Parallelization of Genetic Algorithms using MapReduce

Suman Saha
Faculty of Creative Arts, Technologies & Science
Department of Computer Science and Technology
University of Bedfordshire
London Luton, United Kingdom
sumansaha1981@gmail.com

Abstract — With an exponential increase in the use of Internet for last decades, the use of large-scale data-intensive applications has become one of the important area of computing. Search engines like Google maintains a large index of data for their efficient search operations. Millions of users are connected to this search engine every day and every second millions of search operations are performed. The efficient indexing of large-scale data made it possible for Google to run such effective search engine. MapReduce is a programming model proposed by Google, which made it possible to process such a large-scale of data in a parallel and distributed network where thousands of commodity computing devices are connected to a large cluster. MapReduce reduces the computational time for processing large-scale data by parallelizing the computation using multiple computing nodes and Map and Reduce primitives. Evolutionary algorithms such as Genetic Algorithms are also inherently parallel in nature and are used to optimize no-linear problems. In this paper, the various research works conducted to parallelize Genetic Algorithms using MapReduce are presented. Some of them are: 1) parallelization of simple and compact GAs using MapReduce, 2) A MapReduce based hybrid Genetic Algorithm using island approach for solving time dependent vehicle routing problem, 3) A parallel Genetic Algorithm based on Hadoop MapReduce for the automatic generation of JUnit test suites, 4) scaling populations of a Genetic Algorithm for Job Shop Scheduling problems using MapReduce and 5) A Genetic Algorithm by Hybrid Approach for MapReduce.

Index Terms — MapReduce programming model, Genetic Algorithms, Parallelization, large-scale cluster, Hadoop distributed frame work, vehicle routing problem (VRP), Search-Based Software Testing (SBST).

I. INTRODUCTION

In the recent decades the use of internet has been proliferated and thus, the application of data-intensive framework is increased enormously (Uysal et al., 1998; Beynon et al., 2000; Foster, 2003; Mattmann et al., 2006). Genetic Algorithms (GAs) act as effective global solution for large scale problems such as non-linear optimization (Gallagher and Sambridge, 1994), clustering (Frnti et al., 1997) and job scheduling (Sannomiya et al., 1999). GAs are suitable solution for parallelization (Cantu-Paz, 2000).

Verma et al. presented the work which shows the parallelization of data-intensive computing using GAs and MapReduce models (Dean and Ghemawat, 2008). Their work contributed the following two tasks: 1) parallelize GAs using MapReduce model and 2) Demonstrate the scalability of the implemented MapReduce program to large scale problems (Verma et al., 2009). In section-III-A, the work done by Verma et al. is presented.

A vehicle routing problem (VRP) is another important problem in the field of transportation, distribution and logistics. This is a famous combinatorial optimization problem (Kondek et al., 2012). Kondek et al. presented a MapReduce based hybrid genetic solution for large-scale VRP. This solution uses Island approach and it manages dynamic network with fluctuating link travel time. Routing construction algorithms (NNC, Savings and Random) is used and a mixture of both random and locally optimized population is formed using a hybrid approach (Kondek et al., 2012).

According to Geronimo et al., Genetic Algorithms (GAs) can be used for software testing as GAs can solve several issues related to this problem (Geronimo et al., 2012). Software testing problem requires some search-based methods. However, there is very limited utilization of Search-Based Software Testing (SBST) in real practice. Most of the Search-based software engineering requires larger computational resources such as more number of processors, memory and storage units. One way of solving this issue is to parallelize the search-based techniques as most of these techniques are inherently parallel in nature, thus, parallel implementation of search based approaches enhances the performance.

Geronimo et al. presented a parallel GA which automatically produces test suites. The MapReduce programming model implemented on Hadoop distributed frame work supports cloud computing as well as graphic cards, for this reason Geronimo et al. have selected Hadoop as a suitable and highly scalable GA parallelization framework. They have conducted an analysis of their proposal to verify the speed-up with respect to the sequential execution. They have used real world open source libraries for their analysis (Geronimo et al., 2012).

Zec Amer et al. also presented the Map reduce technology with parallel implementation of genetic algorithm. His work also describes the scaling of map reduce with genetic
algorithm. By ‘divide and conquer’ technology parallel algorithm works to handle the multiple threads or processes simultaneously. In this paper several methods parallelize one population and some converts it into subpopulation. This paper shows four major types of genetic algorithm (Zec Amer et al., 2012).

- Mater-slave genetic algorithm with single population
- Multiple population genetic algorithm
- Fine grained genetic algorithm
- Hierarchical hybrids

This paper analyzes the parallel implementation of GA for map reduce by following types with respect to hybrid model (Zec Amer et al., 2012).

II. BACKGROUND

Large-scale distributed applications can be developed using MapReduce (Dean and Ghemawat, 2008) programming model. Hadoop is initiated by Apache software Foundation which is an open source implementation of the MapReduce programming methodology.

III. RESEARCH RESULTS

A. Scaling Simple and Compact Genetic Algorithms using MapReduce

1) MapReduce Framework

MapReduce model was proposed by Google Incorporation (Dean and Ghemawat, 2008). This is implemented from the “map” and “reduce” primitives present in the high level languages like Java. In this model a set of key/value pairs are taken as input and the produced output is also a set of key/value pairs. MapReduce library has two functions, they are: 1) Map function and 2) Reduce Function. User writes the Map function which takes an input key/value pair (k1,v1) and generates a set of intermediate key/value pairs (list(k2,v2)). All these intermediate values are grouped together with their associated key k2 and are forwarded to the Reduce function by the MapReduce framework. The Reduce function is also written by the user, which takes a key k2 and the associated set of values (list(v2)) as input and produces a possibly smaller set of values (list(v3)) by merging these values together. The intermediate values (v2) generated by the Map function, are forwarded to the Reduce function using an iterator list which provides a mechanism to manage the large data values. These large values are difficult to store in the memory. The conceptual prototypes of Map and Reduce functions are as shown below:

map(k1; v1) → list(k2; v2)
reduce(k2; list(v2)) → list(v3)

The input key/value pairs are taken from different domain than the output keys and values. Moreover, the intermediate keys and values belong to the same domain as the output keys and values.

Initially, the input data is partitioned into a set of M splits and the Map function is invoked by multiple nodes (machines) distributed across the network, and thus, these set of M splits are processed simultaneously by different nodes. A partitioning function is used to partition the intermediate key space into R pieces and the Reduce function is invoked by the multiple nodes and thus the intermediate keys and values are also processed in a parallel fashion. The partitioning function is hash(key)%R as per the default Hadoop (Apache Software Foundation, 2012) settings. The user initializes the number of partition R and specifies the partitioning function. An upper level data-flow diagram of MapReduce framework is shown in figure-1.

![MapReduce dataflow diagram](image-url)

Figure 1. MapReduce dataflow diagram

Verma et al. transformed and implemented two types of Genetic Algorithms into MapReduce framework to achieve the parallel processing of large scale data using multiple machines or nodes distributed across the network (Verma et al., 2009). They first demonstrated the parallel implementation of Simple Genetic Algorithms (SGAs) using MapReduce. Next the Compact Genetic algorithms were parallelized using MapReduce and implemented on Hadoop distributed framework.

2) Parallelize SGAs using MapReduce

The parallel implementation of SGAs was accomplished using MapReduce model. Each iteration of GA is considered as a separate MapReduce job. The user takes the command line parameters, generate GA population and initiate the MapReduce job (Verma et al., 2009).

a) Simple Genetic Algorithms (SGA)

The Selecto-recombinative genetic algorithms (Goldberg, 1989, 2002) work on the principles of selections and recombination techniques. This is one of the simplest genetic algorithms. The algorithm for this GA is as follows (Verma et al., 2009):
Step-1: The GA population is initialized with arbitrary individuals.
Step-2: The fitness value is obtained for the individuals.
Step-3: S-wise tournament selection without replacement (Goldberg et al., 1989b) technique is used to identify better result.
Step-4: Uniform crossover (considering a crossover probability pc=1.0) technique is used to form new individuals. New individuals are created by merging selected population.
Step-5: Fitness value for all offspring is assessed.
Step-6: Step 3 to 5 are repeated till there are no convergence criteria found.

b) Implementation of Map function for each iteration of the GA
The fitness value for individuals as well as all offspring (Step-2 and 5) is calculated and the best individual is stored to a global file in a Distributed File System (in this case HDFS: the Hadoop Distributed File System) using Map function as shown in Appendix-A: Algorithm-1. At the end of MapReduce, user reads these best individual values from all mappers and examines whether the convergence criteria are satisfied or not (Verma et al., 2009).

c) Implementation of Random Partitioner function for GA
Decentralized and distributed GA selection algorithms (Step-3) (Jong and Sarma, 1995) are better than locally executed selection operation. If the selection operation is carried out locally by the local nodes, it brings in the spatial constraint and decreases the selection pressure (Sarma et al., 1998). As soon as the execution of Map function completes the partitioner splits the intermediate key/value pairs (generated by the Map function) and send these keys and values to different Reducer functions. The default partitioner brings an artificial spatial constraint which leads to increase in the number of iteration in the convergence of GA or no convergence at all. A deviation from the uniform distribution occurs due to the same copy of the individual which is sent to a single reducer. Thus the parallelism degrades as the GA converges. Due to this reason the default partitioner method is overridden. The overridden partitioner method rearranges different individuals randomly across different reducer methods as shown in Appendix-A: Algorithm-2 (Verma et al., 2009).

d) Implementation of the Reduce function for each iteration of the GA
For reliable selection operation a kind of tournament selection without replacement (Goldberg et al., 1989a) in a sequential manner. The complete implementation is shown in Appendix-A: Algorithm-3 (Verma et al., 2009).

e) Optimizations for the MapReduce model
The optimization of the MapReduce implementation of Simple Genetic Algorithms is achieved by eliminating the serial initialization of the population which requires longer time (Verma et al., 2009).

3) Parallelize CGA using MapReduce
In Compact Genetic Algorithm (CGA) (Harik et al., 1998), the population is generated by a probabilistic model which gives better solutions by estimating the probability distribution. The probabilistic model comprises of a vector of probabilities which are employed to lead the further search by creating new candidate solutions as per the frequency. The CGA is one of the simplest estimation distribution algorithm (EDAs) (Pelikan et al., 2002; Larrañaga and Lozano, 2002).

a) MapReduce implementation of CGA on Hadoop distributed framework
Every single iteration of CGA is implemented as a distinct MapReduce job. The user receives the command line parameters and generates the initial probability vector splits and forwards it to MapReduce module (Verma et al., 2009).

b) Map implementation for each iteration of the CGA
Generation of two individuals is implemented using the Map function. As shown in Algorithm in appendix-A: Algorithm 4, A probability split Pi is given as input to the Map function and it generates the “tournamentSize” individuals splits and probability split as output. The Map function also checks the number of ones for both the individual and writes in to a global file in the Hadoop Distributed File System. All the Reducer functions when invoked from different nodes, read these values from the global file stored in the Hadoop file system (Verma et al., 2009).

c) Reduce implementation for each iteration of CGA
The reduce function implemented using a technique called “tournament selection without replacement” is mentioned by Goldberg et al. is shown in the Appendix-A: Algorithm-5. As per this technique, the winner is chosen by conducting a tournament among individuals and the probability vector is properly updated. The optimization technique used in CGA is same as the technique used in SGA (Verma et al., 2009).

Verma et al. referred the challenges involved in scaling the SGA and CGA using MapReduce frame work. They also investigated the convergence and scalability of the MapReduce implementation of SGA and CGA and found that more large-scale problem can be solved by adding more resources to the network (such as more number of nodes, storage, processors etc.) and without any changes to the algorithm implementation.
The results are shown in Appendix-B: Result-1 for SGA and Result-2 for CGA (Verma et al., 2009).

OneMax or Bit Counting problem (Schaffer and Eshelman, 1991) is used to examine the convergence and scalability of the MapReduce GA implementation. The OneMax problem can be defined as a problem where the objective is to find a bit-string $x$ that maximizes the following equation-1. The bit-string can be assumed as a vector $x$ consists of elements $x_1, x_2, ..., x_N$, where $x_i \in \{0,1\}$ and $i = 1$ to $N$:

$$F(x) = \sum_{i=1}^{N} x_i \text{...Equation–1.}$$

The OneMax problem was implemented on a Hadoop (Apache Software Foundation, 2012) cluster consists of 416 cores and 52 nodes. The more detail configurations of the Hadoop cluster used in this research work can be obtained from (University of Illinois, 2013). The experiments were conducted based on the following criteria: 1) Convergence Analysis, 2) Scalability with constant load per node, 3) Scalability with constant overall load and 4) Scalability with increasing the problem size.

B. A MapReduce Based Hybrid Genetic Algorithm Using Island Approach for Solving Time Dependent Vehicle Routing Problem

Parallelization of genetic algorithms is achieved using Island model approach. Multiple subpopulations are useful to maintain genetic diversity, as every single island can monitor a different search route through the search space. The route was enhanced by employing various local search techniques such as 2-opt. The algorithm design and implementation of Time Dependent Vehicle Routing Problem with Time Window (TDVRPTW) was executed on Hadoop. The results of the experiments exhibit the computation time is reduced and the efficiency is increased, thus a significant improvement was noticed (Kondekar et al., 2012).

Dantzig and Ramse (Dantzig and Ramse, 1959) proposed the VRP in 1959 and it has some relations with travelling salesman problem (TSP). Large-scale data-intensive applications require an effective distribution scheduling scheme to manage the huge load of E-commerce transactions. VRP is one of the basic fundamental problems of the logistic distribution. In the early version of VRP the road travel time is measured as constant throughout the distribution process and thus it is conceived on static network principles. However the real time road distribution network is affected by various factors such as traffic, emergencies, weather etc. The travel time between two nodes is not only depends on the distance but also on the departure time as well.

The aim of the TDVRPTW is to investigate the best way of organizing vehicle delivery route, so that some specific optimization goals can be obtained such as, minimizing the travel time and the number of vehicles. It has been verified that the early version of VRP is a NP (Non-deterministic Polynomial-time)-hard problem. Therefore, in case of TDVRPTW, the complexity in solving the problem is higher while traffic conditions and time windows are taken into consideration within a distributed environment. Time Dependent Vehicle Routing Problem with Time Window (TDVRPTW) is a kind of VRP which consider the time dependent travel speeds and some other complexities such as attending each customer within a given time window.

Kondekar et al. recommended a practical way to solve the TDVRPTW problem while taking into account four distinct traffic conditions (morning, afternoon, evening & night) and five different types of roads. The total spending time is reduced by employing a mathematical and MapReduce programming model with a hybrid Genetic Algorithm (GA) which uses the Island approach. TDVRPTW has increased much client’s attention with the design and implementation of intelligent transportation technology and meta-heuristic algorithms.

1) Implementation of Hybrid Genetic Algorithm using MapReduce programming model

Genetic Algorithm (Potvin and Bengio, 1994) is more effective optimization technique as compared to other conventional optimization methods such as tabu search technique, simulated annealing and branch and bound method. The parallel search technique makes GA more efficient. As Genetic Algorithms (GAs) are parallel in nature, MapReduce programming model can be used to parallelize the execution of GAs using multiple nodes connected to a distributed computing network environment.

The implementation of Kondekar et al. concentrated on merging the evolutionary concepts of natural selection and genetic with distributed and parallel processing. The best solution is achieved with hybrid GAs. GA is capable of combining other techniques with its framework to build a hybrid model.

Kondekar et al. presented in their research work a fusion of conventional Genetic algorithm with other local search and optimization techniques over massively parallel platform using MapReduce programming paradigm. The hybrid genetic algorithm proposed by Kondekar et al., utilizes optimization mechanisms such as saving heuristics, random method and NNC for initial population generation, local search methods like 2-opt for route enhancement. Island technique is employed in the proposed MapReduce implementation of hybrid genetic algorithm as shown in Appendix-C: Fig.-2.

2) Implementation of GA using Island based approach

In both serial and parallel computations, the Island mechanism is a famous way of devising the Genetic Algorithms. A distributed prototype of Island based genetic algorithm was developed by Kondekar et al. where each node runs its own algorithm and keeps track of its own subpopulation for search. Fitness-based probabilistic selection
is used by each node connected to the distributed networking environment. Diversity in the population is achieved by conducting the crossover and mutation separately in each island. Migration process is carried out within a certain time interval where the nodes shuffle a part of their population.

### C. A Parallel Genetic Algorithm Based on Hadoop MapReduce for the Automatic Generation of JUnit Test Suites

There were research works conducted on several testing problem such as functional testing, mutation testing, structural testing (both static and dynamic), test case prioritization, etc (McMinn, 2004). After investing so much effort, yet there is very limited application of Search-Based Software Testing (SBST) (Lakhoria et al., 2009 and Bertolino, 2007). To solve a large scale problem, search-based techniques require higher computational resources (Lakhoria et al., 2009 and Yoo et al., 2011). In other words, these large-scale search based approaches perform more efficiently when deployed in a massively parallel and distributed computing environment (Harman, 2007). Parallelization reduces the computational time and increases the efficiency in exploring the search-space. Furthermore, many of these search-based techniques are naturally parallelizable (Harman, 2007). The fitness function of GAs for each individual can be executed in parallel. This is possible only because GAs are population based techniques. However, there are very few research work carried out to parallelize the SBST. Yoo et al. recommended “one possible historical barrier to wider application of parallel execution has been the high cost of parallel execution architectures and infrastructure. ..While commodity PCs have significantly reduced the cost of such clusters, their management can still be a non-trivial task, restricting the potential availability for developers.” According to Yoo et al., the main obstacle in the use of parallel applications is the increased cost involved in the architecture and hardware design of the parallel computing systems. The use of inexpensive commodity computing devices in large cluster by organizations, such as Google Inc. (for Google File System) and Apache software foundation (for Hadoop distributed framework), minimizes the cost in a significant manner, but yet the administration and management of these massively parallel and distributed clusters is still a complex task which restricts users to make best out of these systems. With a significant increase in General Purpose computing on Graphical Processing Unit (GPGPU) and Cloud Computing provides an opportunity to SBST parallelization and a scope for multiple research options which can be further implemented in software industries.

The GPGPU is less expensive and requires less management cost than system with multiple computing devices (Yoo et al., 2011). Cloud computing facilitates its users with on demand resource allocation and de-allocation to easily scale up or scale down (for flexible scalability), simple resource management and definitely a parallel and distributes computing design and infrastructure.

With the help of Cloud Computing, organizations can use virtually unlimited computing resources without increasing the work load for IT maintenance and administration. Additionally, the on demand computational resource allocation and de-allocation facility in Cloud Computing helps to minimize the cost on hardware and software as the users need to pay only for computational resources they actually used.

Considering all these features of Cloud Computing, Geronimo et al., proposed a Genetic Algorithm which automatically produces test suites for Object Oriented Software and demonstrated the parallel implementation of this GA on Hadoop MapReduce framework (Apache Software Foundation, 2012).

Hadoop MapReduce is a parallel and distributed computing framework which facilitates fast processing of large-scale data in parallel by multiple nodes distributed across the network (which form the large cluster) using MapReduce programming model. Hadoop has become the de-facto standard for MapReduce implementation and now-a-days it has been extensively used by industries (Verma et al., 2009).

According to Geronimo et al., Hadoop MapReduce is an ultimate solution for parallelizing Genetic Algorithms with higher scalability as it provides support for clusters as well as cloud (Amazon, 2013), computing environment and on graphic cards (CUDA on Hadoop MapReduce, 2011).

The parallel GA (recommended by Geronimo et al.) takes the software and a set of test cases (as the initial population of random solutions) as input. The initial population are updated according to the given coverage criterion (i.e., branch coverage). The individual fitness verification in each iteration of the GA takes the most of the computational time. This fitness evaluation task is parallelized using the MapReduce programming model. The parallel GA produces as output an optimized version of the JUnit test suite which can go through as much branches of test cases as possible for the given software.

Geronimo et al. discussed the following topics in their research work:

1) A GA which produces test suites automatically

A step by step instruction for the GA is given (Geronimo et al., 2012).

2) A parallel GA implemented on Hadoop MapReduce

In this section they covered different parallelization strategies which depend upon the grain of parallelization to realize. They mentioned three level of parallelization:

- fitness evaluation level (i.e., global parallelization model);
• population level (i.e., coarse-grained parallelization or island model);
• individual level (i.e., fine-grained parallelization or grid model) (Stender, 1993).

3) PGA implementation on Hadoop MapReduce
The main techniques used to parallelize the proposed GA using MapReduce programming model are (Geronimo et al., 2012):

• Each iteration of the GA is treated as distinct MapReduce job
• Multiple Map functions are invoked from multiple distributed nodes attached to the Hadoop cluster to parallelize the chromosome fitness evaluation
• A single Reduce function is invoked to collect the output of all Map functions and run all the genetic functions such as crossover, mutation, survival selection and parents selection which are required to generate a new generation of population

The proposed implemented PGA on MapReduce model has the following modules (Geronimo et al., 2012):

• A Parallel Genetic Algorithm
• A Master node
• A number of Mapper nodes and a Reducer nodes
• InputFormat and OutputFormat: splits the data for inputs to the multiple Map functions and stores output of the Reduce function to Hadoop distributed file system

The proposed algorithm was evaluated with respect to the execution time and branch coverage (Geronimo et al., 2012). The execution time is calculated using system clock and the total time. The total time comprises of the following complements:

• InitTime is the total time needed for the Parallel Genetic Algorithm to initialize a Map function with the required data (such as SUT instrumented bytecode, JUnit, test cases). This information is required to run the fitness evaluation in every iteration
• EvalTime is the total time taken to evaluate the fitness of chromosomes
• RemainTime is calculated by the equation = TotalTime - (InitTime+EvalTime)

D. Scaling Populations of a Genetic Algorithm for Job Shop Scheduling Problems using MapReduce

1) Introduction
According to Darwinian development, a genetic algorithm is used one of the solution for solving the difficult problems which are occur in job shop scheduling (JSS). The range of the people in GA special effects the value of solution. To avoid such a problems we have to use map reduce framework. In this frame work, we are separating the large work into smaller works that are running on a bunch of computers which is also well-known as cloud computing. In this way, we have to process large scale of data. They conducted their experiments of the population range up to 10^7. More population sizes not only to give better results but also require smaller amount generations. When we are dealing with difficult problems like jss, we have to improve already existing GA, by extremely scale up population with map reduce. So, the solution is to be finished in parallelized way with in measurable time.

According to Huang and Lin, for solving difficult problems, they are using 3 approaches. Genetic algorithm (L. Davis et al., 1991) 2. simulated annealing (L. Davis et al., 1989). Among of these, GA can be easily implemented because of its basic parallelism, that’s the reason it offers efficient solution for simplifying difficult problems. In GA, they represent the GAs in the form of string of symbols or linear chromosomes. And reproduce the process of natural selection, intersect and transformation with a population of chromosomes, as encouraged by Darwinian growth. Fitness of chromosomes is assessed derived from the excellence of the solutions they signify, and the fitter chromosomes have highest priority of endurance and copy. In this way, a good quality solution is likely to be provided after a number of generations.

2) Map reduce framework
It contains mainly two mechanisms: mappers and reducers (J.dean et al., 2010), which performs map and reduce functions by invoking a map defined by the user and reduce functions, correspondingly. Both two functions are situated on different machines. In map reduce framework, each mapper process a piece of the enter data in the type of key-value pairs, where each pair is sent some enter data to map function. the map function generates zero or more intermediate key_value pairs. These intermediate data is sorted and sent to the reducers. Once the intermediate data is dispatched, that is forced by one user defined component known as partitioner, the reducer process takes the input as intermediate data and produce the result as output key_value pairs. The data which is sent to the reducer is aggregated and sorted by the keys. So, MapReduce can control large size of input data (e.g., terabytes and even petabytes), it is feasible to encode the population of GAs as the input/output data.

3) Background
To assess the capacity of GAs with large populations, the Job Shop Scheduling Problem (JSSP) is chosen as the goal problem to be solve. It not only solves for practical way, but also for calculating in simple manner.
The aim is to process the J jobs on M different machines it takes more time. The time required for completion of job is to be reduced. Each job has sequence number of operations and these operations are process on different machines for a certain duration. Two constraints must be satisfied when performing an operation of duration \(d\) at time \(t\): (1) all precedent operations are completed before \(t\); (2) no other operations are scheduled to the required machine from \(t\) to \(t + d\).

JSS is much harder than travelling sales man problem. since there is no efficient solution for solving this type of problems. Our author provides some approaches that are already mentioned in introduction. Use of MapReduce allowed us to explore population sizes that are significantly larger than typical experiments, and revealed interest in tradeoffs between population sizes and number of generations (S. Reddi et al., 1972).

4) Algorithms used

Here we have different types of algorithms how to implement that algorithms on map reduce framework.

1. Mapper: fitness evaluation
2. Partitioner
3. Reducer: selection and reproduction

These algorithms are added in appendix

5) Experiments

The first experiment shows how the population size is affected in GA. Here the first experiment was run on bunch provided by Google and managed by IBM, shared among a few Universities as part of NSF are CLuE (Cluster Exploratory) Program and the Google/IBM Academic Cloud Computing. The cluster contains 414 physical nodes. Each node has two single-core processors, 4 GB memory, and two 400 GB hard drives (T. White et al., 2010). The second experiment shows effects of the cluster size.

E. A Genetic Algorithm by Hybrid Approach for MapReduce

1) Background

Master-slave genetic algorithm with single population: This is called as Global parallel genetic algorithm. In which one master node or one population is distributed among numerous processors (Golub Marin, Budin Leo, 2011). In this evaluation for fitness have been distributed in many slaves (Zec Amer et al., 2012).

In figure-2 master holds the population and executes genetic algorithm operation as well as distributes it among the slaves, Slaves evaluate fitness of individual (Zec Amer et al., 2012).

Multiple population genetic algorithms: It consists of several subpopulations which replace individual infrequently. This process is called migration. It is most tedious to control because of bad understanding of migration. So the exchange of individual known as migration. And controlled by different parameters. Figure-3 describes simple and full GA as well as each individual communication with other (Zec Amer et al., 2012).

![Figure 2: Master slave genetic algorithm (Zec Amer et al., 2012).](image1)

Fine-grained Genetic algorithm: Paper shows the third type of genetic algorithm is fine-grained genetic algorithm. In this each individual is in the shape of single rectangular grid as a special-structured population. As individual grid it has a single processor as per grid so that evaluation of fitness performed simultaneously for all individual. Then it transferred to SIMD computers, it execute particular instruction on all processors (Zec Amer et al., 2012).

![Figure 3: Multiple population genetic algorithm (Zec Amer et al., 2012).](image2)

Hierarchical hybrid genetic algorithm or Hybrid GA: It is the combination of master-slave GA and Fine-grained GA with all features of both. This algorithm parallelize multiple demes with single population and parallel GA at lower level (Zec Amer et al., 2012), this is more beneficial than any of above which is combined with map-reduce (Zec Amer et al., 2012).

2) Research Result

To implement Hybrid genetic algorithm need to use parallel algorithm. For that identify the serial elements. The main aim of this is to select individual as a serial component to perform different task with next individual with respect to corresponding operator. So that it will reduce their amount as well. So GA use distribution of individual in map task and at some point synchronizes all individual to get global optima. For this it uses map reduce technology. With help of map reduce in Genetic algorithm so with map reduce grouping and
sorting takes place with map and reduce phases (Zec Amer et al., 2012).

3) Map phase (Appendix-E-Fig-3)

At first start with map task in which assign each map task starts with individual for initial population. After that evaluation and selection is performed then it moves to the next step for iteration. Iteration will takes place until it finishes its population and the implementation of GA, its execution is fully parallel just because of mutual execution of map tasks. At the end of each step it is necessary to serialize map tasks. So with the map reduce task each map task later have the entry point to the reduce task. So need to take care of initial population (Zec Amer et al., 2012). In this paper each map task spending time for generating individual and large spending population. For this we choose uniform probability distribution for randomizing individual (Zec Amer et al., 2012). In this the mapper emits the best individual from last generation. It may have possibility of generating undiversified individual so it may have separate population redundant in every map task.

4) Reduce phase

As in reduce task of map reduce algorithm takes individual from last generation to reduce task. But in hybrid implementation of genetic algorithm to reduce the task it takes best individual from every generation then that of the best from the final one. For this reduce task will perform one more iteration of genetic algorithm. In this it will start with that task whose population is best one out of the map task (Zec Amer et al., 2012). Appendix-E shows the setup() as well as reduce() phase of hybrid implementation if genetic algorithm. In this the reducer chooses the best one out of the different mapper.

IV. REFLECTION

A. Scaling Simple and Compact Genetic Algorithms using MapReduce

In recent years, researchers have been trying to model new systems where GA can be parallelized using MapReduce programming model to solve large scale problems (as mentioned earlier) with minimal computational time and with effective solutions. Hadoop is an open source implementation of Google’s MapReduce and it has become the de-facto standard for this technology. Therefore, Hadoop is chosen as the candidate solution for parallelization of GAs using MapReduce. The experimental results of the research works show that more effective results can be obtained with reduced computational time by adding more computing resources to the cluster or cloud environment. There are some new areas of research has been undertaken like software testing using parallel GA. More research works are required to come up with suitable candidate solutions for this type of problem.

B. A MapReduce Based Hybrid Genetic Algorithm Using Island Approach for Solving Time Dependent Vehicle Routing Problem

Large-scale VRP, where thousands of client nodes exist, is impossible to run on a single computing device whereas this problem can easily be solved within minimum time and with maximum resource utilization in a distributed computing environment. It was studied from the experiments conducted by Kondekar et al., that the performance of GA is excellent. The time taken in computation of the GA in a MapReduce based distributed computing environment, verifies that the implementation can be devised to solve large-scale time dependent vehicle routing problem. According to Kondekar et al., even more large-scale problem can be executed and solved by adding more resources without any change in the algorithm design and implementation.

C. A Parallel Genetic Algorithm Based on Hadoop MapReduce for the Automatic Generation of JUnit Test Suites

According to the experimental results obtained from Geronimo et al. work, PGA reduces the computational time by 50%. The concept of parallel SBST is very new and the future research work on this area can be extended in several directions. A practical verification on the basis of more experiments are required to judge the strength or weakness of the proposed system. In addition, the scalability of the proposed system is to be verified with various GA settings, number of Map functions and with even more larger cluster. In future, the Hadoop MapReduce system can be examined with a cloud computing environment and with graphics cards apart from standard cluster environment (Geronimo et al., 2012).

D. Scaling Populations of a Genetic Algorithm for Job Shop Scheduling Problems using MapReduce

GA is one of the solutions for solving hard problems in jssp by using mapreduce framework. In map reduce framework we have 2 functions mapper and reducer. We have to compute population sizes up to 10^7. Fewer generations is more benefit to reduce the overhead of map reduce because of each map reduce job there exists certain initialization and overhead that requires fewer generations. Hence fewer generations reduce overall complexity of the mapreduce overhead.

E. A Genetic Algorithm by Hybrid Approach for MapReduce

By doing this research paper I come to know how the map reduce algorithm works with genetic algorithm and what are the different types of genetic algorithm as well as how hybrid genetic algorithm is used with map reduce function. Using GA with map reduce is most useful technique then others.
V. CONCLUSION

Genetic Algorithm (GA) can be easily parallelized due to its inherent parallel nature. Several phases of GA such as population generation, fitness evaluation, selection etc. can be run in parallel and thus, large-scale problems like huge data-intensive problems, vehicular routing problem, job dispatching, software testing etc. can be easily solved using large clusters of computing nodes in a distributed environment. These problems require large number of computational resources such as processors (to process large data), memories (to temporarily buffer large intermediate and final output data), storage disks (to store final large output data). It is impossible to solve such types of problems using single computing device. However, the infrastructure needed to build such huge clusters is expensive and the maintenance and management cost is also higher for such massively parallel and distributed computing systems. However, the recent use of inexpensive commodity computing devices to form large cluster has significantly reduced the infrastructural cost. Apart from the hardware cost involved in such clusters, the resource virtualization used in Cloud Computing environment has also minimized the maintenance and management cost for IT infrastructure. MapReduce programming model has proved its efficiency in solving large-scale problems in a parallel and distributed environment. MapReduce has demonstrated its excellent performance in various real time and large data-intensive applications such as Google initiated applications, Hadoop Distributed Framework etc.

In this paper we explored the various research works conducted for parallelizing Genetic Algorithms (GAs) using MapReduce programming model. We started with parallelization of simple and compact GAs using MapReduce in Section III-A. A MapReduce based hybrid Genetic Algorithm using island approach for solving time dependent vehicle routing problem was presented in Section III-B. In Section III-C, A parallel Genetic Algorithm based on Hadoop MapReduce for the automatic generation of JUnit test suites is discussed. In Section III-D, scaling populations of a Genetic Algorithm for Job Shop Scheduling problems using MapReduce is explained. A Genetic Algorithm by Hybrid Approach for MapReduce was explored in Section III-E. The reflection upon all the individual areas of research work is presented in Section IV.

REFERENCES


APPENDICES

APPENDIX-A (Algorithms) (Verma et al., 2009)

Algorithm 1 Map phase of each iteration of the GA

```
MAP(key, value):
1: individual ← INDIVIDUALREPRESENTATION(key)
2: fitness ← CALCULATEFITNESS(individual)
3: EMIT (individual, fitness)
4: {Keep track of the current best}
5: if fitness > max then
6:   max ← fitness
7:   maxInd ← individual
8: end if
9: if all individuals have been processed then
10:   Write best individual to global file in DFS
11: end if
```

Algorithm 2 Random partitioner for GA

```
GETPARTITION(key, value, numReducers):
1: return RANDOMINT(0, numReducers - 1)
```

Algorithm 3 Reduce phase of each iteration of the GA

```
REDUCE(key, values):
1: while values.hasNext() do
2:   individual ← INDIVIDUALREPRESENTATION(key)
3:   fitness ← values.getValue()
4:   if processed < Size then
5:     {Wait for individuals to join in the tournament and put them for the last rounds}
6:     turnArray[Size + processed][Size] ← individual
7:   else
8:     {Conduct tournament over past window}
9:     SELECTIONANDCROSSOVER()
10: end if
11: processed ← processed + 1
12: if all individuals have been processed then
13:   for k ← 1 to tournamentSize do
14:     SELECTIONANDCROSSOVER()
15:   end for
16: end while
17: SELECTIONANDCROSSOVER:
18: crossArray[processed][Size] ← TURN(turnArray)
19: if (processed + Size) % Size = 0 then
20:   newIndividuals ← CROSSOVER(crossArray)
21: for individual in newIndividuals do
22:   EMIT (individual, dummyFitness)
23: end for
```

Algorithm 4 Map phase of each iteration of the CGA

```
MAP(key, value):
1: splitNo ← key
2: probSplitArray ← value
3: EMIT(splitNo, [0, probSplitArray])
4: for k ← 1 to tournamentSize do
5:   SELECTIONANDCROSSOVER()
6:   processed ← processed + 1
7:   individual ←
8:   ones ← 0
9:   for prob in probSplitArray do
10:      if RANDOM(0,1) < prob then
11:         individual ← 1
12:      else
13:         individual ← 0
14:      end if
15:      EMIT(splitNo, [k, individual])
16:   WRITE TO DFS(k, ones)
17: end for
```

Algorithm 5 Reduce phase of each iteration of the CGA

```
REDUCE:
1: Initialize:
2: ALLOCATEANDINITIALIZE(onesArray[tournamentSize])
3: winner ← -1
4: loser ← -1
5: processed ← 0
6: n ← 0
7: for k ← 1 to tournamentSize do
8:   for r ← 1 to numReducers do
9:     if One(r) > winner then
10:        winnerIndex ← k
11:      else
12:         if One(r) < loser then
13:            loserIndex ← k
14:         end if
15:      end if
16:   end for
17: end for
18: Reduce:
19: while values.hasNext() do
20:   splitNo ← key
21:   value[processed] ← values.getValue()
22:   processed ← processed + 1
23: end while
24: for prob in value[0] do
25:   if value[winner],bit[n] > value[loser],bit[n] then
26:     if value[winner],bit[n] = 1 then
27:       newProbSplit[n] ← value[0] + 1/population
28:      else
29:       newProbSplit[n] ← value[0] - 1/population
30:    end if
31: end if
32: EMIT(splitNo, [0, newProbSplit])
33: end for
```
APPENDIX-B (Results) (Verma et al., 2009)

Figure 1: (Results-1) Results obtained using Hadoop when implementing a simple genetic algorithm for solving the OneMAX problem

(a) Convergence of GA for $10^4$ variables.
(b) Scalability of GA with constant load per node.
(c) Scalability of GA for 50,000 variables with increasing number of mappers.
(d) Scalability of GA with increasing number of variables.
APPENDIX-C

Figure 2: (Result-2) Results obtained using Hadoop when implementing the compact genetic algorithm

(a) Scalability of compact genetic algorithm with constant load per node for the ONEMAX problem. (b) Scalability of compact genetic algorithm for ONE-MAX problem with increasing number of variables.
Figure 3: Flow chart for MapReduce based Hybrid Genetic Algorithm
APPENDIX-D

Figure 4: Architecture of the proposed PGA for the test suite generation
APPENDIX-E

Reduce phase of hybrid genetic algorithm (Zec Amer et al., 2012)

1: class MAPER
2: method SETUP()
3: begin
4: max := Numeric.MIN
5: population := init()
6: end
7: method MAP(key, value)
8: begin
9: for n<numOfGenerations do
10: begin
11: eval(population)  //selection, mutation, crossing
12: population← new_generation(population)
13: n ← n + 1
14: end
15: max := best_individual(population)
16: EMIT( bits(max), fitness(max) )
17: end

Pseudo code for map phase of GA individual (Zec Amer et al., 2012).

1: class REDUCER
2: method SETUP()
3: begin
4: max := Numeric.MIN
5: max_binary_representation := null
6: end
7: method REDUCE (key binary_representation, value Iterator<fitness_value>)
8: begin
9: for fitness_value: Iterator.next() do
10: begin
11: if(fitness_value > max)
12: max ← fitness_value
13: max_binary_representation ← binary_representation
14: break
15: end
16: end