University of Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization: Computer Science	Spring semester, 2014 Open / Restricted access
Writer: Long Cui	………………………………………… (Writer's signature)
Faculty supervisor: Rui Paulo Maximo Pereira Mateus Esteves Chunming Rong Tomasz Wiktor Wlodarczyk	
Titel of thesis: **Parallel PSO in Spark**	
Credits (ECTS): 30	
Key words: PSO Spark Parallel RDD	Pages: ………………… + enclosure: ………… Stavanger, ……………….. Date/year

Parallel PSO in Spark

Long Cui

Faculty of Science and Technology

University of Stavanger

July 2014

Abstract

It is now commonly realized that the energy consumption in our world is high but the energy efficiency is comparatively low. Therefore scientists and engineers are trying to investigate and develop new methodologies and applications which could lead to improve the total energy efficiency ratio to ease the problem. Moreover, the quantity and diversity of data required to solve energy optimization problems is increasing dramatically every year. Therefore it is urgent to adapt and create new algorithms that can effectively and efficiently solve Big Data energetic optimization problems in useful time. Particle Swarm Optimization (PSO) is an algorithm that can efficiently be used to solve energy optimization problems. However, most of today's energy optimization problems requires huge computational power and an efficient distributed implementation of the PSO.

This thesis focuses in contributing to the above mentioned problem of efficiently distributing the PSO using a new technology named Spark. This thesis describes how to adapt the classic Particle Swarm Optimization algorithm to the distributed Big Data platform Spark.

The solution of the problem is to firstly define the Spark data structure (Resilient Distributed Dataset) for our PSO algorithm and then create the initialization algorithm for parallel PSO. The main processing algorithm consists of Foreach Operation design and Collect Operation design.

We implemented our algorithm using Java and tested it with a real world use case of energy optimization for buildings. The use case is part of the EU FP7 research project named SEEDS[1] (Self-Learning Energy Efficient Buildings and Open Spaces) that has the participation of the University of Stavanger. The experiments show that both Spark and Hadoop could carry out big data calculation which normal serial PSO could not handle. Even given enough memory, serial PSO could take days or months to calculate. We found out that Spark could compute 32 times faster then the well known platform Hadoop and that this difference is even bigger when the number of iterations of PSO is more than 19.

[1] www.**seeds-fp7**.com/

Acknowledgements

I would like to give my gratitude to my Prof. Chunming Rong Prof. Rui Esteves and Prof. Tomasz (his last name). They really give me valuable advices and great suggestions. My greatest thanks go to Prof. Rui Esteves, for his great insight towards the scientific trend and solid foundation of scientific exploration methodology. He was always ready to help me and provided me with a lot of great information.

I would like to extend my gratitude to all the PhD professors, they really offer me a lot of help especially with the experiment data cluster setting up.

Last but not least, I would like to give my sincerely thanks to my family and all my friends. Thanks for giving support to me to fulfill my master thesis.

Long Cui

University of Stavanger

Preface

This thesis is submitted in partial fulfillment to the requirement of computer master program at the Department of Electrical and Computer Engineer at the University of Stavanger(UiS), Stavanger Norway.

The master thesis has been finished under the supervision of Prof. Chunming Rong, Dr. Rui Esteves and Dr Tomasz (his last name). The master thesis starts from January to June 2014.

Content

1 Introduction

It is now commonly realized that the energy consumption in our world is high but the energy efficiency is comparatively low. Therefore scientists and engineers are trying to investigate and develop new methodologies and applications which could lead to improve the total energy efficiency ratio to ease the problem.

Nowadays, these applications have already been developed and tested in small buildings and small areas and it shows that these applications could dramatically help to reduce the energy consumption and increase the use of energy efficiency. However theses application are suitable for small energy environment but not large environment like several buildings or even a city consisting of hundred of thousand of buildings. If we use these applications directly into the environment, we could suffer from the slow calculation of the large calculation units. Therefore an effect methods that can solve energy optimization problems in useful time are necessary.

1.1 Structure

The first part of the thesis introduces the theoretical background that we found relevant for the scope of this thesis.

The second part presents the Particle Swarm Optimization (PSO). It describes the classic Particle Swarm Optimization algorithm and the Discrete Particle Swarm Optimization algorithm and related work found on parallelization of the PSO. Then it illustrates the Map Reduce Framework and Hadoop platform to lay the foundation for the third part of our thesis.

The third part describes the model design and methodology implemented. It describes the data structure and the basic working environment. Then it shows in detail how to design and implement the particle swarm optimization in Spark.

The last part is the experiment part, where we used the use case from the SEEDS project to prove the effectiveness of the algorithm implemented in Spark. We then present the conclusions, references and necessary annexes. The developed code as well as configuration files necessary to setup the platform are inside of a zip file that was submitted with the master thesis.

2 Theoretical background

2.1 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) was first described by James Kennedy and Russell C. Eberhart in 1995[31], they propose this algorithm based on mainly two concepts:

1. The swarm intelligence observed by swarm habits and some other kinds of animal like birds and fish.

2. The evolution computation theory.

The Particle Swan Optimization (PSO) algorithm can search very large spaces of candidate solutions by iteratively moving all the particles towards the global best solutions without traversal all the possible candidates. The idea of this algorithm originally came from the social behaviors of birds flock or fish school. When one bird detects smell of food, he will broadcast the news to others, therefore others will join him in searching for the source of food which adds more probability of finding the right food. In the algorithm, we simulate the birds flock with several of particles, each particle has positions and velocity to simulate the moving of birds flock. During each iteration of the algorithm, it updates its best positions which symbolize the source of the food. After several iterations, there is a high probability that the algorithm has reached the best result. Because of such characteristics of PSO, the optimization function could perform in a faster way than looping through all the possible candidates and find one.

Particle Swarm Optimization (PSO) is used to investigate the search space of some optimization function and test the function coefficients and parameters in order to maximize the optimization function values. The algorithm has been applied to a wide range and has been used in areas as for example: chemistry, physics and economics. PSO can both maximize and minimize a function, in fact, if one set PSO to maximize some function (f) and later on want to minimize some others,

Mathematically, the maximization of a function is defined:

Given $f : R^n \to R$

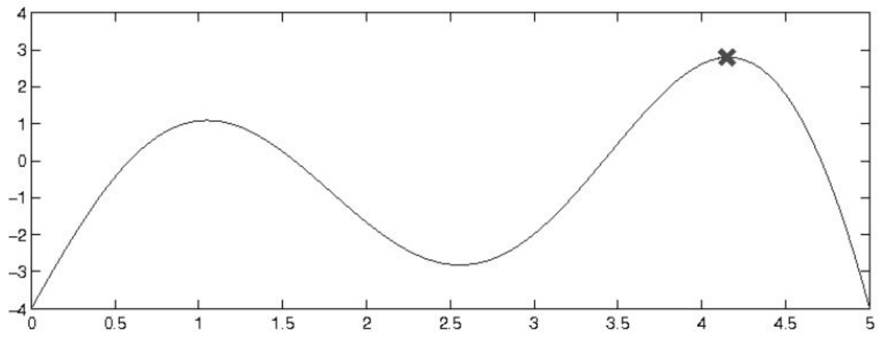
Find $\hat{x} \in R^n$ *therefore* $f(\hat{x}) \geq f(x), \forall x \in R^n$

The same method applies to the minimization of a function:

Given $f : R^n \to R$

Find $\hat{x} \in R^n$ *therefore* $f(\hat{x}) \leq f(x), \forall x \in R^n$

In the given condition, R^n stands for the search space or parameter space of the function, n is the dimension of the search space. Each element inside R^n is a candidate solution for the optimization problem. Function *f* is called objective function which maps all candidate solutions to the function space, function space is only one dimensional. Then functional space could also maps to the fitness space.

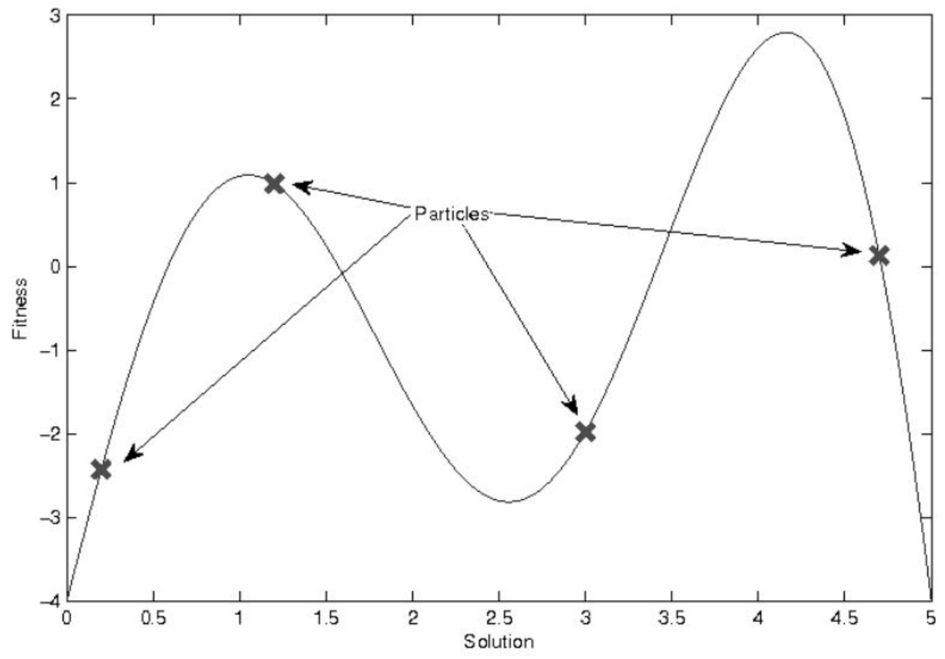


The above picture[5] demonstrates the PSO function, the x-axis represents the search space which however in this case it just shows part of the search space which is from 0 to 5. The curve seen from the diagram illustrates the map between the search space and the vertical value, the vertical value means the function space which in our case, we just draw between 0 to 4. The curve which is the objective function maps the one dimension candidate solutions in x axis to the one dimension function space in y axis. This picture demonstrates what is called fitness landscape of a simple PSO problem.

In theory, for a given function f, we could use calculus method to get the differentiable and derivative in order to get the coefficients and parameters for maximizing or minimizing the formula. However in practice, there might not exist such standard function formula, all we know is that we give an input and we could calculate and get the output, therefore we could treat it as a " black box" which we have to test every input cases in order to maximize the solution. Instead of brute force searching all the possibilities, PSO offers a way to better search the candidates with high accuracy and less time consumption.

2.1.1 Particle Swarm Optimization (PSO) Algorithm

The PSO algorithm works by assigning several particles to move in the search space and update particle parameters during each iteration. Each particle contains the information of its positions in the search space and velocities about its moving speed which indicates its next position in the search space. Each particle also has the information about its best fitness value about the objective function and the global best fitness value based on the entire particle's information.



The above picture[5] shows the initial particle positions in the PSO landscape. Note that from the picture, it can be seen easily that for particle A, it should move towards the right of the x axis. but we should note for PSO, the direct formula form of the objective function is in most cases unknown to PSO, PSO could only get the objective value by supplying the input candidate solution and calculate the result. Therefore the above curve diagram is actually unseen by PSO.

The steps of PSO algorithm is listed below:

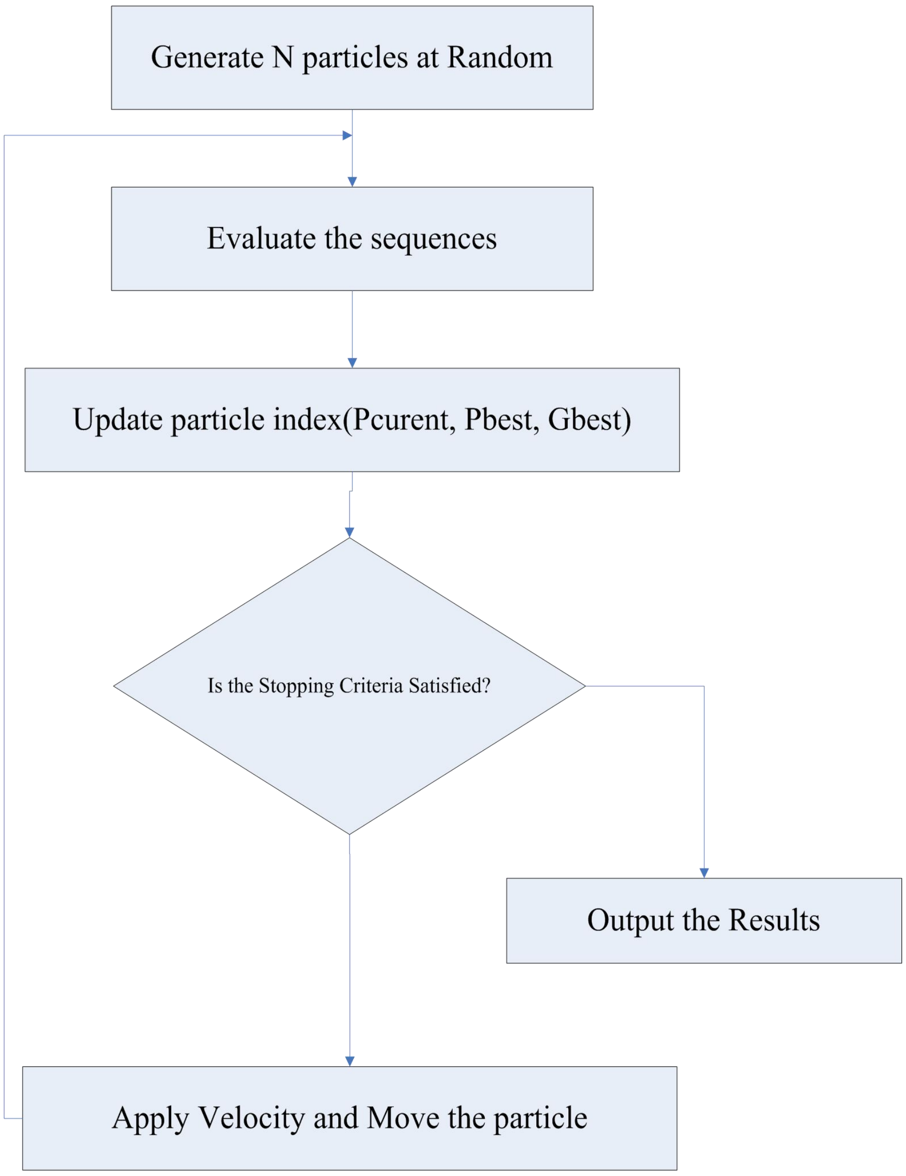
1. Calculate each particle's fitness value based on its positions and the objective function.
2. Update global best fitness value among all the particles and notice each particle and update responding parameters inside each particle.
3. Update each particle's velocities and positions based on PSO velocity equation and position equation.

Note that the potential bottleneck within serial PSO is in the step 1. For the serial implementation of PSO, the calculation of each particle is done serially which means if the calculation of one particle takes a bit more time, the whole step 1 might take incredible long time to process if given a large quantity of particles. For step 2 and 3, the only needed calculation is finding the best qualified value within all the candidate solutions. The time it takes is O(n).

$$v_i(t+1) = wv_i(t) + c_1 r_1[\hat{x}_i(t) - x_i(t)] + c_2 r_2[g(t) - x_i(t)]$$

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

The above two formulas are velocity formula and position formula. i stands for the index of the particle and t stands for the index of the iteration index. The parameter w, c1 and c2 and user setting coefficients. w is the inertial component meaning the impact of the last velocity if it is bigger, it will probably follow keep its original direction it leads, c1 and c2 are the impact the local best position and the global best position.


Generate N particles at Random
Evaluate the sequences
Update particle index(Pcurent, Pbest, Gbest)
Is the Stopping Criteria Satisfied?
Output the Results
Apply Velocity and Move the particle

2.1.2 Discrete Particle Swarm Optimization (DPSO)

The first Discrete Particle Swarm Optimization (DPSO) was proposed by Kennedy and Eberhart, 1997[32], their aim was to optimize a binary problem for a stochastic velocity problem, by modifying the position updating formula, he proved that DPSO could optimize objective function.

Experiments and theories test show that the implementation of DPSO is nearly the same as PSO but need to modify each update position based on its discrete constraint by using mathematically round function or case specific formula, but the main steps are exactly the same as ordinary PSO.

DPSO is broadly used in all areas, from accounting to scientific experiment optimization. What is more, discrete variable is a better way to get the first step to some unknown problems. An example of its usage is applied to solve the well known sales men traveling problem.

$$x_i(t+1) = F(x_i(t) + v_i(t+1))$$

In DPSO, For the function F, it is used to get the discrete value of the estimated position value based on the user defined function F. F could either be the normal round function to the nearest integer number or specially a function which is to fit the variables into the user input definition area. Therefore the next input for the particle swarm optimization follows the discrete rule of the optimization function.

2.1.3 Related work with parallelization of PSO

Due to the great convenience offered by Particle Swarm Optimization (PSO) applied to energy optimization problems, several studies[1][2] have been carried out. The studies either try to adapt PSO to more specific areas or try to work on the limitations of PSO. One of the limitation is the classical PSO is that it s implemented in serialized way. When dealing with large datasets, one particle evolvement may take minutes or each hours to compute. In this situation, serial PSO could take days in order to have all its particles computed. Therefore scientists and engineers have worked towards parallel PSO. Currently there are mainly two ways of parallel PSO.

One of the proposals is to directly use the mapReduce platform to parallel PSO. For normal parallel programming, there might be a wide range of problem between different threads or computers. Inefficient communication between different threads could easily overload that process or even make it fail to operate, what is more, constant hardware problem may cause the aggregated system to easily fail. For example, suppose a thread in a node fails twice a year and our system consists of 128 nodes, therefore the probability that the system fails during one day is $1-(1-\frac{1}{2*365})^{128}=0.16$, it means nearly with probability of 16%, the system could go down, if we upgrade our system to 512 node, the probability could go to $1-(1-\frac{1}{2*365})^{512}=0.50$ which means with probability of 50%, the system could fail.

In the paper parallel PSO using MapReduce, Andrew adopt the Hadoop implementation platform to perform the parallel PSO and the result shows that because of the advance mechanism of Hadoop, the balance of work and node failure is well maintained. Another way of parallelizing PSO is to cluster the datasets and process clusters parallel in order to fast arrive the best result. In this way, we group particles into different clusters by their different velocities. To reach the high quality of clustering, the similarity measurement inside a cluster should be maximal and the similarity measurement between clusters should be minimal. In the centroid based method, the centroid[2] is calculated and updated by the velocities of particles. Besides the calculation of the centroid, it is also necessary to calculate fitness function which measure the average distance between each particle and the centroid. The formula is:

$$Fitness = \frac{\sum_{j=1}^{k} \frac{\sum_{i=1}^{n_j} Distance(R_i, C_j)}{n_j}}{k}$$

The cluster algorithm has mainly three steps:

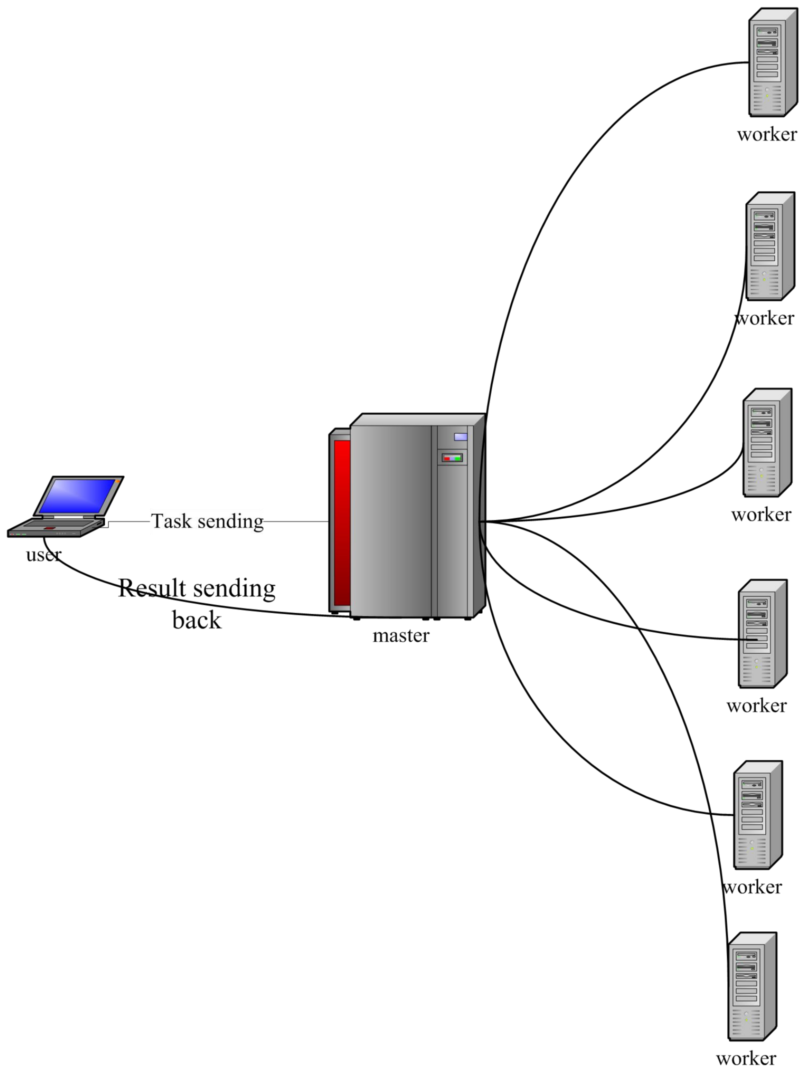
2. Update the particles centroid using mapReduce. It calculates all the new positions for all nodes and then apply the centroid formula to get the result.
3. Update the fitness value with the new centroid. Based on the fitness function, we update its value and the new centroid.
4. Update the best global centroid and best personal centroid. After the calculation of the fitness function value, we could update the centroid value.

For the implementation of mapReduce platform, there are Google's mapReduce[3], Apache Hadoop mapReduce and Disco[4]. All mentioned platforms have fault-tolerant and can maintain load balance.

2.2 Map Reduce Framework

MapReduce is a framework which originated from the marshalling distributed server system, It is used to run parallel programs simultaneously, managing the communication and data transfer between different processes and different nodes, also it can provide redundancy for the system and make the system fault tolerant[33].

In May, 2012 , a new version (2.0) of the MapReduce framework was released. The new version of the distributed system could still have the functionality of tolerating node failure and introducing the redundant mechanism to improve the productivity. But what the new framework provide is the scalability of running all mappers at the same time on different servers and therefore apply parallel distributing calculation. In the new framework each mapper is responsible for filtering and calculation its part of data and map it into the form of key and value in order to let the reducer to work afterwards. The reducer is in the charge of gathering all the result coming from all the mappers and summarize these results in the form that user require.



The dataflow of mapReduce is defined as several phases:

1. Input reader phase

Input reader reads data from a distributed file system (eg, in Hadoop it is called HDFS) and split them to smaller pieces (eg, 16MB to 128MB, in HDFS it is 64MB by default), and assign each small piece to a mapper to make it to be the workspace.

2. Mapper function calculation phase

Each mapper read its data in, using the filtering and sorting function to process the data and put them into key and value format. A common example is to read once a line from its dataset and process them. One line of data might produce one key and value or multiple keys and multiple values.

3. Partition function calculation phase

The function of partition is to map the result of mappers to the responding reducer by applying some methodology, for example a common technique is to calculate the hash code value of the key and modulo the number of reducers to assign the result to the responding reducer with the right index.

4. Compare function calculation phase

   The phase gathers all the map result and compares them to assign them to the right reducer.

5. Reduce function calculation phase

Based on all the input of each reducer, each reducer tries to generate its result key and value by apply the reducer summering function.

6. output writer phase

   Output all the result of reducer to the distributed storage.

2.3 Hadoop

Hadoop computing platform was first invented by Mike Cafarella and Doug Cutting and was originally developed to support the project of distributing the Nutch search engine in Yahoo[34].

Hadoop is an open source framework for data processing and data storage on a large quantity of data. Nowadays, Hadoop framework is widely used in nearly all data processing industries and it is now one of the key projects in Apache project.

Hadoop is made up of several modules:

1. Hadoop common module.

   The common module contains mostly the basic libraries and basic utilities for other modules. It lays the bottom structure of Hadoop system which is on the system file system and operation system level. It contains the necessary jar files and shell scripts to set up Hadoop.

2. Hadoop distributed file system (HDFS) module

   It is a file system that Hadoop cluster is working on, the file system provide redundancy to ensure system availability. It derives from Google's File system.

2. Hadoop YARN

   It is a resource management system which is responsible for all the computing resource scheduling and assigning based on user's application.

3. Hadoop MapReduce

It is a parallel programming platform based on the MapReduce structure. Hadoop implement them using its java library and extend them to support node failure and system robustness. It derives from Google's Mapreduce system.

2.4 Spark

Spark is an open source computing framework specialize in data analytics, it builds on top of Hadoop HDFS system and is not tied to the map reduce platform, Spark offers high speed computing based on HDFS system and developers claim to be 100 times faster for some applications calculation in memory and 10 times faster calculation in disk.

Spark started as an Apache incubator project in June 2013 and became a top level project in February 2014. Up to now, more than 150 individual programmers from more than 30 companies including Yahoo, Intel etc. have contributed to the project. The newest version of Spark is the 0.9.1 which is released on April 09, 2014[35].

Spark loads the necessary data in to the cluster memory based on user's application and apply calculation directly in memory making it faster than the traditional Hadoop- HDFS approach. In opposition to Hadoop that requires loading the necessary data from the HDFS in every iteration, Spark keeps the data in memory in between iterations. Because of this mechanism, Spark is very suitable for algorithms that iterates on the data

Spark mainly consists of several characteristics that make it becoming popular:

1. Ease of use

   As the object-functional scripting and programming language, Scala has been adopted as the development language in several big projects, so is Spark, because of the high level and simplicity of Scala which offers over 80 high level operators for using, Spark could be easily build to carry our parallel computing. Spark could also be operated in common used languages such as java and Python.

2. Generality

Although Spark is built on top of Hadoop, it also support high level application. Up to now, Spark could work seamlessly with application like Shark which is a SQL database query system, Spark Streaming system for real time data processing, MLib for algorithm which requires recursively calculations such as machine learning and GraphX which is mainly deal with graph design.

3. Integrated with Hadoop

As Spark is built on top of Hadoop, there could be definitely seamless integration between Spark and Hadoop. Spark could work on Hadoop 2's Yarn and could also read data from the existing HDFS or even HBase Cassandra systems.

4. Lighting fast Speed

Compared with the Hadoop mapreduce calculation operation. Spark could perform the calculation up to 100 times faster for some applications calculation in memory and 10 times faster calculation in disk.

3 Model and Methodology

3.1 Hypothesis

With the coming of MapReduce methodology and some implementation like Hadoop, it is much easier and efficient to process large scale of data in a faster way, because of its data intensive and fault tolerant properties; it has received great success and has been used widely. However as said earlier MapReduce platform focus on more acyclic data flow which is suitable for data filtering or simple data mining but could not be used in applications that requires recursively perform MapReduce operations on the dataset. Fro example, some machine learning algorithms and some data prediction algorithms.

By adapting the newly top level apache project Spark as our working platform, we might change the way that we have to use the old MapReduce platform to perform iterative operations which is quite time consuming. With Spark, we could load the necessary data into memory and carry out the calculation purely in the distributed cluster memory by adapting the RDD(Resilient Distributed Dataset) dataset type. Because of the speed of memory reading and writing is much faster than that in disk. Therefore Spark is suitable for carry out recursive operations on some big datasets like some machine learning algorithms or interactive data mining tools.

Since PSO is an iterative algorithm trying to optimize the particle positions. Therefore Spark might be a better platform to implement parallel PSO and optimize the particles. As Spark is open source, it would offer lots of convenience when implementing the parallel PSO and could be later on adapted to other versions of PSO like discrete PSO which could solve the problem when expanding the Optimization in SEEDS project.

Within Spark, we could persist the particle information all in the memory and update them at the end of each iteration. The particle information consists of the particle position, particle velocity and local best and global best position. If the dimension of each particle is very large, then each particle data might be quite large, therefore we should well arrange all the particles numbers inside each mapper to maintain the memory overflow. We will discuss in detail about the procedure of particle assignment.

H0: Apache Spark platform can be used to speed up optimization problem based on PSO.

To test *H0* we will apply our new Spark based PSO implementation to the real world use case SEEDS project.

3.2 **Resilient Distributed Dataset (RDD) Representation Design**

In Spark, one of the most important abstractions is the Resilient Distributed Dataset (RDD) which maintains the in-memory calculation. It represents the collection of objects in the partitions across the whole nodes. RDD is a read-only collection and has fault tolerant characteristic. If one partition in a node fails, it will reconstruct this partition from other RDD since other RDDs have enough information and could rebuild the broken partition.

From the user's view, the programmers of Spark should construct its own driver program to lead the Spark process from the high level control and start its parallel programming operation.

Based on this data structure, we will create a RDD and suit all particles variable into the resilient distributed datasets.

3.2.1 Construct RDD for parallel PSO

Spark offers us fours ways in order to fit into the resilient distributed data (RDD). Each of the RDD dataset could be represented by a Scala object. The four ways are:

1. Reading from a distributed file. Assign the path to the constructor of a specific RDD type. The RDD could analyze and convert it to RDD automatically, and then it returns a handle for user to operate.
2. Using Parallelizing mechanism. We should pass a collection of Scala to the driver program, for example an array and RDD could assign them to each node.
3. Transforming RDD. If we have already generated an RDD and want to have another instance. We could treat it like the normal programming variable and assign it to other variables.

4. Changing persistency. Normally the RDD dataset type is ephemeral and is discarded after use. Therefore we need to create a new RDD in this case, however we could cache the RDD if we are going to reuse it.

For the parallel PSO computing, we will both use the method 1 and method 4 for construction of new RDD and reuse of the RDD datasets in order to optimize PSO.

The creation of the new RDD for PSO is better if we create our specific PSO particle pattern and save it in a distributed database and the RDD constructor will read directly from the database and persistent it in the memory for later computation.

For each particle, the most important information should be position, velocity. Therefore In this master thesis the variables are arranged in a file called initial data. The initial data pattern is in the form of:

Local				Global				Local	Global
Positions	Velocities	Best Positions	Best Velocities	Positions	Velocities	Best Positions	Best Velocities	Best fitness	

For each particle, it has two domains of variables, one of called local which is all the information about itself, the other is called global which keeps all the best variable values for updating its velocities.

The local variable has positions which are the place of the particles, it is the input parameter for the PSO optimizing function. All the other variables are trying to help the particle to find a better position which lead to the optimized PSO function.

The local velocities keep track of how faster the particle move towards the one direction based on its existing position. Basically it is a random value within a region which is selected based on the velocity formula.

The local best fitness is the value of the PSO function value after applying the input parameters. It is an indication for the tendency towards the global best positions.

For each variable in local there is a responding variable in global variable. The global variable keeps track of the best particles's information in the whole PSO. It is updated during the end of each iteration. The local variables take the global variable into consideration and update its velocity accordingly.

We could set the dimension of positions to be the number of variable that we want to optimize and it could be illustrated through the graph below.

Positions			
First Dimension	Second Dimension	third Dimension	…

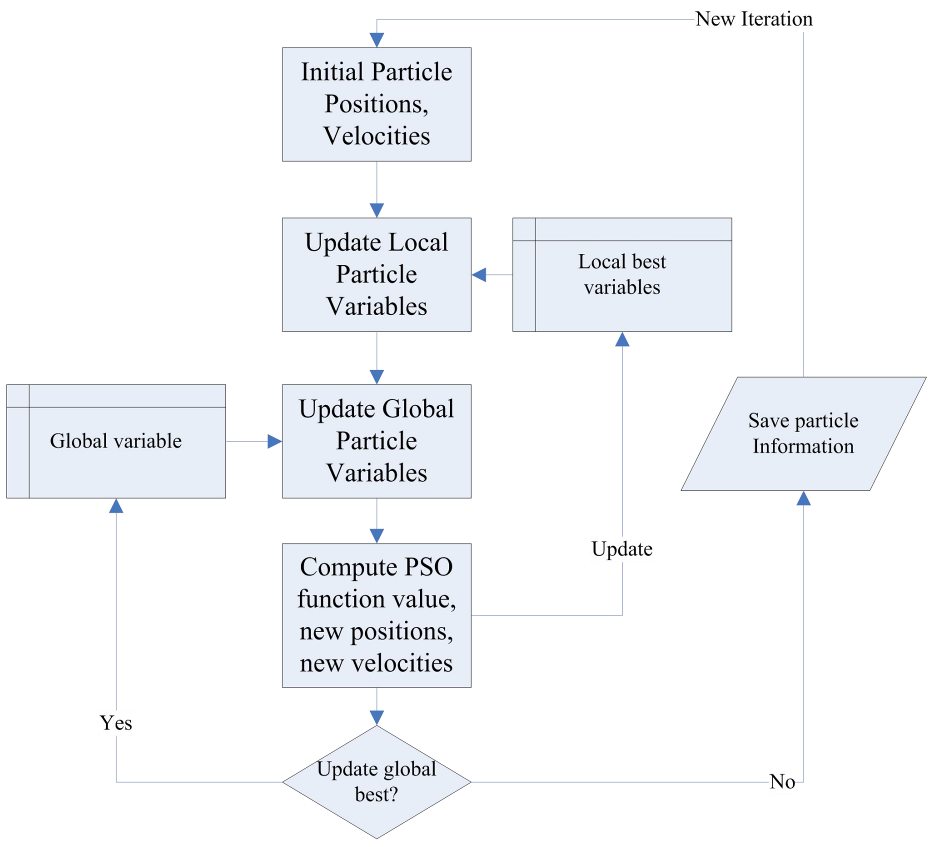
The format of data is that for each variable inside the initial file, it is separated by a space symbol. The Spark has the information about the dimension of the particle and therefore there is no need to create several levels of separation for the initial file.

3.2.2 Parallel PSO initialization algorithm

Before running parallel PSO algorithm, we need to set the initial information about the particles, for both its local variables and global variables. This process is usually referred as initialization of the particles. The reason for a careful initializing is to start the PSO with particles closer to the global optimum. This way the PSO can provide a more accurate and faster convergence.

We could also choose random algorithm to randomly set the initial values for all the particles but they might take a long time to evolve and might also fall into local best situation. Therefore it is better to create specific algorithm for every case in parallel PSO to get better result.

The graph shows how to create a specific algorithm for the initializing of the parallel PSO algorithm.


New Iteration
Initial Particle Positions, Velocities
Update Local Particle Variables
Local best variables
Global variable
Update Global Particle Variables
Save particle Information
Update
Compute PSO function value, new positions, new velocities
Yes
Update global best?
No

As the graph shown above, for each new particle, we need to specify all its variables. For most of the particles, we could randomly select a value within the search boundary for its positions and velocities. But for some of the particles, we could give them specific initial positions to better converge. Therefore the Initial Particle step could be implemented in two ways.

The steps for the first method are listed as below:

1. Find the dimension of the positions and each boundary for each dimension.
2. Adapt random algorithm to generate random value and get the random value within each boundary.
3. Assign the value to the responding position.

The steps for the second method are nearly the same as the first one except the step two which in the second method it is not to generate random values but to get the most likely value from some algorithm or user's experience and then assign them to each particle.

It is the same way to treat the initialization of velocity. If the position is a discrete value, we could use math function to round the random value to a discrete value. If the search area does not contain all the discrete value within the boundary, then we should create an algorithm to convert the real value to the discrete value. As all the random values are created with the same probability, therefore the basic idea for this algorithm is to find the shortest qualified discrete value for the random real value.

After setting up the initial particle positions and velocities, we need to set the local best positions and local best fitness value which is the objective function value. All these variable values are fetched from the local best variables which are updated during each iteration after the computing of the function value result, it will update the local best variable if it is better than the old result defining by the user function. The local best variables keep track of the best local fitness value and the responding positions. These values and positions are the input parameters for the velocities which lead the particle towards the local best positions.

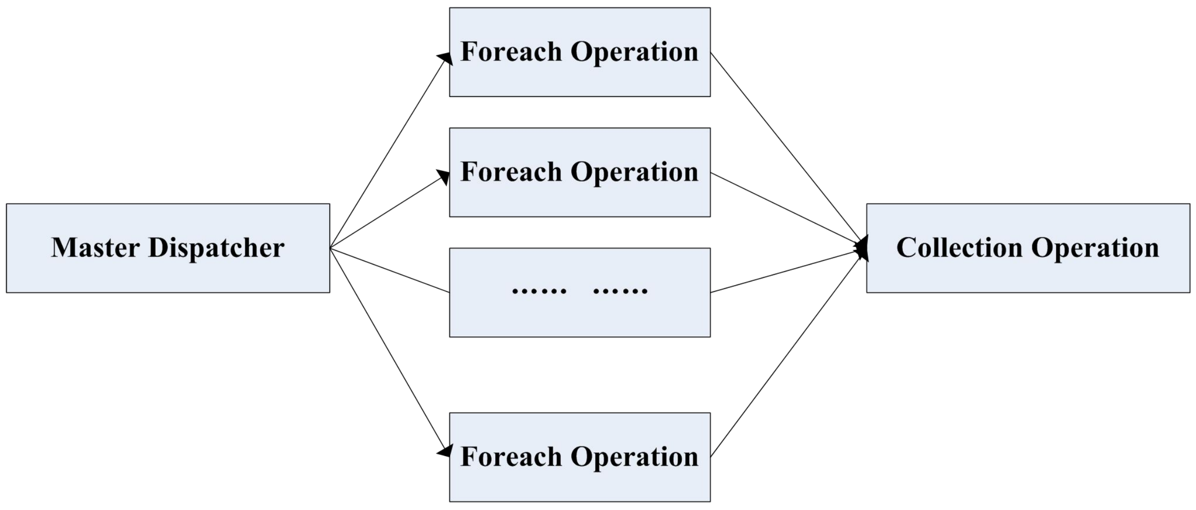
The next step is to update the global particle variables. The global variable contains the global best positions and global best fitness values. Theses values are fetched by comparing with all other particle's best positions and best fitness values. The updating method is the same as the local best variable updating.

After all these 3 steps, all the variables are set up. Then we need to compute the new fitness value together with the new positions and the new velocities. The formulas are listed in the introduction part of the PSO.

Up to now, the initialization of a particle is finished, depending on this particle's position, we will decide to update the global best or not. At last, we save the particle information and ready to start the next iteration to initialize the next particle.

3.3 Parallel Programming Design

The parallel design mainly consists of two parts, the parallel foreach operation which is the particle evolvement and the collection operation which is the summary of all the information across the parallel cluster. As the chart shown below that the master dispatcher dispatch assignment across all the node to carry out foreach operation and then the collection operation collects all the results of foreach and summarize it, at last it returns the summarized data to user.



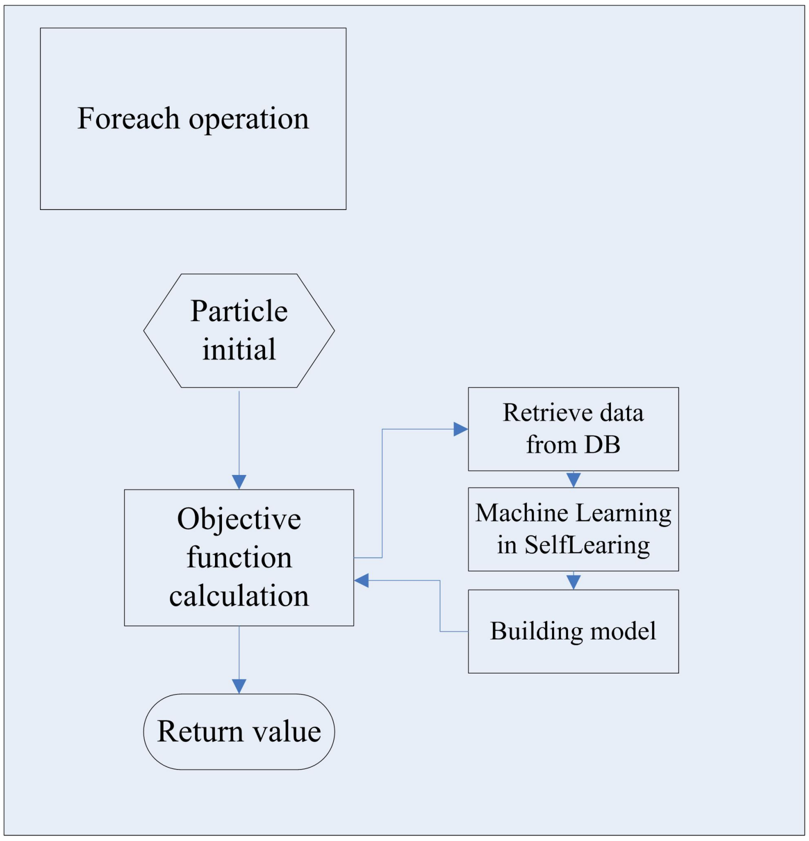
In Spark, we need to write the top level driver program and assign them to all the nodes. Each node will process the foreach Operation part. In our case it deals with the evolvement of each particle, the most important is to calculate the objective function. Because the calculation might take a long time depending on the calculation method. For example, in our case, the PSO calls the selflearning module which is a machine learning algorithm, in this algorithm, it not only need to carry out calculation but also perform the database operation. Therefore the whole process might consume lots of time. If we parallel this part, the time will be just the longest PSO particle calculation time instead of summing up the whole particles time.

3.3.1 Foreach Operation Design

The foreach operation is carried out in each node in the cluster, first it define the user data type, for example, double, String or int. then it read in the user data type as the foreach operation input. In this sense, it is pretty much the same like the mapper in MapReduce platform which define the key and value. However in Spark, one can define several inputs at the same time. For example:

PairFunction<String, Double, String>

The function reads a String type in, run the user defined program, calculate the objected function value and return them in the data type of Double and String.



We first extend the PairFunction class called PsoMrMapper and override our own call method with input of String and output of Tuple2<Double, String>. And define our own field variables for the PsoMrMapper class and then construct the call method for each node.

The detail steps are listed below:

1. It reads in all the parameters in as a String and later parses the content of the String. The benefits of doing so is we could adapt different kinds of values and different numbers in to the input parameter, therefore we create a universal adaptor for the foreach operation. the other benefit is the input variable number is a lot, therefore if we set one variable one input parameter. The input parameter would be quite large and uneasy to analysis. The input String is in the format of {position[0...n], velocity[0...n], localBestPosition[0...n], globalBestPosition[0...n], localBestFitnessresult, GlobalBestFitnessResult}.
2. After calculating and performing all the machine learning algorithms and related operations. The foreach operation returns a pair of two values with data type of Double and String. The double value is the particle best fitness value and the string value is the all the particle information after its evolvement.

3.3.2 Collect Operation Design

As the operation name suggests, the operation collect all the elements across all the node and gather all the data and save them as RDD and return it in a form of an array to user. Each for element is a member of the array and has the same return type, in our case, it is the Tuple2<Double, String>. The size of the array is the same as the number of foreach nodes.

3.4 **Foreach and Collect in Context**

The algorithm is designed to implement parallel PSO based on mapReduce approach. The detail is as below:

Parameter declaration:

1. Particle number. It indicates how many particle are going to be paralleling run.

2. Particle dimension and variable dimension boundary. Particle dimension is the number of optimization variable number. Variable Dimension boundary consists of position dimension boundary and velocity dimension boundary. For position in each dimension, it is the variable running scope that PSO can manipulate. We get it from the input domain of optimization formula. For velocity in each dimension, we set it to be half length between maximal and minimal position in responding domain.

$$v[i] = 0.5 * (p_{\max}[i] - p_{\min}[i])$$

3. Number of iterations. Define how many times PSO are going to evolve.

4. Fitness function. Optimization formula, the goal of PSO is to search the minimal or maximal of fitness function result.

Algorithm Steps:

Step 1: Initialization of particles.

Step 1.1: Randomly set the position of all the particles within dimension boundary. Set the particle moving velocity according to the formula(1). Set the local best position to be its current randomly picked position.

Step 1.2: based on the positions(fitness function input), calculate the fitness result f[i] for all particles. Updating global best fitness function result and fitness position.

Step 1.3: add global best fitness function result and fitness position to each particle.

Step 1.4: save each particle to an output file in the format of {position[0...n], velocity[0...n], localBestPosition[0...n], globalBestPosition[0...n], localBestFitnessresult, GlobalBestFitnessResult}.

Step 2: Evolvement of particles.

Step 2.1: Mapper: each mapper is responsible for one particle, it read the particle information from the output file and update the particle status information according to formula:

$$v^{t+1}[i] = w \cdot v^{t}[i] + c_1 \cdot \varphi_1 \cdot (l^{t}[i] - p^{t}[i]) + c_2 \cdot \varphi_2 \cdot (g^{t}[i] - p^{t}[i])$$

$$p^{t+1}[i] = p^{t}[i] + v^{t+1}[i]$$

Step 2.1: Mapper: each mapper calculates the fitness function result based on the new position which is the new input variable for the function.

Step 2.2: Mapper: update the local best and responding local best position if it is better than before.

Step 2.3: Mapper: each mapper sends out its local best result as value and position as key to reducer.

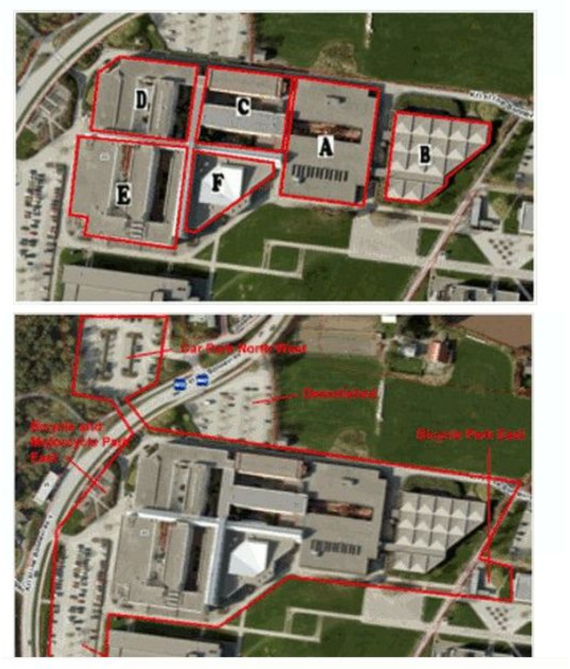
Step 2.4: Reducer get the smallest key and assign the responding value(the position) to the global best position place in all the particles.

Step2.5: update all the particles information in the output file.

4 Experiment and analysis

4.1 Experiment background

As the introduction mentioned, we are trying to implement a new fast algorithm inside the new Spark platform to solve the optimization scalable issues in the SEEDS project.



The SEEDS project consists of three main working modules and its accessory modules. The three main modules are: (1). Building module. (2). Self-learning module. (3). Optimization module. The accessory modules are (1). Graphical user interface module. (2). WISAN communication server module together with database.

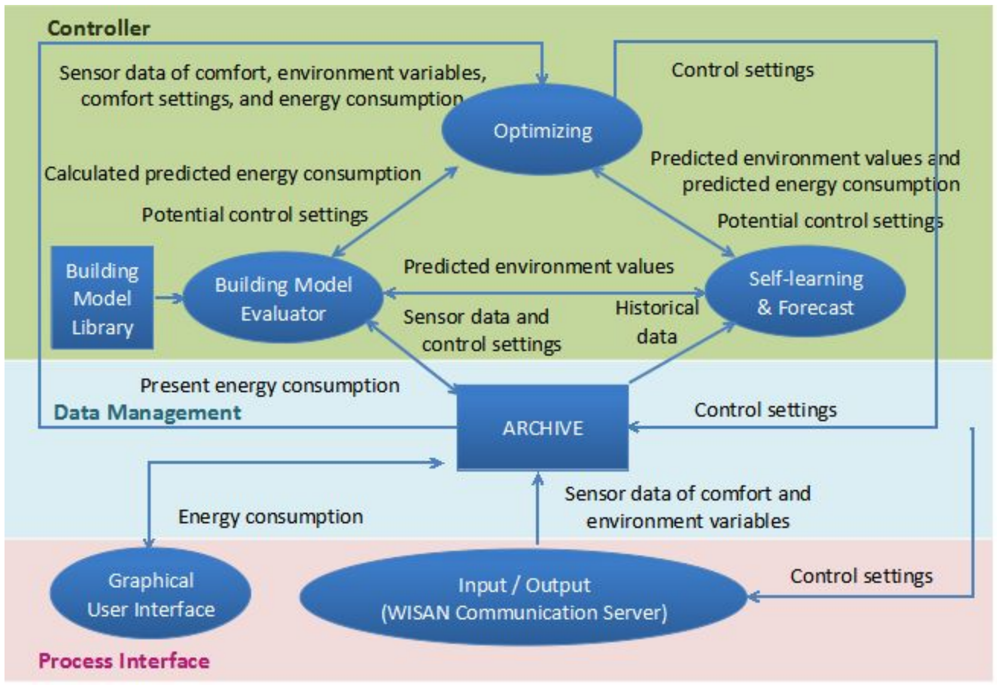
The Building module models the whole building parameters such as the energy consumptions and the controller setting and comfort level. The building module consists of a building module library which shows the characteristic and parameter of each sensor and the relationship between each sensor.

The self learning model is based on machine learning techniques, after training with the previous sensor control setting data together with the comfort setting point and energy consumption;

it could predict the sensor settings for future time point based on the minimization of total energy consumptions and meeting the comfort user level criteria.

The optimization function (OPT) works together with self learning. Its usage is to quickly find the most suitable potential control settings with minimal total energy consumption under the requirement of meeting the user comfort setting level. The original optimization plan was to loop through all the possible solutions calculated by self learning. For small amount of data, it is quite easy and affordable to loop through and check every possibility. However as the project grows bigger and more sensors are being put out and more energy form are to be calculated, the optimization could not return the most suitable result within the limited time. Therefore the OPT could be the bottleneck for the whole project. In SEEDS project, PSO algorithm was used to solve this problem.

The PSO algorithm could search very large spaces of candidate solutions by iteratively moving all the particles towards the global best solutions. The algorithm originally came from the social behaviors of birds flock or fish school, when one has detected smell of food, he will broadcast the news to others, therefore others will join him in searching for the source of food which adds more probability of finding the right food. In algorithm, we simulate the birds flock with lots of particles, each particle has positions and velocity to simulate the moving of birds flock. During each iteration of the algorithm, it updates its best positions which symbolize the source of the food. After several iterations, there is a high probability that the algorithm has reached the best result. Because of such characteristics of PSO, the optimization function could perform in a faster way than before.


Controller
Sensor data of comfort, environment variables, comfort settings, and energy consumption
Optimizing
Control settings
Calculated predicted energy consumption
Potential control settings
Predicted environment values and predicted energy consumption
Potential control settings
Building Model Library
Building Model Evaluator
Predicted environment values
Sensor data and control settings
Historical data
Self-learning & Forecast
Present energy consumption
Data Management
ARCHIVE
Control settings
Energy consumption
Sensor data of comfort and environment variables
Graphical User Interface
Input / Output (WISAN Communication Server)
Control settings
Process Interface

The developed BEMS could not only be suitable for simple minimization energy consumption but also for CO2 and Renewable Energy Sources (RES), heating/cooling systems and so on. With the idea of energy saving for both indoor building and open space like street and campus, the SEEDS could be suitable to easily adapt to more general environment like a whole town or a city, even a vast of areas of different kinds of buildings and streets and squares.

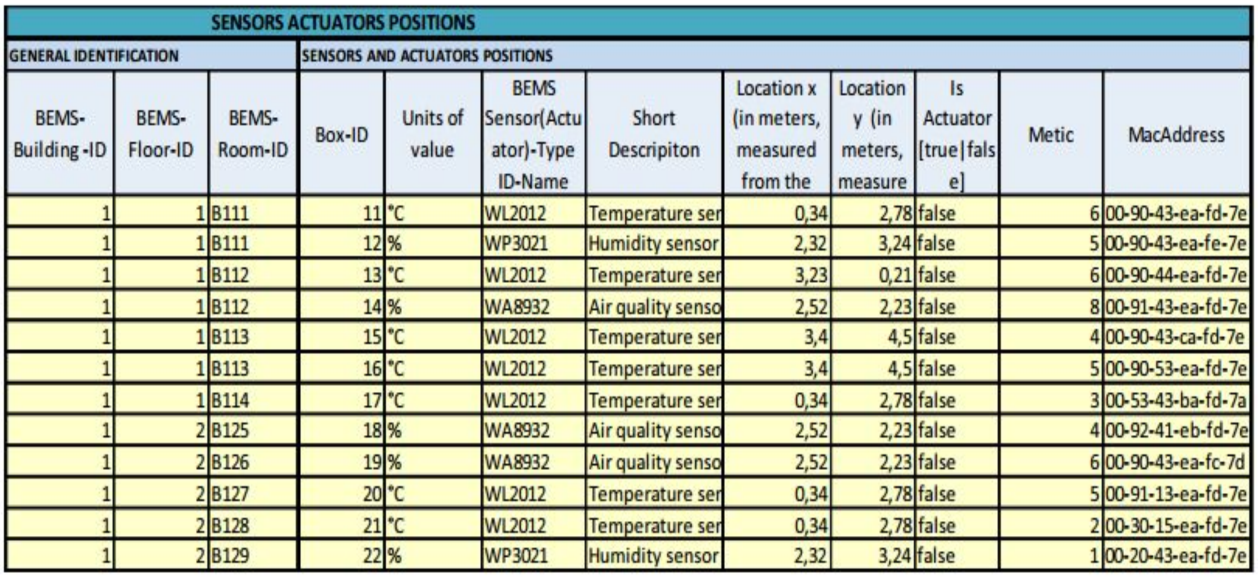
Given this idea in thought, the whole BEMS need to process more data than a small building which creates demand both for the input and output of BEMS but also for the main processing mechanism, as a developer of the optimization function; I could foresee the bottleneck when facing hundred of thousands of controlling sensors based on a simple calculation. Normally with 99% possibility of finding the most suitable sensor control settings under that criteria, the OPT need to carries out around 35 rounds of particle moving, each round need to calculate the suitable control settings for all the sensor controller for the required time steps. Experiment shows for each time step the calculation of all the sensor control settings (right now the number is 176), the time is around 0.1 second. Therefore the running time of OPT is the number of rounds (35) times number of required time steps (60) times the unit step calculation time (0.1 s) which equals to 210 seconds.

Item	Count
sensor control settings	176
Average unit processing time	0.1 second

Number of round of PSO	35
time steps	60
All time	35 * 60 * 0.1 second = 210 second

For a real time system, 210 seconds is such a long time, not to mention that if someone set up more sensors, the round for OPT should be more as well because the search dimensions is more than before (each sensor occupies one dimension). What is more, with more sensor data, the SelfLearning model could use more time to predict future data which causes the unit step calculation to be longer. Therefore when sensor data becomes larger, the bottleneck issue of OPT becomes more severe. We should investigate into OPT to upgrade its algorithm and reduce the calculation time of OPT get shorter.

The optimization module creates a set of potential control settings and pass them to selflearning to get the result of the model building with the given potential control settings. For the work of PSO inside optimization, it calls the selftlearning module inside each objective function, the selflearning then calls the build model module

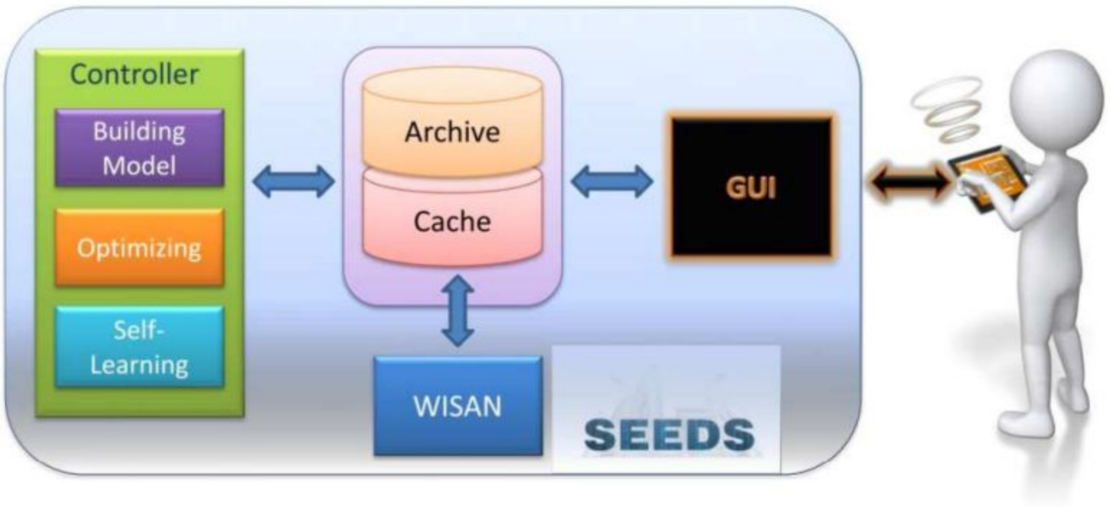


SENSORS ACTUATORS POSITIONS											
GENERAL IDENTIFICATION			SENSORS AND ACTUATORS POSITIONS								
BEMS-Building -ID	BEMS-Floor-ID	BEMS-Room-ID	Box-ID	Units of value	BEMS Sensor(Actuator)-Type ID-Name	Short Descripiton	Location x (in meters, measured from the	Location y (in meters, measure	Is Actuator [true|false]	Metic	MacAddress
1	1	B111	11	°C	WL2012	Temperature ser	0,34	2,78	false	6	00-90-43-ea-fd-7e
1	1	B111	12	%	WP3021	Humidity sensor	2,32	3,24	false	5	00-90-43-ea-fe-7e
1	1	B112	13	°C	WL2012	Temperature ser	3,23	0,21	false	6	00-90-44-ea-fd-7e
1	1	B112	14	%	WA8932	Air quality senso	2,52	2,23	false	8	00-91-43-ea-fd-7e
1	1	B113	15	°C	WL2012	Temperature ser	3,4	4,5	false	4	00-90-43-ca-fd-7e
1	1	B113	16	°C	WL2012	Temperature ser	3,4	4,5	false	5	00-90-53-ea-fd-7e
1	1	B114	17	°C	WL2012	Temperature ser	0,34	2,78	false	3	00-53-43-ba-fd-7a
1	2	B125	18	%	WA8932	Air quality senso	2,52	2,23	false	4	00-92-41-eb-fd-7e
1	2	B126	19	%	WA8932	Air quality senso	2,52	2,23	false	6	00-90-43-ea-fc-7d
1	2	B127	20	°C	WL2012	Temperature ser	0,34	2,78	false	5	00-91-13-ea-fd-7e
1	2	B128	21	°C	WL2012	Temperature ser	0,34	2,78	false	2	00-30-15-ea-fd-7e
1	2	B129	22	%	WP3021	Humidity sensor	2,32	3,24	false	1	00-20-43-ea-fd-7e

4.2 **Platform and environment setup**

It is quite necessary to test our method based on the SEEDS project and check the results. The first step with Spark and Hadoop is to set it up. As we are using it as a cluster computing algorithm, we need to first set up the cluster.

After the cluster is set up, we configure the environment for Spark and Hadoop together with all their dependencies.



To help the reader understand the use case we describe here some details of the SEEDS platform. The above picture shows the basic unit level structure. The Controller part is the brain of the whole system. The controller retrieves data from the data set system(Archive/Cache) and returns the calculation result to the dataset to be shown on GUI unit. The WISAN is the sensor unit which update the sensor data in the

4.2.1 System configuration

Based on the above description, test environment has been set up. We configured Hadoop , Hbase and OpenTSDB. The basic hardware description is as below:

Hardware	Configuration	15 nodes live in 3 racks. All racks have backup and UPS.
CPU	AMD Opteron(tm) 4180 six-core 2.6GHz	

	processor	They are connected by 1Gps switch.
RAM	16 GB DDR-2	
Hard Disk	3x3TB	
Network card	HP ProCurve 2650 100 Base Tx-FD	

Each of the 16 nodes have the same configuration of Hadoop, one of them is treated as master node(NameNode) and the others are Data node which actually perform the calculation operations. Some specific configurations of Hadoop are:

Hadoop Parameter	**configuration**
Hadoop Version	0.20.2-cdh3u6
Replication	3
HDFS Block size	128 MB
io.file.buffer.size	128 MB
io.sort.factor	100
io.sort.mb	100
mapred.map.tasks	100
mapred.tasktracker.map.tasks.maximum	6
mapred.reduce.tasks	22
mapred.tasktracker.reduce.tasks.maximum	3
mapred.child.java.opts	Xmx1024m
mapred.child.ulimit	4194304
dfs.block.size	134217728
dfs.replicaton	3
HBaseVersion	0.90.6-cdh3u6
Load	average
Zookeeper	Quorum
Zookeeper	Port

Next, we follow the tutorial to install Spark, which has friendly user interface. inexperienced users could easily find ways to retrieve the data they need. Simple by choosing the start time, end time and metrics. Appendix 1 is system configuration of Spark platform.

4.2.2 Simulation benchmark

There are mainly three ways to deploy Spark in clusters, (1). Mesos (2). YARN (3) standalone Mode. In our simulation benchmark, we will use the standalone deploying mode. Judging by the name, it might not seems to be a cluster deployment method, but it is one of the easiest way of realizing Spark cluster.

The steps for setting the simulation workbench are as below:

(1). Download the newest version of Spark and build it on all the nodes.

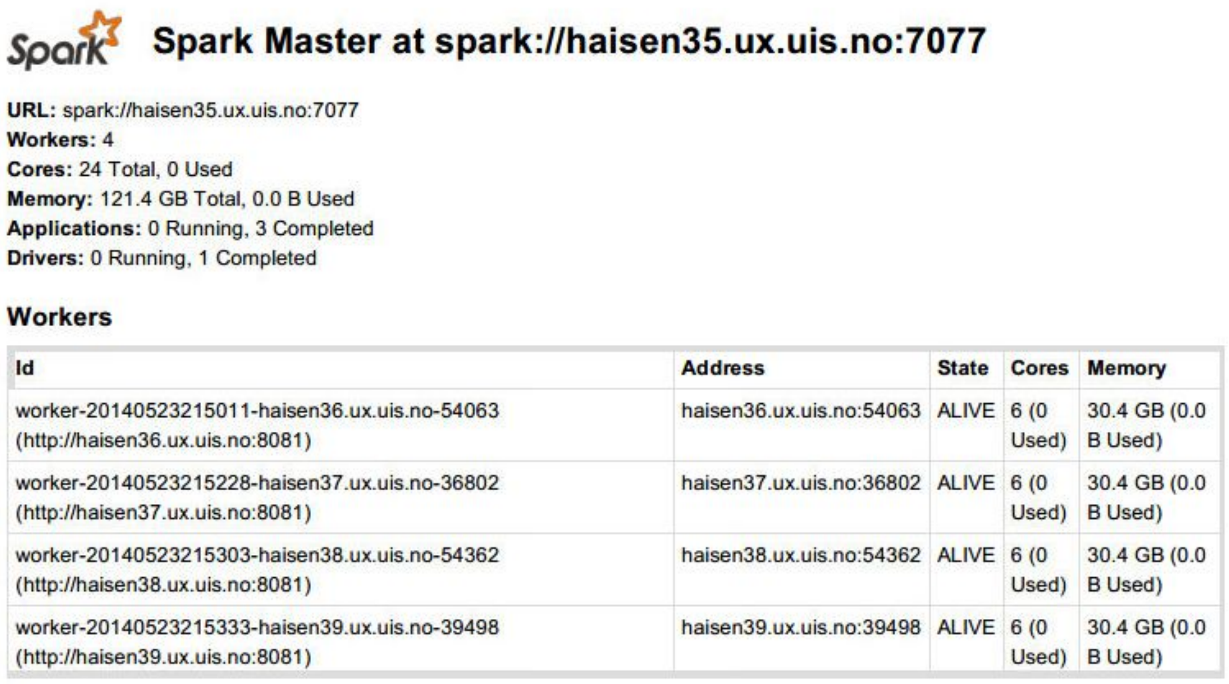
(2). On the master node, run the standalone master server starting scripts: ./sbin/start-master.sh after running the script, it will print the master server web UI like: Spark://HOST:PORT URL. In our case, we could not monitor it directly in our master node by entering localhost://8080, because it is a unix server, since the unix service internet could only be visited within it. we could monitor the server by visiting:

```
//haisen35.ux.uis.no:8080
```

(3). Next we will attach several workers to the master by running the script inside each worker you want to connect:

```
./bin/Spark-class org.apache.Spark.deploy.worker.Worker Spark://haisen35.ux.uis.no:7077
```

(4). We could see all the master and workers information in the master web UI.



Spark Master at spark://haisen35.ux.uis.no:7077

URL: spark://haisen35.ux.uis.no:7077
Workers: 4
Cores: 24 Total, 0 Used
Memory: 121.4 GB Total, 0.0 B Used
Applications: 0 Running, 3 Completed
Drivers: 0 Running, 1 Completed

Workers

Id	Address	State	Cores	Memory
worker-20140523215011-haisen36.ux.uis.no-54063 (http://haisen36.ux.uis.no:8081)	haisen36.ux.uis.no:54063	ALIVE	6 (0 Used)	30.4 GB (0.0 B Used)
worker-20140523215228-haisen37.ux.uis.no-36802 (http://haisen37.ux.uis.no:8081)	haisen37.ux.uis.no:36802	ALIVE	6 (0 Used)	30.4 GB (0.0 B Used)
worker-20140523215303-haisen38.ux.uis.no-54362 (http://haisen38.ux.uis.no:8081)	haisen38.ux.uis.no:54362	ALIVE	6 (0 Used)	30.4 GB (0.0 B Used)
worker-20140523215333-haisen39.ux.uis.no-39498 (http://haisen39.ux.uis.no:8081)	haisen39.ux.uis.no:39498	ALIVE	6 (0 Used)	30.4 GB (0.0 B Used)

(5). For launching the application inside the cluster, we basically have three ways of realizing it: 1. MASTER=Spark://localhost:8080 ./bin/Spark-shell which means running the interactive Spark shell.

2. Pass Spark://localhost:8080 to the Spark context constructor as the master and run the application by remotely connecting to the cluster master node.

3. We could also upload our application to the master server and spread among all the nodes. In this case, we have to add all the dependencies our application has and set responding environment as necessary. The script is as follow:

```
./bin/Spark-class org.apache.Spark.deploy.Client launch
   [client-options] \
   <cluster-url> <application-jar-url> <main-class> \
```

The client options allow users to specify how much memory we are going to allocate to our driver program in the unit of Megabytes. In our case it is –memory 512. We could also set the CPU core number for our specific driver to run. In our case it is –cores 5(we leave one core for other services).

In our experiment, we put our java file into the example folder, because the sbt compile tool could automatically help us build it and add all the dependencies. After compiling, it will put the all in one jar into the target folder called: Spark-examples-assembly-0.9.1.jar.

We run our application directly in the master server and let the master server send all the commands to the work nodes.

```
./bin/Spark-class org.apache.Spark.deploy.Client launch \
Spark://haisen35.ux.uis.no:7077 file:///home/haisen/long/Spark-0.9.1/examples/target/scala-2.10/Spark-examples-assembly-0.9.1.jar \
org.apache.Spark.examples.JavaSparkPi \
Spark://haisen35.ux.uis.no:7077
```

4.3 Performance and analysis

In order to verify the veracity of our hypothesis *H0* we first tested the speed for Spark in compare with Hadoop to see whether there are some improvement between the Hadoop and Spark and then we will carry out experiment to see the effeteness and correctness of the implementation of Spark version of PSO. The reason why we do not compare with the simple PSO is because the extended amount of time that the serial PSO based on our experiment settingwould take.

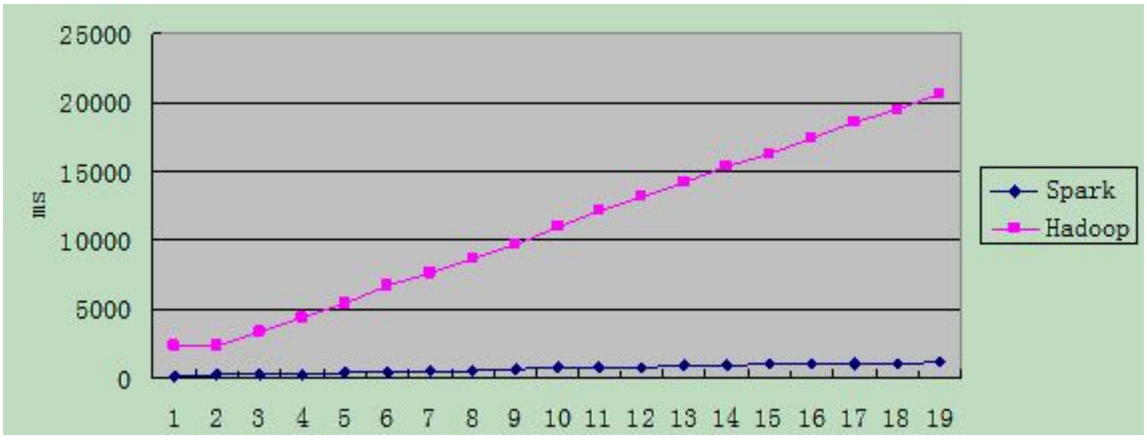
The experiment is carried out based on all the controllers set in UiS campus.

The first experiment is the running time between Hadoop and Spark with different PSO evolvement round. The horizontal axis is the evolvement round from 1 to 19 and the vertical axis is the running time of PSO in different context in the time unit of millisecond.

We could see that if we have only one evolvement round in the PSO, the time for Spark and hadoop could be around 1 second to 2 second which does not make too much difference. However, when the evolvement round reaches to 19, there is a huge difference between Spark and hadoop. The Spark running time stays nearly the same when the evolvement round is small, however the running time of Spark is around 20 second when the evolvement round reaches 19 which could definitely be the bottleneck of lots of applications.

From the chart, we could observe that the time of hadoop in respond to the evolvement round time increases dramatically while the time of Spark increase little while running from 1 evolvement round to 19 evolvement rounds. It is because that we utilize the Spark RDD data structure to avoid constant hard disk writing operation which is used in Hadoop.

When there is substantial amount of controller data, we need to write all the particle information which each has all the information of the controllers into the database at the end of each round and read all the data in from the database at the start of the next evolvement round. Because of the IO operation takes a lot of time, that is the reason why Hadoop uses much more time than Spark.

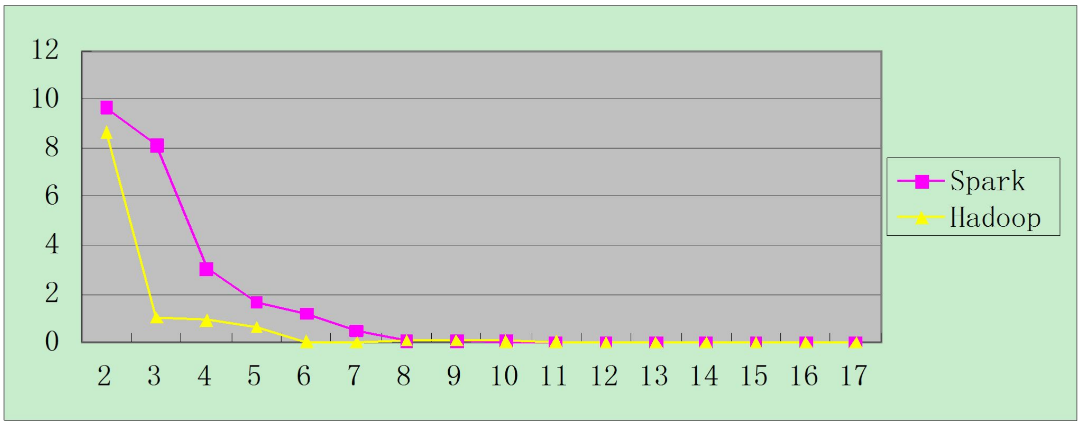


Next we will compare the accuracy of both Spark and Hadoop to see whether the Spark still have more accuracy or even the same accuracy with Hadoop.

We set the experiment to be like this, the horizontal axis is the evolvement round of PSO and the vertical axis is the distance towards PSO global best position which is the ultimate best result.

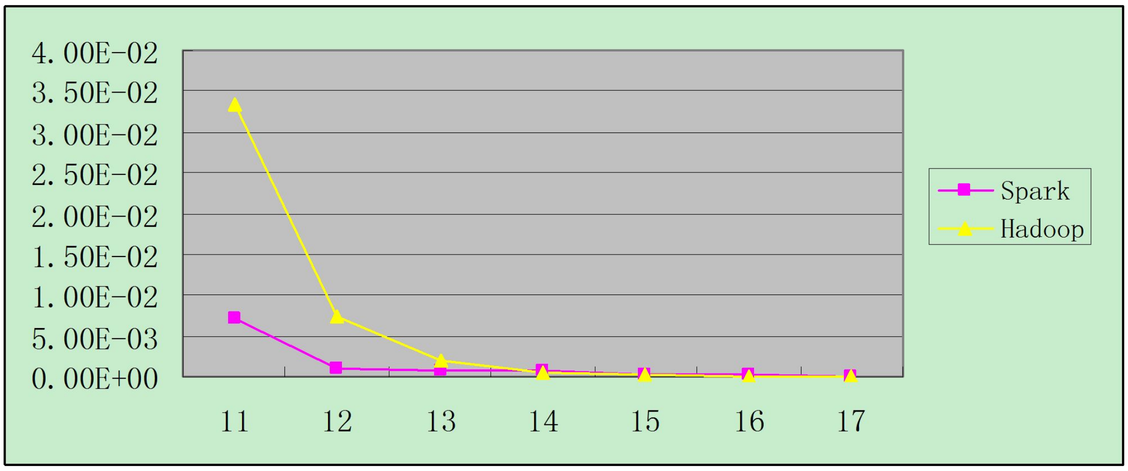
From the chart, we could see that the horizontal axis starts from round 2, the reason it starts from 2 is because during the first evolvement the positions of all the controllers are randomly set, there is no point to compare the result of the randomly select parameters. That is the reason why we start from the round 2.

As each time we draw the char, it is slightly different from the each other, because the PSO algorithm contains randomly variable selection. from the chart we could see, both Spark and Hadoop are far from the optimized value at the evolvement round 2, but Hadoop is slightly better most of the time at the early evolvement period. But as the evolvement round increases, both Spark and Hadoop have better result compare with before. As the chart shows, both Spark and Hadoop are nearly to the optimized when the round has reached 8. however from the chart it is difficult to see which one is better when they reach the round 9, the reason is because the dimension of the vertical axis does not allow the high resolution when it reaches zero, therefore in order to clarify the detail of which one is better after round 8, we need to have higher resolution chart.



The following chart is a higher resolution of what the accuracy vs evolvement round. From the chart we could easily see that although at the initial rounds the Hadoop has smaller value which means it is closer to the optimized value, however when both of them have almost reached the optimized value, Spark performs better than Hadoop when arriving at the global best.

However, as the chart shows, both of them have nearly arrived at the global best after around 15 round and both of them could definitely reach the optimized global value.



As each time, the initialization of positions and velocities are consisting of random value generation. Therefore each time we generate the chart above, We could create a new chart. However if we average all the results, we could see that both the Spark and Hadoop have nearly the same figure. That is because it is the same distributed PSO algorithm that we have implemented on both the Spark platform and Hadoop Platform.

Another information we could get from the chart is that averagely the experiment reaches its global best which means the distance between its predicted point and the global best is zero in around 17 times. From the chart 1, the experiment shows that hadoop uses 175000 millisecond while Spark uses 5500 millisecond. It means the running time of Spark is 175000/5500 = 32 times faster than hadoop in the calculation of interactive distributed PSO.

5 Conclusion

This thesis contributes to the solving optimization problems in useful time. To address this problem this thesis suggests the utilization of the Spark platform together with our improved Particle Swarm Optimization algorithm. This thesis describes how to adapt the classic Particle Swarm Optimization algorithm to the distributed big data platform Spark.

The solution of the problem is to firstly define the Spark Resilient Distributed Dataset for our PSO algorithm and then create the initialization algorithm for parallel PSO. The main processing algorithm consists of Foreach Operation design and Collect Operation design.

We then implement our algorithm using Java and test it in the use case SEEDS project. The experiments show that Spark could compute faster then the well know Hadoop and could have the same accuracy as the Hadoop version of parallel PSO.

However, one of the main Spark's drawback is the inability of automatically configure new nodes when they are required. Therefore, when we want to expand our cluster, we have to stop the whole cluster and restart it again. Because SEEDS is a real time system, this could cause temporary unstableness and inaccuracy of the system.

6 Reference

[1] SEEDS_D2.8_Energy Control Strategy First Version_r0.pdf， Available online: http://seeds-fp7.eu/documents.php

[2] SEEDS_D5.3_Report on optimization_r0.pdf， Available online: http://seeds-fp7.eu/documents.php

[3] https://developers.google.com/appengine/docs/python/dataprocessing/

[4] http://discoproject.org/

[5] cs.armstrong.edu/saad/csci8100/pso_tutorial.pdf

[6] Aljarah, Ibrahim, and Simone A. Ludwig. "Parallel particle swarm optimization clustering algorithm based on mapreduce methodology." Nature and Biologically Inspired Computing (NaBIC), 2012 Fourth World Congress on. IEEE, 2012.

[7] SEEDS_D2.8_Energy Control Strategy First Version_r0.pdf， Available online: http://seeds-fp7.eu/documents.php

[8] S. Andradóttir, Aglobal search method for discrete stochastic optimization.SIAM J. Control Optim., vol. 6, no. 6, pp. 513-530, May 1996.

[9] S. Andradóttir, Accelerating the convergence of random search methods for discrete stochastic optimization. ACM Trans. Model. Comput. Simul., vol.9, no. 4, pp. 349-380, Oct. 1999.

[10]A.Schwartz. A Reinforcement Learning method for maximizing undiscounted rewards[J], In: Proceedings of the Tenth International Conference on Machine Learning, 1993,pp. 298-305.

[11]K. Apt and T. Radzik, “Stable partitions in coalitional games,”arXiv:cs/0605132v1 [cs.GT],May 2006.

[12] OWEN G. Game theory[M]. 3rd edition. San Diego: Academic
Press, 1995.

[13] Yu, Jianbo, Shijin Wang, and Lifeng Xi. "Evolving artificial neural networks using an improved PSO and DPSO." Neurocomputing 71.4 (2008): 1054-1060.

[14] Kecskés, István, et al. "PSO and GA optimization methods comparison on simulation model of a real hexapod robot." Computational Cybernetics (ICCC), 2013 IEEE 9th International Conference on. IEEE, 2013.

APA

[15] Cai, Zhuoran, et al. "A modular spectrum sensing system based on PSO-SVM." Sensors 12.11 (2012): 15292-15307.

[16] Yun, Ling, Peng Qionglin, and Xiao Shenping. "The Spectrum Analysis Optimization for Band-limited Discrete Spectrum Continuous Signal Based on PSO." *Intelligent System Design and Engineering Applications (ISDEA), 2013 Third International Conference on*. IEEE, 2013.

[17] Wu, Zhao Xia, et al. "Analysis of FBG reflection spectrum with PSO algorithm."*Applied Mechanics and Materials* 182 (2012): 1953-1957.

[18] Rashvand, Habib, and Han-Chieh Chao. *Dynamic ad hoc networks*. The Institution of Engineering and Technology, 2013.

[19] Winston, Ojenge, et al. "PSO of Neural Networks to Predict Busy Times of Cellular Traffic for Assignment to TV Idle Channels by Cognitive Radio."*Modelling Symposium (EMS), 2013 European*. IEEE, 2013.
[20] Si, Wa, et al. "An improved PSO method for energy saving system of office lighting." *SICE Annual Conference (SICE), 2011 Proceedings of.* IEEE, 2011.
[21] Rehmani, Mubashir Husain, Stéphane Lohier, and Abderrezak Rachedi. "Channel bonding in cognitive radio wireless sensor networks." *Mobile and Wireless Networking (iCOST), 2012 International Conference on Selected Topics in*. IEEE, 2012.
[22] Dutta, Raju, Shishir Gupta, and Mukul K. Das. "Efficient Statistical Clustering Techniques for Optimizing Cluster Size in Wireless Sensor Network." *Procedia Engineering* 38 (2012): 1501-1507.
[23] Rappaport, Theodore S. Wireless communications: principles and practice. Vol. 2. New Jersey: Prentice Hall PTR, 1996.
[24] Apt, Krzysztof R., and Tadeusz Radzik. "Stable partitions in coalitional games." arXiv preprint cs/0605132 (2006).
[25] Kennedy, James, and Russell Eberhart. "Particle swarm optimization." Proceedings of IEEE international conference on neural networks. Vol. 4. No. 2. 1995.
[26] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.
[27] Zaharia, Matei, et al. "Spark: cluster computing with working sets." Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. 2010.
[28] BAKİROV, Aslan, Kevser Nur ÇOĞALMIŞ, and Ahmet BULUT. "Scalable sentiment analytics."
[29] Yu, Jianbo, Shijin Wang, and Lifeng Xi. "Evolving artificial neural networks using an improved PSO and DPSO." Neurocomputing 71.4 (2008): 1054-1060.
[30] Rameshkumar, K., R. K. Suresh, and K. M. Mohanasundaram. "Discrete particle swarm optimization (DPSO) algorithm for permutation flowshop scheduling to minimize makespan." Advances in Natural Computation. Springer Berlin Heidelberg, 2005. 572-581.
[31] http://en.wikipedia.org/wiki/Particle_swarm_optimization

[32] Eberhart, Russell C., and Yuhui Shi. "Particle swarm optimization: developments, applications and resources." *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*. Vol. 1. IEEE, 2001.

[33] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." *Communications of the ACM* 51.1 (2008): 107-113.

[34] White, Tom. Hadoop: The Definitive Guide: The Definitive Guide. O'Reilly Media, 2009.

[35] Zaharia, Matei, et al. "Spark: cluster computing with working sets." Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. 2010.

7 Appendix

A1: system configuration of Spark platform

Name	**Value**
Java Home	/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.55.x86_64/jre
Java Version	1.7.0_55 (Oracle Corporation)
Scala Version	version 2.10.3
Spark.app.name	Spark shell
Spark.driver.host	ip-172-31-14-144.ec2.internal
Spark.driver.port	47224
Spark.fileserver.uri	http://172.31.14.144:51972
Spark.home	/root/Spark
Spark.httpBroadcast.uri	http://172.31.14.144:49080
Spark.local.dir	/mnt/Spark
Spark.master	Spark://ec2-54-86-164-237.compute-1.amazonaws.com:7077
Spark.repl.class.uri	http://172.31.14.144:58145
awt.toolkit	sun.awt.X11.XToolkit
file.encoding	UTF-8
file.encoding.pkg	sun.io
file.separator	/
java.awt.graphicsenv	sun.awt.X11GraphicsEnvironment
java.awt.printerjob	sun.print.PSPrinterJob
java.class.version	51.0
java.endorsed.dirs	/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.55.x86_64/jre/lib/endorse
java.ext.dirs	/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.55.x86_64/jre/lib/ext:/usr/java/packages/lib/ext
java.home	/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.55.x86_64/jre
java.io.tmpdir	/tmp

Name	Value
java.library.path	/root/ephemeral-hdfs/lib/native/
java.runtime.name	OpenJDK Runtime Environment
java.runtime.version	1.7.0_55-mockbuild_2014_04_18_00_21-b00
java.specification.name	Java Platform API Specification
java.specification.vendor	Oracle Corporation
java.specification.version	1.7
java.vendor	Oracle Corporation
java.vendor.url	http://java.oracle.com/
java.vendor.url.bug	http://bugreport.sun.com/bugreport/
java.version	1.7.0_55
java.vm.info	mixed mode
java.vm.name	OpenJDK 64-Bit Server VM
java.vm.specification.name	Java Virtual Machine Specification
java.vm.specification.vendor	Oracle Corporation
java.vm.specification.version	1.7
java.vm.vendor	Oracle Corporation
java.vm.version	24.51-b03
line.separator	
os.arch	amd64
os.name	Linux
os.version	3.4.37-40.44.amzn1.x86_64
path.separator	:
sun.arch.data.model	64
sun.boot.class.path	/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.55.x86_64/jre/lib/resources.jar:/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.55.x86_64/jre/lib/rt.jar:/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.55.x86_64/jre/lib/sunrsasign.jar:/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.55.x86_64/jre/lib/jsse.jar:/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.55.x86_64/jre/lib/jce.jar:/usr/lib/jvm/java-1.7.0-

Name	Value
	openjdk-1.7.0.55.x86_64/jre/lib/charsets.jar:/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.55.x86_64/jre/lib/netx.jar:/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.55.x86_64/jre/lib/plugin.jar:/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.55.x86_64/jre/lib/rhino.jar:/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.55.x86_64/jre/lib/jfr.jar:/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.55.x86_64/jre/classes
sun.boot.library.path	/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.55.x86_64/jre/lib/amd64
sun.cpu.endian	little
sun.io.unicode.encoding	UnicodeLittle
sun.java.command	org.apache.Spark.repl.Main
sun.java.launcher	SUN_STANDARD
sun.jnu.encoding	UTF-8
sun.management.compiler	HotSpot 64-Bit Tiered Compilers
sun.os.patch.level	unknown
user.country	US
user.dir	/root/Spark/bin
user.home	/root
user.language	en
user.name	root
user.timezone	Universal

Classpath Entries

/root/ephemeral-hdfs/conf	System Classpath
/root/ephemeral-hdfs/conf	System Classpath
/root/ephemeral-hdfs/conf	System Classpath
/root/Spark/assembly/target/scala-2.10/Spark-assembly_2.10-0.9.1-Hadoop1.0.4.jar	System Classpath
/root/Spark/conf	System Classpath