Genetic algorithm for Travelling Salesman Problem using MapReduce

Hnin Thant Lwin
University of Computer Studies,Yangon
hninthantlwin@gmail.com

Abstract

The Travelling Salesman Problem (TSP) is one of the hardest and the most fundamental problems in Computer Science. Although several techniques have been used in the past to reduce the running time of TSP, Genetic algorithms can reduce the running times of NP-complete problems substantially and have the capability of being parallelized. MapReduce is a parallel programming paradigm currently use and Hadoop is one of the most popular MapReduce frameworks because its robust, well designed and scalable file system. In this paper we use a genetic algorithm and parallelizing it on MapReduce Hadoop framework to reduce the running time of Travelling Salesman Problem.

1. Introduction

Genetic algorithm is heuristic optimization method which mimics the process of natural evolution. Optimization problems like Traveling Salesman require a lot of computer resources to be solved even if we use genetic algorithm as optimization method. Because one computer machine is not capable to resolve problems of this magnitude, parallel implementation of genetic algorithm is one solution.

Parallel Genetic Algorithms [8] normally split a problem space into a number of smaller sub-spaces, then explore sub-optimal solutions for each sub-space, and finally find out a set of optimal solutions based on the sub-optimal solutions. PGAs can not only reduce the execution time, but also can tackle more complex problems by taking advantage of distributed computing systems. Furthermore, PGAs are more versatile than their corresponding sequential version as they have a less possibility of getting stuck in local optima.

The MapReduce model [2] provides a parallel design pattern for simplifying application developments in distributed environments. This model can split a large problem space into small pieces and automatically parallelize the execution of small tasks on the smaller space. It was proposed by Google for easily harnessing a large number of resources in data centers to process data-intensive applications and has been proposed to form the basis of a "data center computer" [5]. This model allows users to benefit from advanced features of distributed computing without worrying about the difficulty of coordinating the execution of parallel tasks in distributed environments.

This paper presents the parallel implementation of genetic algorithm as a MapReduce task for the Travelling Salesman Problem. The framework for running MapReduce job uses Hadoop because it can reduce the run time of NP-Complete with the high number of inputs.

2. Related Work

Much research has been done in genetic algorithm using MapReduce model. Abhishek Derma, Xavier Llor_a, David E. Goldberg and Roy H. Campbell[13] described the algorithm design and implementation of GAs on Hadoop. The convergence and scalability of the implementation has been investigated. Adding more resources would enable them to solve even larger problems without any changes in the algorithm implementation.

Dino Keč o a and Abdulhamit Subasi [1] develop a parallelization of genetic algorithms using Hadoop Map/Reduce in 2011. In this work they model for parallelization of genetic algorithm shows better performances and fitness convergence than model presented in Scaling Genetic Algorithms using MapReduce developed by Abhishek Verma, XavierLlor'a, David E. Goldberg, Roy H. Campbell, but their model has lower quality of solution because of species problem. They also said that in future work both models should be used for solving different problems, like TSP (Traveling Salesman Problem).

Siddhartha Jain and Matthew Mallozzi[14] analyze the possibility of parallelizing the Traveling Salesman Problem over the MapReduce architecture. They present the serial and parallel versions of two algorithms - Tabu Search and Large Neighborhood Search. They compare the best tour length achieved by the Serial version versus the best achieved by the MapReduce version.

3. Background

3.1. The Travelling Salesman Problem

Travelling salesman problem is one of the well know and extensively studied problems in discrete or combination optimization and asks for the shortest roundtrip of minimal total cost visiting each given city(node) exactly once. Cost can be distance, time, money energy ,etc. TSP is an NP-hard problem ad researchers especially mathematicians and scientists have been studying to develop efficient scientists have been studying to develop efficient solving method since 1950's.Because it is so easy to describe and so difficult to solve. Graph theory defines the problem as finding the Hamiltonian cycle with the least weight for a given complete weighted graph.

The travelling salesman problem is finding a shortest possible cycle visiting every city in a map given the set of cities and pair wise distances between them. This modeling of this problem can also be done with an undirected weighted graph. The vertices of the graph are cities. The edges of the graph are distances. The Travelling Salesman Problem is a Hamiltonian Cycle if distances are either 0 or 1. [10]

The travelling salesman problem is one the toughest and fundamental problems in Computer Science. Genetic Algorithms have proven efficient in solving the Travelling Salesman Problem in last 30 years or so. Travelling Salesman is an NP-Hard problem. We propose to reduce the run time of this problem by designing a genetic algorithm.

3.2. MapReduce Model

Google introduced MapReduce, inspired by the map and reduce primitives in functional languages. It is used to enable users to develop large-scale distributed applications. MapReduce parallelizes large computations easily since each map function runs independently and provides fault tolerance through re-execution. In MapReduce, the input is a set of key/value pairs, and the output is a set key/value pairs. MapReduce operation breaks down to two functions; map and reduce. Operations on a set of pairs occur in three stages: the map stage, the shuffle stage and the reduce stage as shown on figure 1.

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 crucial that the map operation is stateless - that is, it operates on one pair at a time. This allows for easy parallelization as different inputs for the map can be processed by different machines.[3] During 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 machine. This occurs automatically, and is seamless to the programmer. [4]

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 Map/Reduce computation: all of the maps need to finish before the reduce stage can begin. [9]

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. Overall, a program in the Map/Reduce paradigm can consist of many rounds of different map and reduce functions performed one after another [6].

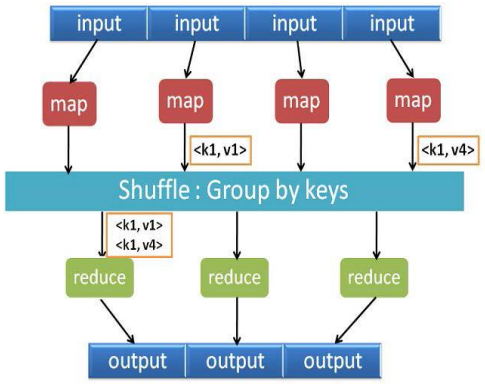

Figure 1: Operation phases in Map/Reduce programming model

3.3. Hadoop

Hadoop [7] is a java open source implementation of MapReduce sponsored by Yahoo. The Hadoop project is a collection of various subprojects for reliable, scalable distributed computing. The two fundamental subprojects are the Hadoop MapReduce framework and the Hadoop Distributed File System (HDFS). HDFS is a distributed file system that provides high throughput access to application data [7]. HDFS has master/slave architecture. The master server, called NameNode, splits files into blocks and distributes them across the cluster with replications for fault tolerance. It holds all metadata information about stored files. The HDFS slaves, the actual store of the data blocks called DataNodes, serve read/write requests from clients and propagate replication tasks as directed by the NameNode.

The Hadoop MapReduce is a software framework for distributed processing of large data sets on compute clusters [7]. It runs on the top of the HDFS. Thus data processing is collocated with data storage. It also has master/slave architecture. The master, called Job Tracker (JT), is responsible of : (a) Querying the NameNode for the block locations, (b) considering the information retrieved by the NameNode, JT schedule the tasks on the slaves, called Task Trackers (TT), and (c) monitoring the success and failures of the tasks.

4. A Parallel Genetic Algorithm based on Hadoop MapReduce for TSP

The Genetic Algorithm consists of four major stages: Generation of Random population, Selection, Crossover and Mutation. The first step is only performed for the first iteration, but the other three steps are performed each iteration. The process of Selection is one of the critical phases in the process of evolution. The preliminary steps of Selection such as calculating the cost of each map and Ranking individuals are computationally expensive. These steps are a good

candidate for parallelization or splitting tasks. The pseudo code for the each Phase is as follows.

Step 1: Initialize the Population

Step2: **Map (LongWritable Key, Text Value, OutputCollector oc,)**
Begin: Calculate the Cost of every individual (Value)
: Calculate the Rank of every individual (Cost, Selective_Pressure, Size_of_Population)
: Assign id (key) to each individual in the Population.
: Generate Value for the Reduction phase: String Route|Cost|Rank
: Emit the Key Value pair for the Reducer Phase.
: The key Value pairs are passed into the reducer phase.
Note: The key and value emitted by the output collector are of the Text class.

Step3: **Reduce (Text key, Iterator<Text> Values, OutputCollector<Text,Text>)**
Begin: While (Values.hasNext())
: Same Ranked Individuals in the population are Randomly Deleted.
: Two individuals in the population are randomly chosen and they participate in Crossover.
: Parents with ranks over threshold are allowed to move into the next stage.
: Two randomly chosen cities in every individual are randomly swapped.
: The output collector emits the key values in the form: (Individual id , Newly Generated Individual)

The input to the Reduce function is the output of the Map function. The Iterator<Text> values are the list of values on which Genetic Algorithm works on. The output collector of the genetic algorithm emits individual id and new individuals, both objects of the Text class designed for Hadoop. The Map and Reduce operations carry on till near optimal or optimal values are reached. The number of iterations will be decided by experiments. In every iteration, the output written by the Reduce function is the input for the map function.

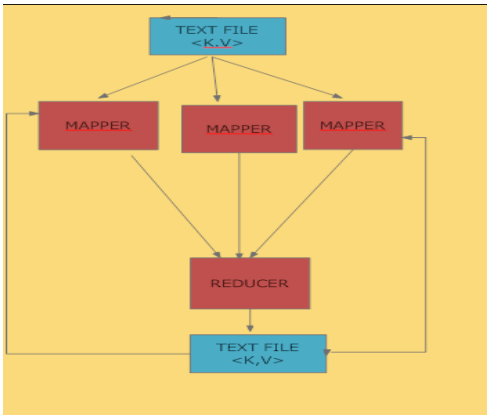

Figure 2: the Mapper/Reducer architecture

The Mapper Reducer architecture is described in Figure 2.It has been designed keeping in mind the nature of problem being solved in this paper. Genetic Algorithms are iterative in nature; therefore the population set undergoes repeated mapping and reduction. The key and values in the input file are shuffle using the map function, and input file is split across various mappers. The calculation of cost and rank are computationally intensive, therefore a good candidate for performing map operations. The computed cost, ranks, routes and id are passed into one reducer function. The reducer function gathers all the values from all the mappers. Later the Reducer performs elimination of individuals of the same rank randomly. This process is followed by Crossover and further Mutation.

5. Experiments

Amazon elastic map reduce is one of the computing platforms offered by Amazon to perform Map Reduce operations. The genetic algorithm was also run on Amazon's hardware using the wide variety of software offered by Amazon. The input files and output files were stored in Amazon's s3 bucket. The jar file of the Map Reduce job is also stored in one of Amazon's bucket. The parameters passed as arguments were path of the input file, the path of the output files, the name of the class containing the main function and the number of iterations. A job with all these parameters is created and runs across the hardware configured according to the inputs requested in the job. The results of the job are stored in one of the s3 buckets.

The reading and writing into text files is time consuming. Hadoop also creates chunks of the files, therefore it does not read and write into file but many files. The performance of the algorithm also depends on the initial random population. The larger the initial population, the mapreduce version of the algorithm runs a lot slower. There are other internal factors that affect the run time of the algorithm such as the selective pressure and participation of fit parents into the next generation. If the selective pressure is low, each generation will consist of more individuals. If the selective pressure is too high then the algorithm faces the problem of local minima and maxima.

Experiments were conducted on Amazon Web Services on different data sets. The size of the data set varied according to the number of cities. The range on which the experiments were carried out was from 10-30 with an interval of 10. The distances between the cities and routes are randomly generated using a random number using Random class in Java.Util. The nodes varied from 2-10 with an interval of 2.

Different instances were being used for experiments such as small memory, large memory, high computation, medium computation and low computation. The performance of the Map-Reduce version of the genetic algorithm on 10 Cities is in following figure.

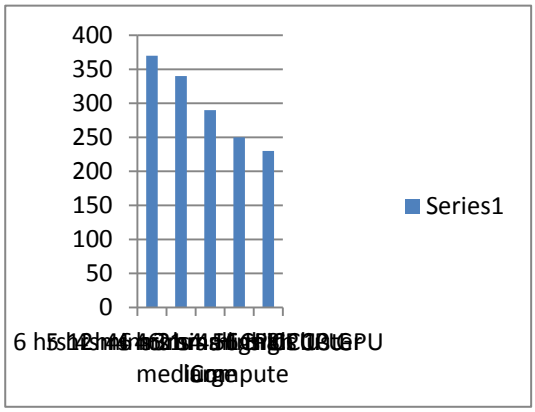

Figure 3. The performance of the Map-Reduce version of the genetic algorithm on 10 cities.

With higher number of nodes and better computational ability, the run time of the algorithm was reduced. The highest time was observed with 2 nodes and lowest time was observed with 10 nodes.

The performance of the algorithm on 20 Cities is as follows. The running time was less with better instances and lesser number of nodes but not good enough.

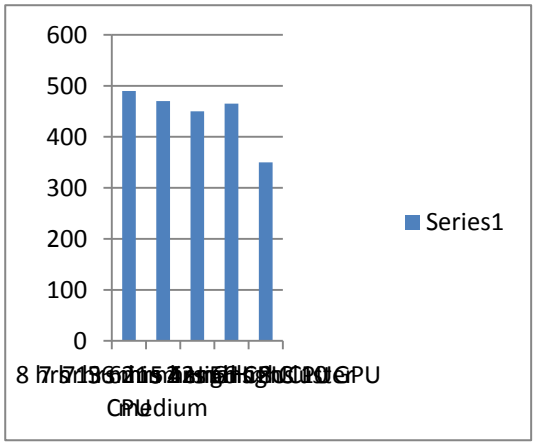

Figure 4. The performance of the Map-Reduce version of the genetic algorithm on 20 cities.

The performance of the algorithm with 20 cities was no different from 10 cities. There was no particular trend of speed up but running time of the algorithm was reduced with better instances and higher number of instances. In most cases optimal values were not reached in the specified number of iterations. More iteration required for the genetic algorithm to reach the optimal value.

According to experiment, the results were similar for different number of cities. Five dataset for different cities showed the same pattern. The increase in the number of nodes and better quality of computing instances showed reduced run time. The drop in the run time was not significant but worth mentioning. The small computing instances took the highest time demonstrating the fact that genetic algorithms are a good candidate for parallelization. The GPU instances shows the best performance which could hint at GPUs' being the best hardware available for parallelization for Genetic algorithm.

6. Conclusions

Genetic Algorithms are easy to apply to a wide range of problems, from optimization problems like the travelling salesperson problem, to inductive concept learning, scheduling, and layout problems. The results can be very good on some problems, and rather poor on others. Implementing these GAs with modification have proven to be very useful, especially hierarchical GAs, resulting in faster and more robust algorithms. In this paper we proposed the use of a Parallel Genetic Algorithm (PGA) for TSP using exploiting Hadoop MapReduce. The run time of the genetic algorithm is not as high as compared to other parallel versions. This could be I/O latencies caused by HDFS (Hadoop Distributed File System). The HDFS divided texts into smaller chunks on stores it on several data nodes. The reading and writing into these text files causes severe I/O delays. Also the Mapper-Reducer architecture could be improved further by adding more reducers and utilize the power of parallel processing.

References

[1]Dino Keco, Abdulhamit Subasi,Parallelization of genetic algorithms using Hadoop Map/Reduce(2011)

[2] E.Cant´u-Paz, "Efficient and Accurate Parallel Genetic Algorithms", Kluwer Academic Publishers, Norwell, MA, USA, 2000, ISBN: 978-0-7923-7221-9

[3] Howard Karlo, Siddharth Suri, Sergei Vassilvitskii ,A Model of Computation for MapReduce

[4]Jeffrey Dean and Sanjay Ghemawat,MapReduce: Simplified Data Processing on Large Clusters –

[5]. B. He, W. Fang, Q. Luo, N. K. Govindaraju, T. Wang, Mars: a mapreduce framework on graphics processors, in: Proceedings of the 17th international conference on Parallel architectures and compilation techniques,
Toronto, Ontario, Canada, 2008, pp. 260–269.

[6] Satish Narayana Srirama, Pelle Jakovits, Eero Vainikko ,Adapting scientific computing problems to clouds using MapReduce

[7]Hadoop project, http://lucene.apache.org/hadoop (2011).

[8] J. Schutte, J. Reinbolt, B. Fregly, R. Haftka, and A. George, "Parallel global optimization with the particle swarm algorithm," International
Journal for Numerical Methods in Engineering, vol. 61, no. 13, 2004

[9] Shadi Ibrahim _ Hai Jin _ Lu Lu _
Bingsheng He _ Gabriel Antoniu _ Song
Wu, Handling Partitioning Skew in MapReduce using *LEEN(2012)*

[10]http://en.wikipedia.org/wiki/Travelling_salesman_problem

[11]http://hadoop.apache.org/common/docs/r0.19.2/hdfs_design.html

[12]hadoop.apache.org/common/docs/r0.19.2/images/hdfsarchitecture.gif
[13]Abhishek Verma, X. L. (oct2009). *Scaling Simple and Compact Genetic Algorithmsand MapReduce.* Chicago, IL.
[14] Siddhartha Jain Matthew Mallozzi "Parallel Heuristics for TSP on MapReduce(2010)