Genetic Algorithms with Mapreduce Runtimes

Fei Teng\textsuperscript{1}, Doga Tuncay\textsuperscript{2}

Indiana University Bloomington School of Informatics and Computing Department
CS PhD Candidate\textsuperscript{1}, Masters of CS Student\textsuperscript{2}

\{feiteng,dtuncay\}@indiana.edu

Abstract

Data-intensive Computing has played a key role in processing vast volumes of data exploiting massive parallelism. Parallel computing frameworks have proven that terabytes of data can be routinely processed. Mapreduce is a parallel programming model and associated implementation founded by Google, which is one of the leading companies in IT. Genetic Algorithms have increasingly applied on parallel computing to large scale problems. Since, GAs have the parallelism in their nature, they can be easily applied on Parallel Runtimes such as MPI, Hadoop Runtime and Twister\textsuperscript{9}. Our researches have shown that, Genetic Algorithms were very successful and can be modeled on MPI (Message Passing Interface) and Mapreduce Models. We will explain the nature of Genetic Algorithms, how they are efficient to use Parallel computing networks and how can they be applied MapReduce Model. In this project, we will be applying Genetic Algorithms, primarily the basic algorithm Onemax Problem on Hadoop Mapreduce Framework and the Twister Iterative Mapreduce Framework and do the performance Analysis. Our expectation from this project, to prove that the GAs will perform better performance on Iterative Model from its nature and Twister’s computing model.

Keywords: Genetic Algorithms, Onemax Problem, Mapreduce, Hadoop, Twister, Parallel Computing, Data-intensive Computing

1. Introduction

Genetic Algorithms are increasingly being used for to solve large scale problems, such as clustering[1], non-linear optimization [2] and job scheduling [3]. The parallel nature of Genetic Algorithms makes them an optimal base for parallelization. We will call parallel genetic algorithms as PGAs and PGAs are not the only parallel version of serial GAs. In fact, they actually support ideal base for having a parallel algorithm that behaves better than the sum of the separate behaviors of its component sub-algorithms[4].
An observation has shown us that the PGAs’ structured populations, either in a form of set of islands or a diffusion grid lead to superior numerical performance which is a benefit for the system. Therefore, many parallel computing scientists do not use a parallel machine to run structure-population because of its parallel model and they still are able to get better results than serial GAs. [5] There is another way to parallelize the system which is the hardware parallelization a way of speeding up the execution of the algorithm in hardware manners. Hence, a structured population model can be defined and implemented on any parallel machine. There are some researches on this topic that are showing a ring of panmictic GAs on MIMD/SIMD computers.

Beyond parallel machines, we are able to implement GA’s on Mapreduce Runtimes to improve the parallel behavior of the GAs. A research has shown that the implementation of simple and compact GAs of Hadoop demonstrate the convergence and scalability up to $10^5$ to $10^8$ variable problems respectively [6]. Therefore, the main contributions of this paper to demonstrate that genetic algorithms can be transformed into the map and reduce primitives and can be implemented to a Mapreduce program with scalable results for the large problem size. We also support the idea of Twister Iterative Mapreduce Runtime will have a better performance, because of its nature supports the design of evolutionary computing algorithms. Therefore, we will implement both Hadoop and Twister Runtime, by using the basic Genetic Algorithm Onemax and do the performance analysis with the comparison of structures, designs of the Mapreduce algorithms with an environment-independent approach.

2. Onemax Problem Statement

Onemax problem is a basic optimization problem that aims to maximize the number of ones in a bit string. The bit strings are used to show genes with fixed length strings.

In Onemax problem, we formally describe this problem as finding a string

$$\vec{x} = \{x_1, x_2, ..., x_n\} \text{ and } x_i \in \{0,1\}$$

which maximizes the following sum equation,

$$f(\vec{x}) = \sum_{i=1}^{n} (x_i)$$

where x is a gene bit string with a fixed length n = |x|. We consider that there is no noise involved with our gene encoding.

Onemax represents an optimization problem with independent variables and it is widely used as benchmark to measure the effectiveness and efficiency of a class of genetic algorithms.

3. Architecture design
To solve the Onemax problem described in section 1, we need efficient computing architecture for Mapreduce genetic algorithm. However, before put genetic algorithm under the Mapreduce context, we must have a thorough understanding about the genetic algorithm for Onemax problem. For this sake, we will first give out the design of serial Onemax GA and then present how to program the same GA in Mapreduce context to tackle large-scale Onemax application.

### 3.1 Serial Onemax genetic algorithm

- **Gene representative**

  It’s not hard to represent a gene for Onemax problem. Traditionally, a gene in Onemax population is represented as a bit string. For instance, if we set the problem size as 10, which means the maximum length of a gene is 10, then we can have a legal gene representative as a binary number with length of 10 like this <1101110010>.

- **Genetic operators**

  We use tournament selection[7] without replacement as the select operator. The tournament size is set as 5 initially. We will run experiments to tune the parameter to make it optimal for our problem size. [6] shows that uniform crossover[8] is well applicable for Onemax problem with binary gene representative. So we choose uniform crossover as well. At last, to guarantee a fast algorithm converging, we don’t need a large mutation rate. Currently, we set mutation rate as 0.01.

- **Fitness function**

  The fitness function is explicitly given in the problem statement. And the fitness value is how many one appears in the gene representative.

- **Serial GA flow chart**

  Based on what’s mentioned above, we give out the GA flow chart as Figure 1.

  For a population with size N, we conduct N select operator and use uniform crossover to derive N new better offspring in terms of fitness value. If we find that population in generation K is converged and meets the stop criterion, then we output the final optimal gene from the final population.
3.2 Mapreduce Onemax genetic algorithm

3.2.1 Basic Mapreduce GA design

Based on the serial GA, we can naturally come up with the Mapper and Reducer design. At first, we will create a new value type for the gene representative mentioned above. Then we can have the <key, value> pair in this form, <GeneUid, GeneValue>. What’s worthy of attention here is that we don’t simply choose the binary gene representative as the key, but to generate an Uid for each gene. The reason underlying this design is that with the evolution of the whole population, most of the genes will have the same representative because all genes have the pressure to converge on the single optimal solution for Onemax problem. Thus, if we trivially use binary
representative as the key, almost all the genes in the population will be assigned to one single reducer (reducer receives its input from mappers based on the key). It’s really undesirable design with respect to load balance. The uid of a gene is randomly generated and if we choose a big enough random range, the probability of two genes having the same uid is very small. In this way, we can archive nearly even load balance.

The reducer will be in charge of conducting the select and crossover operation. After each reducer emits all new offspring to combiner, the combiner will partition the new population into M parts for M mappers to consume. In this way, the iterative nature of GA is embodied well in Mapreduce context. What’s more, twister explicitly supports iterative Mapreduce. So implementation complexity is reduced a lot with the twister iterative API.

From the description above, Figure 2 presents the Mapreduce GA design.

![Figure 2. Mapreduce GA design](image)

### 3.2.2 Optimization

Because we aim to resolve large scale Onemax problem, the size of population and the length of a gene representative will be much larger than what serial GA can handle. For example, we want to tackle a Onemax problem with a $10^5$ size, then the corresponding population size will be $10^4$ (the relation between problem size and population size please reference [7]). In this way, each individual gene will need 100KB memory space by using plain text input format. The whole
population will occupy 100K*10,000=1GB space. Then the communication cost will be the potential bottleneck.

Thus, we plan to use binary file format to replace traditional text input file format. This will reduce the file size to 12.5% of the plain text format. For the instance above, only 125MB space is enough for the whole population. With the input file format, we can tackle problem with a size of 8 times larger with the same computing hardware.

4. Implementation timeline

Table 1 gives the expected timeline of our Mapreduce GA project.

<table>
<thead>
<tr>
<th>Event</th>
<th>Time</th>
<th>Status</th>
<th>Person in charge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detailed design</td>
<td>Oct.1 - Oct. 31</td>
<td>Completed</td>
<td>Both</td>
</tr>
<tr>
<td>Implementation on Hadoop</td>
<td>Nov.1 - Nov. 30</td>
<td>In progress</td>
<td>Doga Tuncay</td>
</tr>
<tr>
<td>Implementation on Twister</td>
<td>Nov.1 - Nov. 30</td>
<td>In progress</td>
<td>Fei Teng</td>
</tr>
<tr>
<td>Experiments &amp; Analysis</td>
<td>Dec.1 – Dec.5</td>
<td>Not yet started</td>
<td>Both</td>
</tr>
</tbody>
</table>

Table 1. project timeline

5. Validation method

The Onemax problem is very easy to validate because the optimal solution is obvious. If our Mapreduce GA can give out the size of the problem, e.g, the length of the gene representative, then our results can be declared as correct. Besides, we will also do a performance validation for Mapreduce GA with some parallelism metric like speed-up to prove that Mapreduce is not only correct but efficient. Some performance charts based on different problem scale will be given as well to demonstrate the scalability of Mapreduce GA.

REFERENCES:


