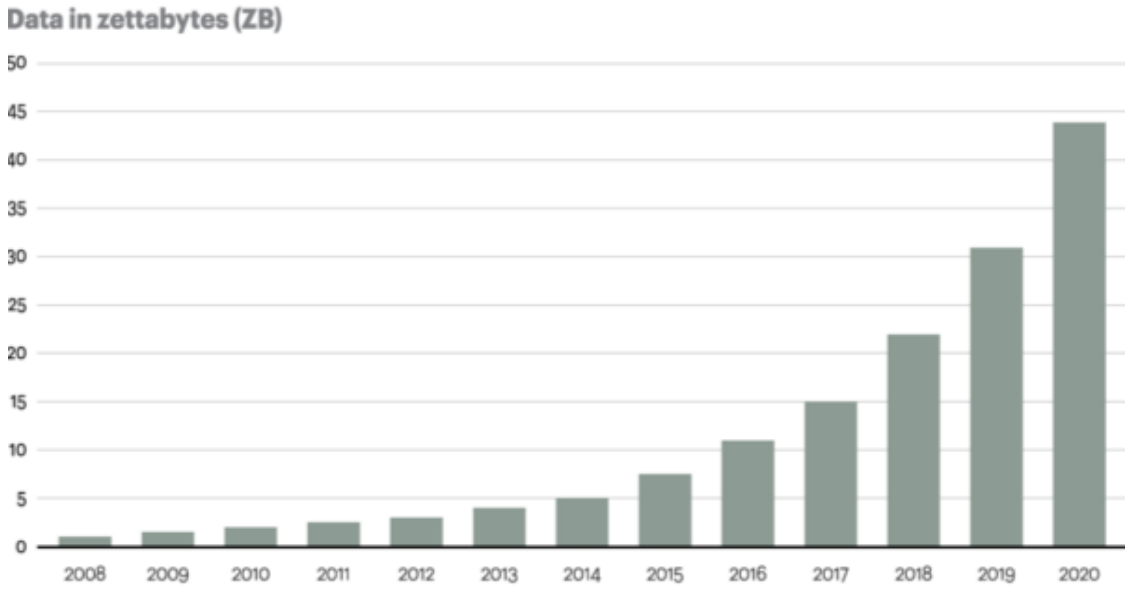

Figure 2. Big Data [6]

Big data is an assortment of data sets so large and complex that it becomes difficult to process using conventional database management tools. The challenges as mentioned before include capture, storage, search, sharing, analysis, and visualization. The trend to larger data sets is due to the additional information derivable from analysis of a single large set of related data, as compared to separate smaller sets with the same total amount of data, allowing correlations to be found such as to spot business trends, determine quality of research, prevent diseases, link legal citations, combat crime, and determine real-time roadway traffic conditions [7]. There are three main properties of big data: variety, volume and velocity.

1.1.1. Variety of Big Data

Data can be stored in multiple formats. For example database, excel, CSV files, access or it can be stored in a simple text file. There is no predefined structure for big data set. Rather, to be very specific, most of the time it is semi-structured or unstructured. Previously, we used to store data from sources like spreadsheets and databases. Now, data comes in the form of emails, photos, videos, monitoring devices, PDFs, audio, etc. It is the need of the organization to arrange the data and make it meaningful. [7][8]. It would be easy to do so if we have data in the same format, however, this is not the case most of the time. The real world has data in many different formats and that is the challenge we need to overcome when working with Big Data.

1.1.2. Velocity of Big Data

The data growth and social media explosion have changed how we look at data. Data reception is much faster than traditional data. Big data velocity deals with the pace at which data flows in from sources like business processes, machines, networks and human interaction with things like social media sites, mobile devices, etc. The flow of data is usually massive and continuous. There was a time when we used to believe that data of yesterday is recent. Today, people reply on social media to update them with the latest happenings [8]. Artificial intelligence algorithms are collecting massive amount of data from social media and websites. The data movement is now almost real time and the update window has reduced to fractions of the second.

1.1.3. Volume of Big Data

We currently see the exponential growth in the data storage as the data is now more than text data. We can find data in the form of videos, music, and large images on our social media channels. It is very common to have Terabytes and Petabytes of storage systems for enterprises [8]. Sometimes, the same data is re-evaluated from multiple angles and even though the original data is the same the new found insights create the explosion of data.

1.1.4. The Significance of Big Data is not Only Data

The knowledge discovery in datasets is the process of converting raw data into useful information. The big data may contain many varieties of unstructured or semi-structured data; these datasets need to be transformed as structured data to extract meaningful interpretation. Data mining is the process that attempts to discover useful information or patterns in large data repositories [9]. Big data requires exceptional technologies to efficiently process large quantities of data within tolerable execution times.

There are many other known technologies available today that include, crowdsourcing, data fusion and integration, genetic algorithms, machine learning, natural language processing, signal processing, simulation, time series analysis and visualization. To process and analyze the data we need technology and systems that not only perform data analysis, but then also communicate the results in minor time budget. For the most part, we know what we want out of the data [9]. We know what analysis needs to be made, what correlations need to be found, and what comparisons need to be performed. By taking what we know and putting it into the automated system that can do all of this processing and then explain the findings in natural language, we can achieve the effectiveness and scale of insight from the data. By embracing the power of parallel computing, we can speedily generate information from the data that bridge the gap between numbers and knowledge.

1.2. Hadoop

Hadoop is one of the most widely known and widely used implementation of the Map-Reduce paradigm. Google was one of the first organizations to face the problem of dealing with massive volumes of data where they wanted to download the whole Internet data and index it to support search queries [10]. They built a framework for large-scale data processing deriving from the "map" and "reduce" functions of the functional programming paradigm.

Google released two academic papers describing their technology in 2003 and 2004. The Google file system; and Map-Reduce. Doug Cutting started implementing Google MapReduce platform, as a project within the Apache open source foundation. He later joined Yahoo in 2006 and supported the Hadoop project [10].

1.2.1. Map-Reduce

Map-Reduce is a widely used parallel programming model. It is very easy to develop parallel programs to process data intensive applications on clusters of commodity machines. The Map-Reduce model represents a simplified way of parallelization for programs, which are written with the Map-Reduce paradigm. In Map-Reduce programming paradigm, the basic unit of information is a (key; value) pair where each key and each value are binary strings [11]. The input to the Map-Reduce algorithm is a set of (key; value) pairs. Operations on a set of pairs occur in three stages: the map stage, the shuffle stage, and the reduce stage.

The Map Stage: In the map stage, the mapper takes as input a single (key; value) pair and produces as output any number of new (key; value) pairs. It is essential that the map operation function works on one pair at a time. This allows for easy parallelization as different inputs for the map can be processed by different machines. It is applied in parallel to every pair in the input dataset. This produces a list of pairs for each call. After that, the Map-Reduce framework collects all pairs with the same key from all lists, and groups them together, creating one group for each key [11]. Table 1 describes the overview of the Map function.

Table 1. Map Function

	Input	**Output**
Map	<k1, v1>	List (<k2, v2>)

The Shuffle Stage: In the Shuffle phase, the underlying system that implements Map-Reduce sends all of the values that are associated with an individual key to the same node. This occurs automatically, and is seamless to the programmer.

The Reduce Stage: In the reduce stage, the reducer 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. This highlights one of the sequential aspects of the Map-Reduce computation; all of the maps need to finish before the reduce stage can begin. Since the reducer has access to all the values with the same key, it can perform sequential computations on these values. In the reduce step, the parallelism is exploited by observing that reducers operating on different keys can be executed simultaneously [11] [12]. Overall, a program in the Map-Reduce paradigm usually consists of many rounds of different map and reduce functions performed one after another. Table 2 represents the concept of the reduce function.

Table 2. Reduce Function

	Input	**Output**
Reduce	<k2, list(v2)>	List (<k3, v3>)

Each Reduce call typically produces either one value v3 or an empty return, though one call is allowed to return more than one value. The returns of all calls are collected as the desired result list. Thus, the Map-Reduce framework transforms a list of (key, value) pairs into a list of values.

Programs written in this functional arrangement are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines,

handling machine failures, and managing the required inter-machine communication [11] [12]. The map reduce strategy is show in Figure 3.

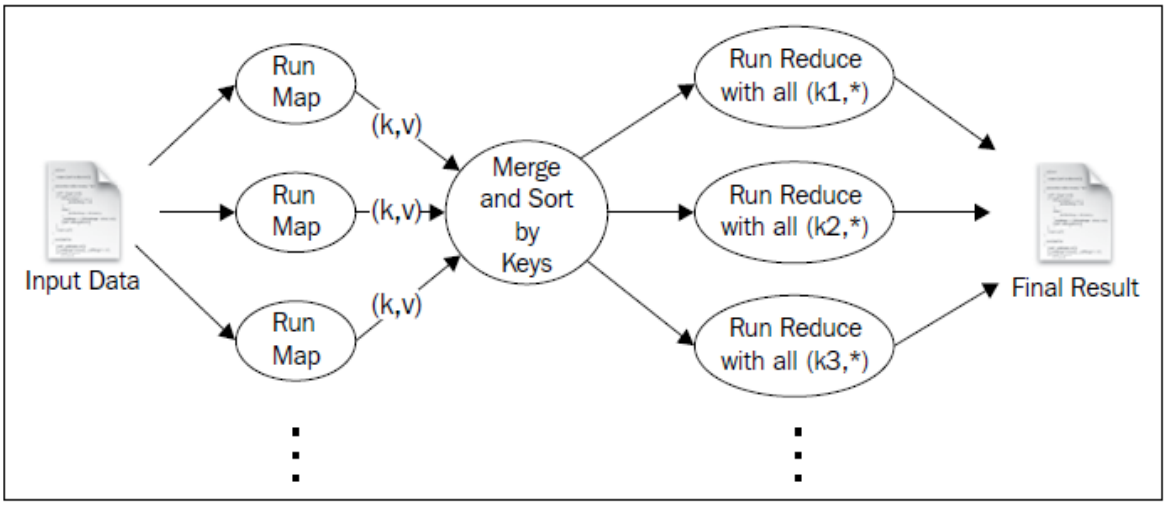

Figure 3. Map reduce programming model [13]

To illustrate the concept of Map-Reduce with the help of an example, Figure 4 shows a simple word count example, given an input file, it is required to compute the frequency of each word in the file using the Map-Reduce strategy.

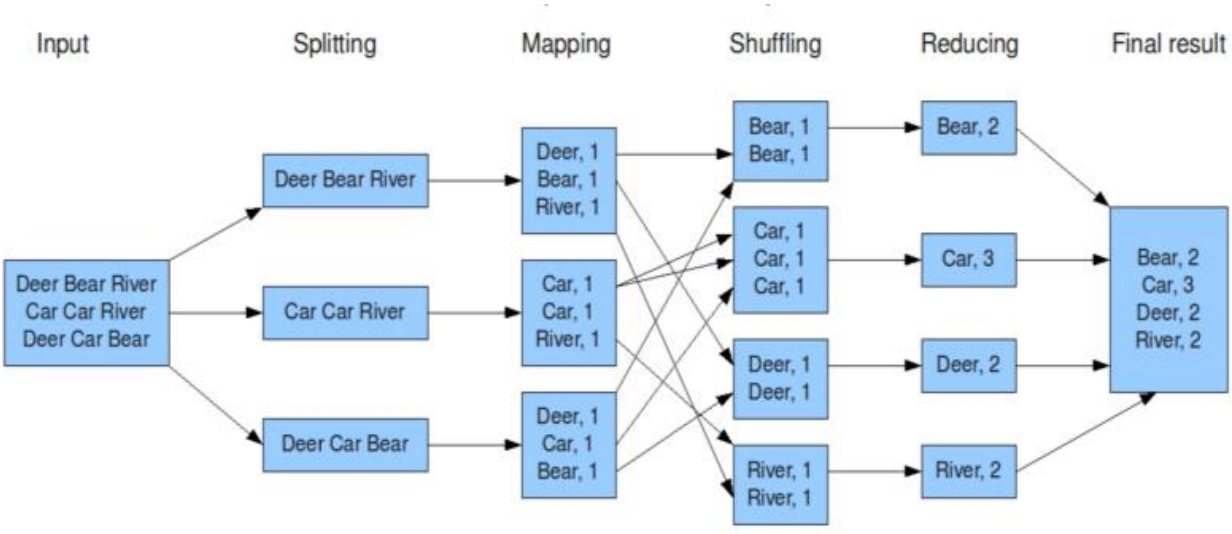

Figure 4. Map reduce flow of simple Word count program [14]

1.2.2. HDFS

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS was originally built as infrastructure for the Apache web search engine project. HDFS is now an Apache Hadoop subproject [15].

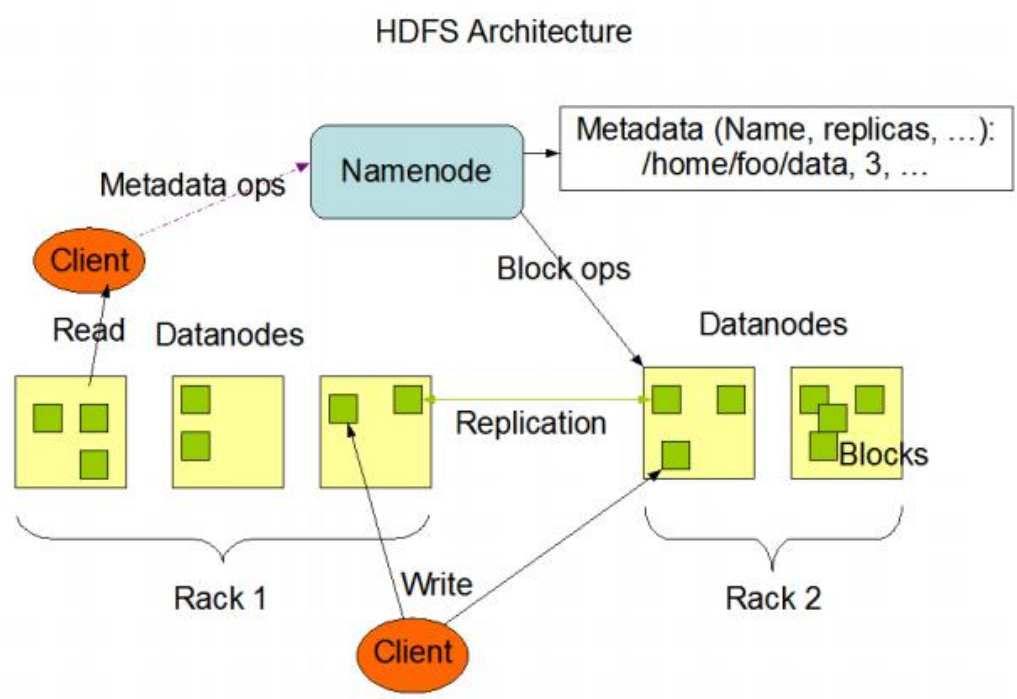

Figure 5. HDFS Architecture [15]

Figure 5 shows that HDFS consists of a master-slave architecture. An HDFS cluster consists of a single Name-Node, a master server that manages the file system namespace and regulates access to files by clients [15]. In addition, there are a number of Data-Nodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of Data-Nodes.

The Name-Node executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to Data-Nodes. The Data-Nodes are responsible for serving read and write requests from the file system's clients. The Data-Nodes also perform block creation, deletion, and replication upon instruction from the Name-Node [15]. A typical deployment has a dedicated machine that runs only the Name-Node software. Each of the other machines in the cluster runs one instance of the Data-Node software. The architecture does not preclude running multiple Data-Nodes on the same machine but in a real deployment that is rarely the case [15]. The existence of a single Name-Node in a cluster greatly simplifies the architecture of the system. The Name-Node is the negotiator and repository for all HDFS metadata.

2. BACKGROUND

Many real-world applications can be characterized as optimization problems of which algorithms are required to have the capability to search for the optimal solution. Most traditional methods can only be applied to continuous and differentiable functions. Swarm intelligence is the study of computational systems inspired by the collective intelligence [16].

Collective intelligence emerges through the cooperation of large numbers of homogeneous agents in the environment. Examples include flocks of birds, and colonies of ants. Such intelligence is decentralized, self-organizing and distributed throughout an environment. In nature such systems are commonly used to solve problems such as effective foraging for food, prey evading, or colony re-location [2] [17]. The information is typically stored throughout the participating homogeneous agents, or is stored or communicated in the environment itself such as through the use of pheromones in ants, dancing in bees, and proximity in fish and birds.

The swarm intelligence paradigm consists of two leading sub-fields 1) Ant Colony Optimization that investigates probabilistic algorithms inspired by the foraging behavior of ants, and, 2) Particle Swarm Optimization that investigates probabilistic algorithms inspired by the flocking, schooling and herding [3]. Like evolutionary computation, swarm intelligence algorithms are considered adaptive strategies and are typically applied to search in optimization domains.

At the beginning of the search, the solutions are initialized randomly in the search space, and are guided toward the better solutions through the interaction among the individuals over many iterations. As a general principle, the expected fitness of a solution returned should improve as the search method proceeds, i.e., in any single run, the quality of the solution returned by the method over iterations should improve. The quality of the solution at time t + 1

should be no worse than the quality at time t, i.e., f fitness (t + 1) ≤ f fitness (t) for minimization problems. There are many variations of swarm intelligence algorithms available, in this paper we are concentrating on the Particle Swarm Optimization (PSO) algorithm, which was originally designed for solving continuous optimization problems.

2.1. Particle Swarm Optimization

PSO is a population based stochastic optimization technique developed by Eberhart and Kennedy in 1995, inspired by the social behavior of bird flocking or fish schooling. PSO can be implemented and applied easily to solve various function optimization problems, or problems that can be transformed to function optimization problems [2] [18]. As an algorithm, the main strength of PSO is its fast convergence, which compares favorably with many global optimization algorithms like Genetic Algorithms.

In PSO, individuals are referred to as particles and the population is called a swarm. A PSO algorithm maintains a swarm of particles, where each particle represents a potential solution. Each particle is given a position and a velocity. Once a particle finds a good direction to move, other particles are notified and will be able to steer toward that direction immediately [18]. The particles roam in the space, convey their positions to each other, and adjust their own positions and velocities based on these “better” positions.

Changes to the position of particles within the search space are based on the social-psychological tendency of individuals to emulate the success of other individuals. The changes to a particle within the swarm are therefore influenced by the experience, or knowledge, of its neighbors [4] [19]. The search behavior of a particle is thus affected by that of other particles within the swarm. Figure 6 depicts how particles move in a 2-dimensional search space to reach the single global solution.

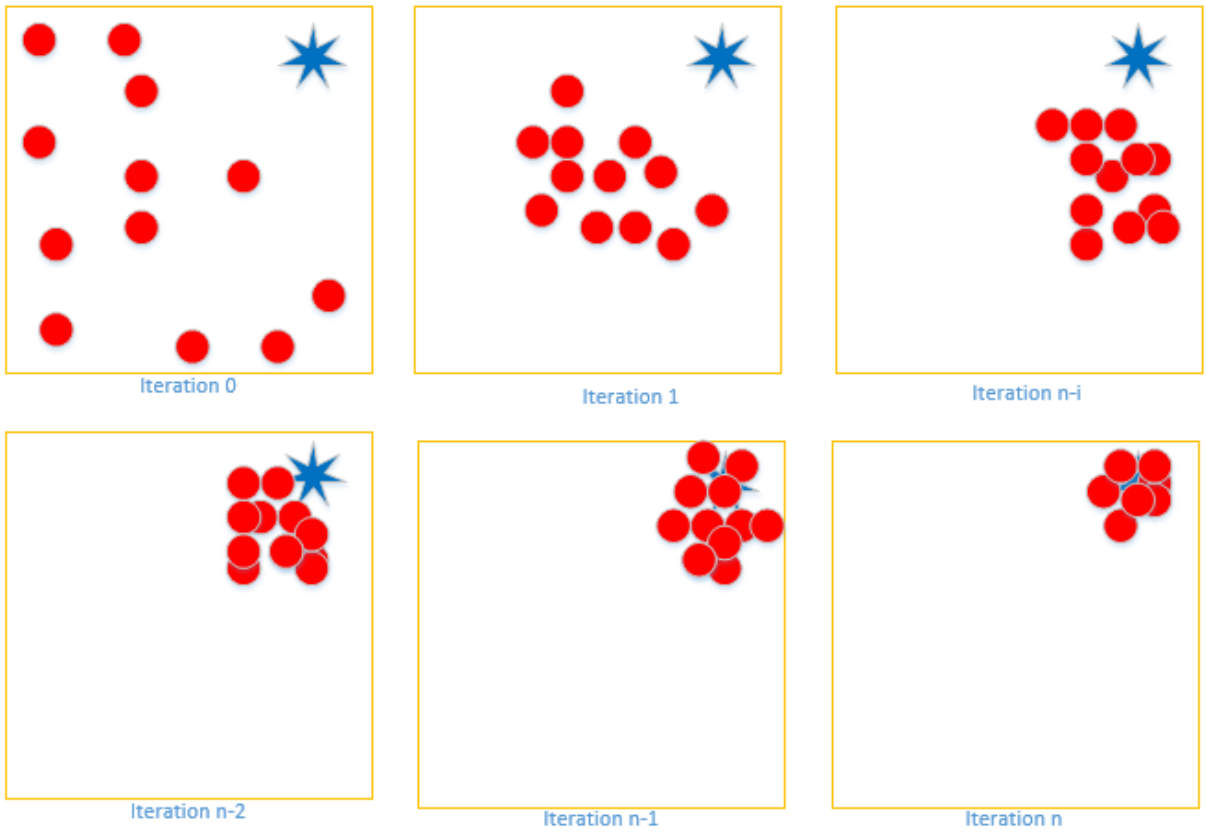

Figure 6. Conceptual Diagram of PSO in action with a global best solution

The particles are "flown" through a multi-dimensional search space, where the position of each particle is adjusted according to its own experience and that of its neighbors. Initially, particles are distributed throughout the search space and their positions and velocities are modified based on the knowledge of the best solution found so far by each particle in the 'swarm' [20]. Attraction towards the best-found solution occurs stochastically, and uses dynamically-adjusted particle velocities.

Let xi (n) denotes the position of particle i in the search space at time step n; where n denotes discrete time steps. The position of the particle is changed by adding a velocity, vi (n), to the current position.

$$xi\ (n + 1) = xi\ (n) + vi\ (n + 1) \quad (1)$$

The system is initialized with a population of random solutions that searches for the optimum by updating their positions. Each particle keeps track of its coordinates in the problem space, which are associated with the best fitness solution it has achieved so far [19]. Another best value that is tracked by the particle swarm optimizer is the best value obtained so far by any particle in the swarm. When a particle takes all the population, the best value is a global. In PSO, at each time step, the velocity of each particle to achieve its global best position, and personal best position is updated.

The position of the ith particle xii = (xi1, xi2 … xiD)

The velocity for the ith particle vi = (vi1, vi2,…, viD)

The positions and velocities of these particles are confined within (xmin, xmax) D, and (vmin, vmax) D, respectively [20].

The best encountered position of the ith particle, denoted as Pbest i, is represented by pi= (pi1, pi2,…, piD). The best value of all the individual Pbest i values are denoted as the global best position g = (g1, g2, …, gD) called Gbest. The PSO process is initialized with a population of random particles, and the algorithm executes a search for optimal solutions by continuously updating the positions of the particles. At each time of the generation, the position and velocity of the ith particle are updated by Pbest and Gbest of the population [21]. The updated equations are formulated as follow

$$xij\ (n+1) = xij\ (n) + vij\ (n+1) \tag{2}$$

$$vij(n+1) = w(n)*vij(n) + c1ij*r1ij*[PBestij(n)-xij(n)] + c2ij*r2ij*[GBestij(n)–xij(n)] \tag{3}$$

Where,

xij = Position of particle i

j = Dimension

PBestij (n) - xij (n) calculates a vector directed towards the personal best position GBestij (n) - xij (n) calculates a vector directed towards the global best position, and the inertia weight w, the personal best weight c1, the global best weight c2. Both r1ij, r2ij are random numbers uniformly distributed between 0 and 1. The pseudo code of PSO is shown below.

Algorithm 1. PSO

01: Begin
02: Randomly initialize particle swarm
03: While (number of iterations<=Max iterations)
04: Evaluate fitness of the each particle
05: For n = 1 to the number of particles
06: Find Pbest
07: Find Gbest
08: For d = 1 to the number of dimension of particle
09: update the position of particles
10: Next d
11: Next n
12: Next generation until the stopping criterion is met
13: End

2.2. Advantageous Characteristics of Particle Swarm Optimization Algorithm

i. Particle Swarm Optimization algorithm is based on swarm intelligence. It can easily be applied into wide variety of optimization problems including scientific research and engineering use.

ii. The Particle Swarm Optimization algorithm has no overlapping and mutation calculation. The search can be carried out by the speed of the particle. During the development of several generations, only the most optimist particle can transmit information onto the other particles, and the speed of the updates is very fast [21].

iii. The calculations of the Particle Swarm Optimization algorithm are very simple [21].

3. PARALLELIZATION OF PARTICLE SWARM OPTIMIZATION ALGORITHM

Optimization problems involve large amounts of data, such as web content, commercial transaction information, or bioinformatics data. Solving complex and difficult optimization problems are very challenging. Individual function evaluations may take several minutes or even hours [22]. The PSO algorithm has been used increasingly as an effective technique for solving such problems. Although PSO has many advantages, it has the downside of a long runtime to find solutions for large-scale problems in multidimensional search spaces.

One of the main reasons of this problem is that the optimization process of PSO requires a large number of fitness evaluations, which is the most time consuming part and is usually done sequentially [23]. One machine is not capable to resolve problems of this magnitude; parallel implementation of Particle swarm optimization is one of the available solutions. For the parallelization, Hadoop is one of the most widely known and used runtime environment using the MapReduce paradigm. The PSO algorithm can be expressed with MapReduce and developed as a simple and robust parallel implementation [24].

There are different parallelization methods available to implement algorithms, and the two most popular techniques [25] that we are considering for our implementations are:

i) Parallelization on the algorithm level, i.e., an entire population is evaluated per node.

ii) Parallelization on the population level, i.e., the population is split among different nodes.

The first implementation method of parallelization uses an entire population per node and evaluates the entire fitness of this population as presented in [26]. The second implementation splits the population to different nodes, and each node processes a portion of the population [27].

3.1. Our Contribution

In this paper, we implement the PSO algorithm with MapReduce using both methods, 1) all the nodes in the cluster work on an entire population, and 2) each node in cluster has a portion of the entire population. We analyzed the performance of both implementations on a Hadoop cluster measuring the execution time and speedup.

3.2. The PSO Algorithm

The algorithm that we implemented as a data-intensive flow is as follows:

1. Initialize the population (swarm) with random individuals (particles)
2. For each particle, calculate the fitness value
3. If the fitness value is better than the best fitness (Pbest) in history
4. Set current value as new Pbest
5. Calculate particle velocity
6. Update particle position
7. Choose the particle with the best fitness value of all the particles as the Gbest for each particle.
8. Repeat Steps 2-7 until stopping criterion (maximum number of iterations) is met.

3.3. Implementation I: Parallelization on Algorithm Level

In this implementation, we generate the swarm of particles and provide it as input. In addition, all the nodes of the Hadoop cluster work on an entire population of particles independently. The mappers will take the complete population and evaluate the fitness of the particles.

3.3.1. Map Function

The Map evaluates the fitness of the given particles and update their positions. It also keeps track of the best fitness achieved and then passes it to reducer.

Algorithm 2. Map()

***Input**: Particles and their Positions*
***Output**: Gbest*
01: Begin
02: While (number of iterations<=max. iterations)
03: Evaluate fitness of each of the particle
04: For n = 1 to the number of particles
05: Find Pbest and Gbest
06: For d = 1 to the number of dimension of particle
07: update the position of the particle
08: Next d
09: Next n
10: End while
11: Send Gbest to reducer
12: End

3.3.2. Reducer Function

The Reducer function collects the Gbest values from all algorithm runs that are performed on different nodes in the cluster. After comparing all Gbest values, the reducer returns the best Gbest value and writes it to HDFS.

Algorithm 3. Reduce()

***Input**: Gbest values*
***Output**: Gbest value*
Out of all of the Gbest values, find overall best Gbest value
01: Begin
02: For n = 1 to the number of runs
03: If FitnessValue(n) >FitnessValue(N+1) Then Update Gbest
04: Next n
05: End
Emit (Gbest)
Write (Gbest) to HDFS

3.4. Implementation II: Parallelization on Population Level

In the second implementation, we split an entire population to several nodes available on the cluster. In the first iteration, we randomly generate particles of a swarm and write the data into HDFS; the input file contains the particle with their positions.

3.4.1. Initialization

Algorithm 4. Initialization

Input*: Population size, no. of dimensions*
Output*: Particles and their positions*
01: Begin
02:PopulationSize = SpecifyPopulationSize()
03:NumberofMaps = SpecifyNo.ofMaps()
04:GeneratePopulation(PopulationSize)
05: Write Population to HDFS
06: While (number of iterations<=max. iterations)
07: Map and Reduce functions are called
08:End While
09: End
Write the data to HDFS

3.4.2. Map Function

We split the input file and each mapper takes a fragment of file (i.e., portion of the population) and evaluates several particles in a mapper by evaluating the fitness of each particle (Pbest). Each particle position is updated and written back to the input file in HDFS, and each particle's Pbest is sent to the Reducer.

Algorithm 5. Map()

Input*: File containing particles*
Output*: Pbest*
01: Begin
03: Evaluate fitness of each of particle
04: Update Pbest if better
05:Write particle to HDFS
06: Send Pbest to Reducer
07: End

3.4.3. Reducer Function

The Reducer function collects the Pbest values after each iteration and compares all the Pbest values to identify Gbest, which is written to HDFS. This value represents the best value achieve so far. When a better Gbest value comes in, the reducer replaces the old Gbest value that was saved in the output file with the new Gbest value achieved.

Algorithm 6. Reduce()

Input*: Pbest values*
Output*: Gbest value*
Out of all of the Pbest values find Gbest value
01: Begin
02: For n = 1 to the number of particles
03: Update Gbest if Pbest is better
04:End For
05: Update Gbest to HDFS
06: End
Emit (Gbest)
Write (Gbest) to HDFS

4. EXPERIMENTS, RESULTS & OBSERVATIONS

In this section, we present the implementation details, the experiments that were performed with outcomes of both MapReduce implementations of the PSO algorithm. The parallel algorithms were executed on the departmental Hadoop cluster 2.7.1. In order to evaluate the algorithm, the Rastrigin benchmark function is used.

4.1. Benchmark Function

In the field of evolutionary computation, benchmark functions provide a good way to evaluate an optimization algorithm. We are using the Rastrigin benchmark function for evaluating our algorithm implementations. The Rastrigin function is a highly multimodal function where the degree of multimodality increases with the increase in the dimension of the problem. The Rastrigin function is based on the function of De Jong with the addition of Cosine modulation in order to produce frequent local minima and is defined as follows:

$$\mathrm{F(x)} = \sum_{i=1}^{D} g1(xi)2 - 10\cos(2\pi g1(xi)) + 10 \qquad (4)$$

Where,

D is the dimension

xi = (x1, x2, … , xd) is a D-dimensional row vector (i.e., a 1×D matrix).

The Rastrigin function is a classical multimodal problem. It is difficult since the number of local optima grows exponentially with the increase of dimensionality. Test area is usually restricted to the hypercube -5.12≤ Xi ≤ 5.12, i=1,…,n. Its global minimum F(x) = 0 is obtainable for xi=0, i= 1,…,n. Figure 7 shows a 2-dimensional representation of the Rastrigin function.

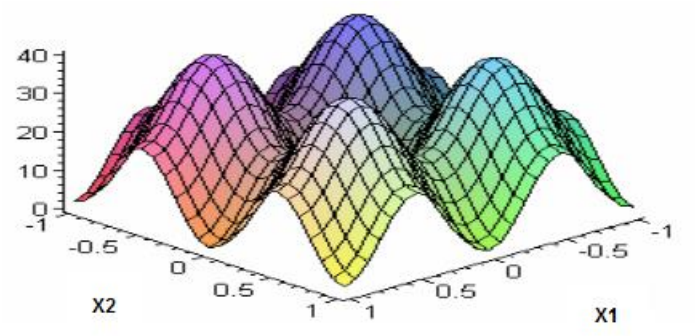

Figure 7. Rastrigin's Function [28]

4.2. Evaluation Measures

We have used the execution time speedup measure to evaluate the performance of the algorithm implementations by continuously increasing the number of computational nodes and keeping the dimensionality of the problem constant. The speedup is calculated by:

$$\text{Speedup} = \frac{T2}{Tn} \tag{5}$$

Where,

T2 is the running time using 2 nodes

Tn is the running time using n nodes, where n is a multiple of 2.

4.3. Experiments

Our experiments involve both implementations of the PSO algorithm. We have tested the performance of both the MapReduce implementations on the Hadoop cluster 2.7.1, Java version 1.6 with the focus on speedup and performance.

4.3.1. Case 1: Execution Time and Speedup with Fixed Population Sizes

In this experiment, we keep the problem size fixed and increase the number of mappers. We measure the execution time and calculate the speedup using Equation (5).

4.3.2. Case 2: Execution Time and Speedup with Increasing Population Sizes

In this experiment, we increase the problem size (dimension) and observe the performance of algorithm while utilizing the maximum available resources.

4.3.3. Case 3: Execution Time with Increasing Population Sizes and Keeping Other Variables Fixed

In this experiment, we increase the population size and keep the number of nodes, dimension and number of iterations fixed.

For above mentioned three cases, we have performed the following experiments:

4.3.4. Experiment – 1

We ran both implementations of PSO algorithm for the following setting:

5,000 iterations, 50,000 particles, 50 dimensions. Evaluation function: Rastrigin

Figure 8 shows the execution time of both implementations while increasing the cluster nodes with multiples of 2. Figure 9 shows the speedup achieved by both implementations. As can be noted from the figure, the speed up achieved by Implementations I and II are 5.4 and 4.1, respectively.

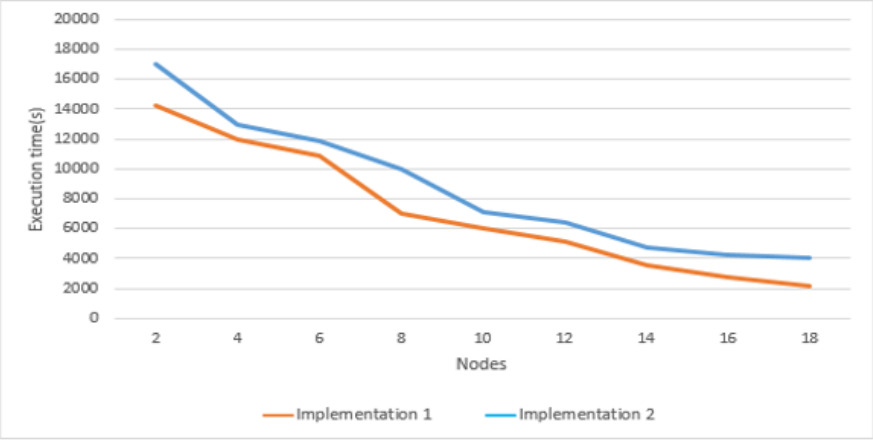

Figure 8. Execution time – Experiment 1

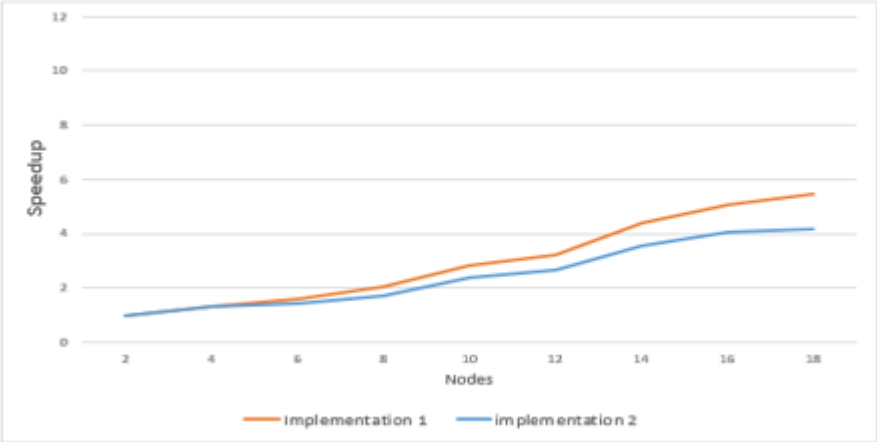

Figure 9. Speedup - Experiment 1

4.3.5. Experiment - 2

We ran both implementations of the PSO algorithm for the following setting: 10,000 Iterations, 100,000 Particles, 70 dimensions. Evaluation function: Rastrigin

Figure 10 shows the execution time of both implementations while increasing the cluster nodes with multiples of 2. Figure 11 shows the speedup achieved by both implementations. The speedup achieved by Implementation I and II are 6.2 and 4.3, respectively.

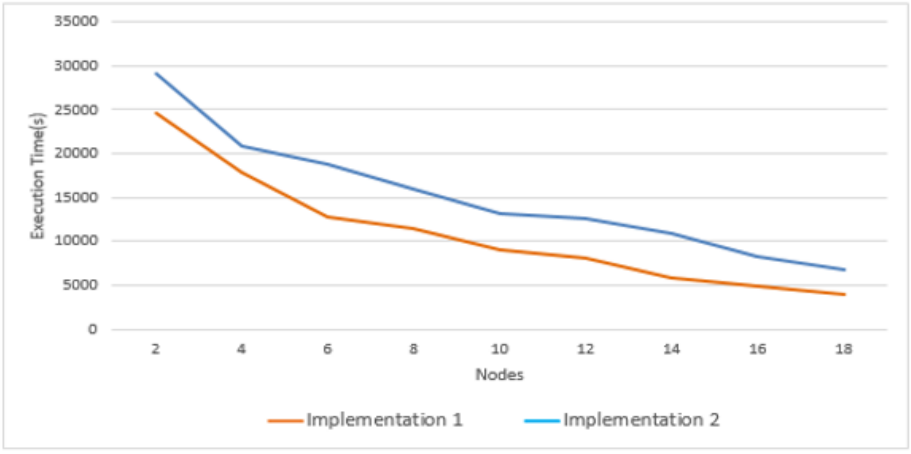

Figure 10. Execution time – Experiment 2

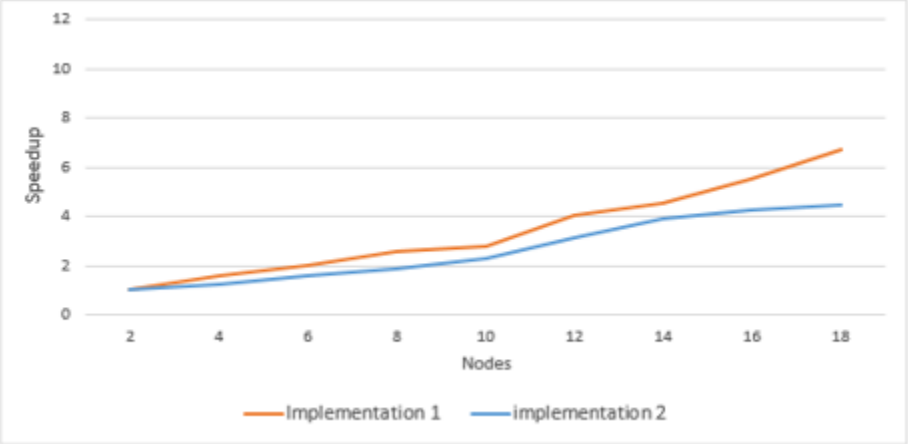

Figure 11. Speedup – Experiment 2

4.3.6. Experiment - 3

For experiment 3, we ran both the implementations of PSO algorithm for the following setting: 12,000 Iterations, 120,000 Particles, 60 dimensions. Evaluation function: Rastrigin

Figure 12 shows the total execution time of both implementations while increasing the cluster nodes with multiples of 2. Figure 13 shows the speedup achieved by both implementations. The figure represents the speedup achieved by implementation I and II, which are 6.6 and 4.5, respectively.

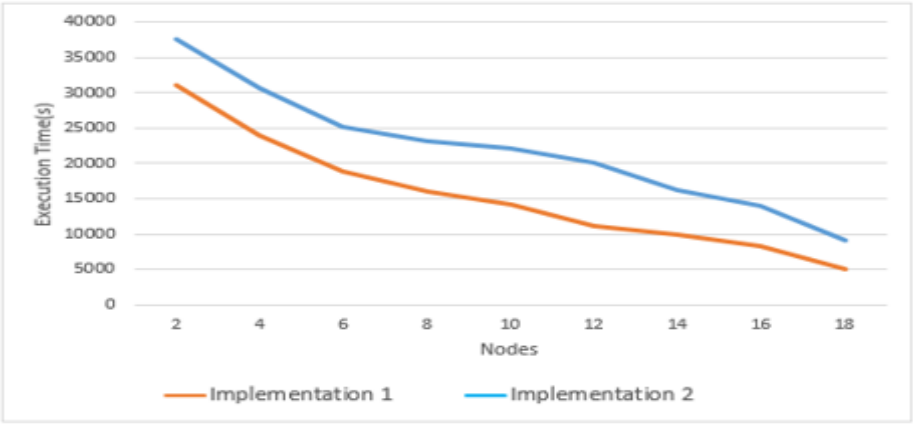

Figure 12. Execution time – Experiment 3

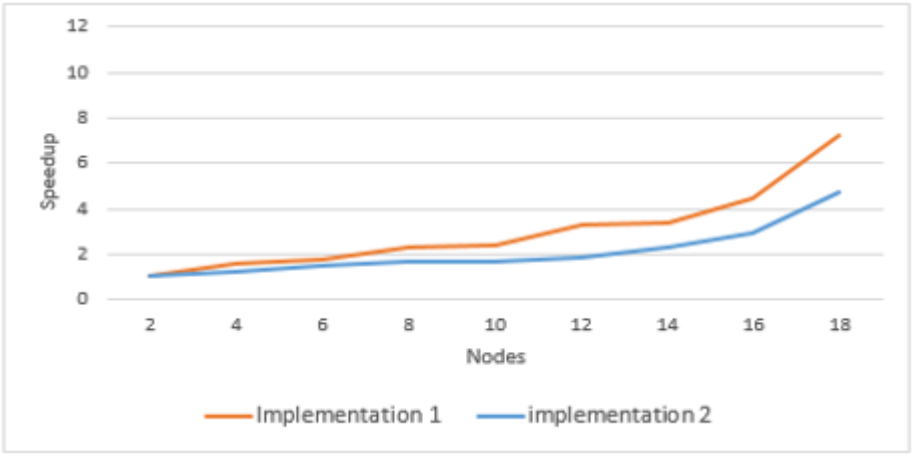

Figure 13. Speedup – Experiment 3

4.3.7. Experiment - 4

For experiment 4, we ran both implementations of the PSO algorithm for the following setting: 14,000 iterations, 140,000 particles, 70 dimensions. Evaluation function: Rastrigin

Figure 14 shows the total execution time of both implementations while increasing the cluster nodes with multiples of 2. Figure 15 shows the speedup achieved by both implementations. As can be noted from the figure, the speedup achieved by Implementation I and II are 6.9 and 4.6, respectively.

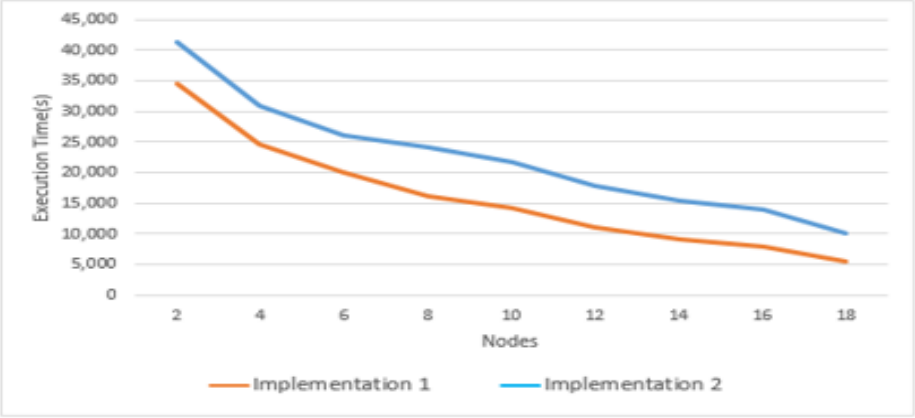

Figure 14. Execution time – Experiment 4

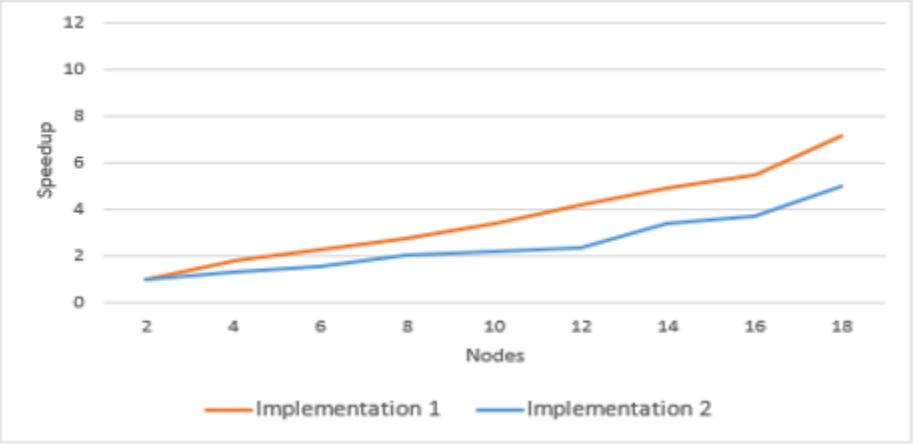

Figure 15. Speedup – Experiment 4

4.3.8. Experiment - 5

In experiment 5, we ran both PSO implementations for the following setting:

Particles: 100,000, 150,000, 200,000, 250,000, 300,000

Fixed parameters: 6,000 iterations, 10 nodes, 50 dimensions

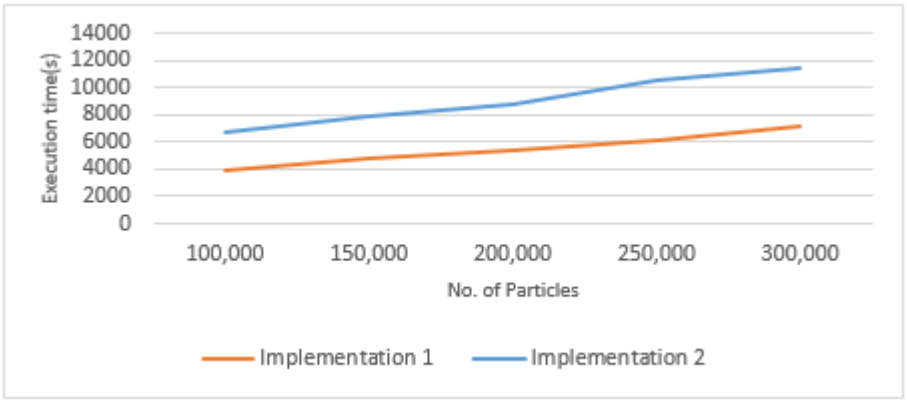

Figure 16. Execution time – Experiment 5

4.4. Results

We tested both implementations on the basis of speedup and performance, using three experiments, measuring speedup with fixed population size, speedup with increasing population size and execution time with increasing population size while keeping number of nodes, dimensions and iterations constant. Considering all three experiments, we observed that for both

Hadoop implementations the performance has improved when we increased the number of nodes. Thus, adding more resources while keeping the problem size fixed and also with increasing the population size decreases the execution time. However, with increasing population size and keeping number of nodes, iterations and dimensions constant gradually increases the execution time. We also observed that the performance of the first implementation is faster than the second implementation. For larger problem dimensions, if we would increase the hardware and resources, both implementations can scale well.

5. CONCLUSION & FUTURE WORK

The performance of algorithms degrades significantly as the dimension of the search space increases. In this paper, we parallelized the PSO algorithm using MapReduce and executed the code on Hadoop, which is the most widely used implementation of the MapReduce programming paradigm. We used two different techniques for the parallelization, 1) entire population is evaluated on one node, 2) each node in cluster evaluates a portion of the population.

In the first implementation, all the nodes available in the cluster receive an entire population as the input and work on this data. In the second implementation, each node in the cluster has a portion of the population. In this implementation, the input file (population) gets divided among nodes and each node available on the cluster gets its own part of the population to perform the evaluations.

We analyzed both models using different combinations of iterations, population size, dimensions, and nodes. Based on our experiments, we realized that both parallel implementations can easily be scaled and used for large swarm populations to address more complex optimization problems. If we want to scale the problem size, we can easily add the hardware and scale both implementations. We also observed that the performance of the first implementation, parallelization at the algorithm level, performs faster than the second implementation where we implemented parallelization on the population level. Regardless, both implementations showed promising results and scalable nature to provide feasible solution for complex optimization problems in a reasonable time.

As for future work, we would like to implement the PSO algorithm using the Apache spark framework, which is a promising framework for iterative programming to observe the differences.

6. REFERENCES

[1] S. Cheng, Y. Shi, Q. Qin and R. Bai, "Swarm Intelligence in Big Data Analytics", *Intelligent Data Engineering and Automated Learning"*, p. 417-426, 2013.

[2] J. Brownlee, "Clever Algorithms: Nature-Inspired Programming Recipes", Lulu.com publication, p. 414, 2011.

[3] A. P. Engelbrecht, Computational Intelligence An Introduction, 2nd ed. John Wiley & Sons Ltd, p. 630, 2007.

[4] J. Kennedy, R. Eberhart. "Particle Swarm Optimization". *Proceedings of IEEE International Conference on Neural Networks*, p. 1942–1948, 1995.

[5] D. Puhes, "Swarm Intelligence V0.8.1", 2016 [Online]. Available: Lab.cmikavac.net

[6] S. Vale, "Big Data - MSIS Wiki - UNECE Statistics Wikis", Www1.unece.org, 2016. [Online]. Available: http://www1.unece.org/stat/platform/display/msis/Big+Data.

[7] C. Hagen, A. Yadav, M. Ciobo, D. Wall, H. Evans, J. Miller and K. Khan, "Big data and the creative destruction of todays business models", Slideshare.net, 2013. [Online]. Available: http://www.slideshare.net/mciobo/big-data-and-the-creative-destruction-of-todays-business-models.

[8] P. Dave, "Big Data - Evolution of Big Data - Day 3 of 21 - Journey to SQL Authority with Pinal Dave", Journey to SQL Authority with Pinal Dave, 2013. [Online]. Available: http://blog.sqlauthority.com/2013/10/03/big-data-evolution-of-big-data-day-3-of-21/.

[9] J. Min-Zheng. "Research On The Performance Optimization Of Hadoop In Big Data Environment". International Journal of Database Theory and Application 8.5, p1-12, 2015

[10] V. Vijayalakshmi, A Akila, and S Nagadivya. "The Survey On Map Reduce". International Journal of Engineering Science and Technology 4.7, p 1-9, 2012.

[11] M. Usuelli, "An example of MapReduce with rmr2 - MilanoR", MilanoR, 2013. [Online]. Available: http://www.milanor.net/blog/an-example-of-mapreduce-with-rmr2/.

[12] A. McNabb, C. Monson, and K. Seppi. "Parallel PSO Using Mapreduce". IEEE Congress on Evolutionary Computation, p1-9, 2007.

[13] S. Perera and T. Gunarathne, Hadoop MapReduce Cookbook. PACKT Publication, p. 300, 2013.

[14] A. Sagar, "Distributed Computation of spatial data on Hadoop Cluster", 2016. [Online]. Available: http://www.gise.cse.iitb.ac.in/wiki/images/e/e4/MPT-Stage1-Report.pdf.

[15] D. Borthakur, "HDFS Architecture Guide", 2008. [Online]. Available: https://Hadoop. apache.org/docs/r1.2.1/hdfs_design.

[16] A. McNabb. "Parallel Particle Swarm Optimization and Large Swarms". Brigham Young University, 2011.

[17] A. Firdus, B. A. Fadzil. "Analysis Of Evolutionary Computing Performance Via Map Reduce Parallel Processing Architecture", 2014.

[18] I. Aljarah, and S. A. Ludwig. "Parallel Particle Swarm Optimization Clustering Algorithm Based On Mapreduce Methodology". 2012 Fourth World Congress on Nature and Biologically Inspired Computing (NaBIC), p104 – 111, 2012.

[19] Y. Zhang, S. Wang, and G. Ji. "A Comprehensive Survey On Particle Swarm Optimization Algorithm And Its Applications". Mathematical Problems in Engineering 8.5, p 1-38, 2015.

[20] S. Daggubati, "Comparison of Particle Swarm Optimization Variants", p. 45, 2012.

[21] Q. Bai, "Analysis of Particle Swarm Optimization Algorithm", Computer and Information Science, vol. 3, no. 1, p. 5, 2010.

[22] D. Sedighizadeh, and E. Masehian. "Particle Swarm Optimization methods, Taxonomy and Applications". International Journal of Computer Theory and Engineering 1.4, 486-502, 2009.

[23] J. Kennedy, “Stereotyping: Improving particle swarm performance with cluster analysis”, Proc. IEEE Int. Conf. Evolutionary Computation, vol. 2, p. 303–308, 2000.

[24] A. McNabb. "Parallel Particle Swarm Optimization and Large Swarms". Brigham Young University, 2011.

[25] R. Shonkwiler "Parallel Genetic Algorithms", Georgia Institute of Technology Atlanta, p.1-7, 2016.

[26] A. Verma "Scaling Simple, Compact And Extended Compact Genetic Algorithms Using Mapreduce", University of Illinois at Urbana-Champaign, p.6-14, 2010.

[27] D. Kečo and A. Subasi, "Parallelization of genetic algorithms using Hadoop Map/Reduce", Southeast Europe Journal of Soft Computing, vol.1, p.1-5, 2012.

[28] M. Pant, R. Thangaraj and A. Abraham, "Particle Swarm Optimization: Performance Tuning and Empirical Analysis". [Online]. Available: http://www.softcomputing.net/pso-foci-chapter.pdf.