SOUTHEAST EUROPE JOURNAL OF SOFT COMPUTING
Available online at www.scjournal.com.ba

Parallelization of genetic algorithms using Hadoop Map/Reduce

Dino Kečo [a], Abdulhamit Subasi [b]

[a] Faculty of Engineering, International Burch University, 71000 Sarajevo, dino.keco@gmail.com

[b] Faculty of Engineering, International Burch University, 71000 Sarajevo, asubasi@ibu.edu.ba

***Abstract*—In this paper we present parallel implementation of genetic algorithm using map/reduce programming paradigm. Hadoop implementation of map/reduce library is used for this purpose. We compare our implementation with implementation presented in [1]. These two implementations are compared in solving One Max (Bit counting) problem. The comparison criteria between implementations are fitness convergence, quality of final solution, algorithm scalability, and cloud resource utilization. Our model for parallelization of genetic algorithm shows better performances and fitness convergence than model presented in [1], but our model has lower quality of solution because of species problem.**

Keywords— genetic algorithm, parallelization, map/reduce, hadoop.

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.

There are couple of techniques to implement parallel genetic algorithm and two most popular are [7]:

- Cluster nodes work on same population,
- Each node in cluster has own population.

First implementation of parallel genetic algorithm is presented in [1]. In this paper we are presenting second implementation and results of comparison between these two implementations. For parallelization of genetic algorithm Hadoop Map/Reduce library is used.

This paper makes the following research contributions:

- New model for parallelization of genetic algorithm,
- Implementation of that model with Hadoop Map/Reduce library,
- Comparison with implementation presented in [1].

Section 2 provides more detailed explanation of Map/Reduce model. In section 3 we describe how genetic algorithm could be implemented inside Map/Reduce model. Section 4 provides implementation details with focus on solving of One Max. We present scaling measurements of these two implementations on Hadoop Map/Reduce cluster in section 5 and we conclude in section 6.

MAP REDUCE MODEL

Map/Reduce model is first time proposed by Google [3] in 2004 and it is inspired by functional languages like List. Map/Reduce model represent simplified way of parallelization for all programs which are written in Map/Reduce spirit. In Map/Reduce programming paradigm, the basic unit of information is a (key; value) pair where each key and each value are binary strings. The input to Map/Reduce algorithm is set of (key; value) pairs. Operations on a set of pairs occur in three stages: the map stage, the shuffle stage and the reduce stage 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.[9]

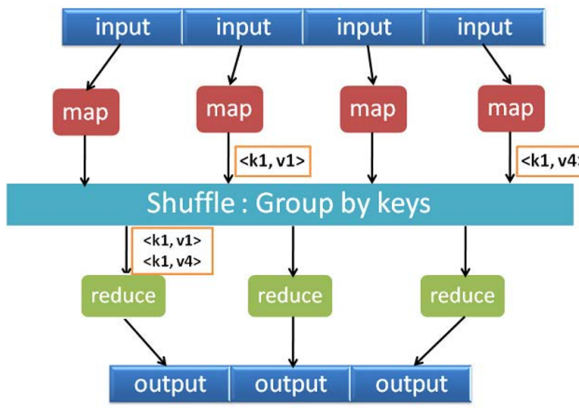


Figure 1: Operation phases in Map/Reduce programming model [8]

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. [9]

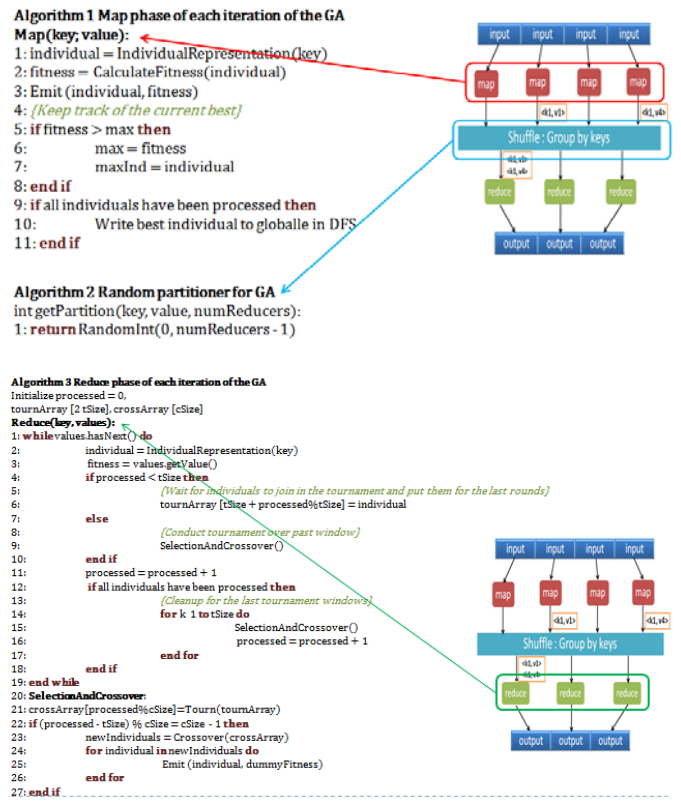
In the reduce stage, the reducer takes all of the values associated with a single key k, and outputs a multi set of (key; value) pairs with the same key, k. This highlights one of the sequential aspects of Map/Reduce computation: all of the maps need to finish before the reduce stage can begin. [9]

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 [2].

HOW PARALLELIZATION OF GA FIT INTO MAP/REDUCE MODEL

In this section we present two models of parallel genetic algorithms written in Map/Reduce programming spirit. First model is presented in [1] and the second model is built as part of this research. First model uses one map reduce phase for one generation of genetic algorithm, and for each generation new map reduce phase is executed, while second model uses only one map reduce phase for all generations of genetic algorithm.

First model and its pseudo code are presented on figure 2. On the figure it’s shown in which phase each part of pseudo code is executed.

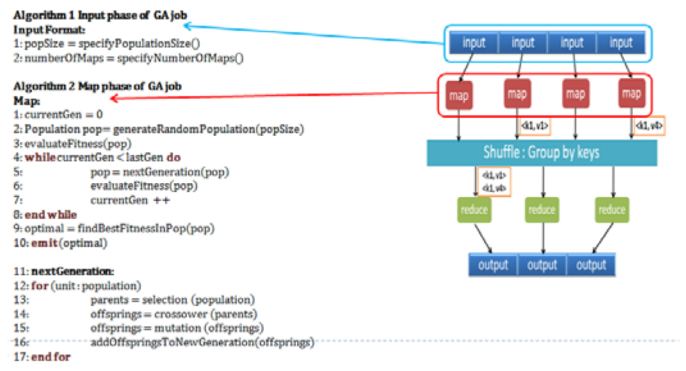


```
Algorithm 1 Map phase of each iteration of the GA
Map(key, value):
1: individual = IndividualRepresentation(key)
2: fitness = CalculateFitness(individual)
3: Emit (individual, fitness)
4: {Keep track of the current best}
5: if fitness > max then
6:         max = fitness
7:         maxInd = individual
8: end if
9: if all individuals have been processed then
10:        Write best individual to globalle in DFS
11: end if

Algorithm 2 Random partitioner for GA
int getPartition(key, value, numReducers):
1: return RandomInt(0, numReducers - 1)

Algorithm 3 Reduce phase of each iteration of the GA
Initialize processed = 0,
tournArray [2 tSize], crossArray [cSize]
Reduce(key, values):
1: while values.hasNext() do
2:         individual = IndividualRepresentation(key)
3:         fitness = values.getValue()
4:         if processed < tSize then
5:                 {Wait for individuals to join in the tournament and put them for the last rounds}
6:                 tournArray [tSize + processed%tSize] = 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:                {Cleanup for the last tournament windows}
14:                for k 1 to tSize do
15:                        SelectionAndCrossover()
16:                        processed = processed + 1
17:                end for
18:        end if
19: end while
20: SelectionAndCrossover:
21: crossArray[processed%cSize]=Tourn(tournArray)
22: if (processed - tSize) % cSize = cSize - 1 then
23:        newIndividuals = Crossover(crossArray)
24:        for individual in newIndividuals do
25:                Emit (individual, dummyFitness)
26:        end for
27: end if
```

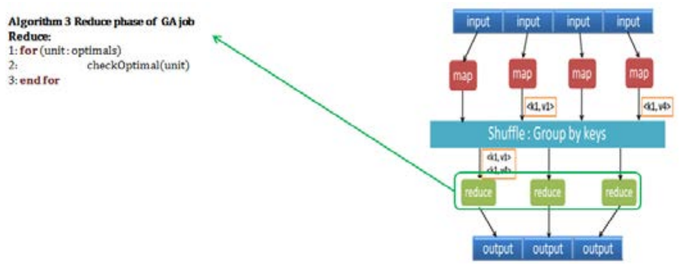
Figure 2: Pseudo code of GA model which uses one population for all nodes in cluster

Most of details related to this model are shown by pseudo code. One important detail related to this model is that it uses chain of map reduce actions for each generation of genetic algorithm. HDFS (Hadoop Distributed File System) is used as a data transfer between each generation of GA. [1] This model have big IO footprint because after each generation of GA full population is saved to HDFS, and this cause big degradation of performances.

Second model and its pseudo code are shown on figure 3. On the figure it's shown in which phase each part of pseudo code is executed.



```
Algorithm 1 Input phase of GA job
Input Format:
1: popSize = specifyPopulationSize()
2: numberOfMaps = specifyNumberOfMaps()

Algorithm 2 Map phase of GA job
Map:
1: currentGen = 0
2: Population pop= generateRandomPopulation(popSize)
3: evaluateFitness(pop)
4: while currentGen < lastGen do
5:         pop = nextGeneration(pop)
6:         evaluateFitness(pop)
7:         currentGen ++
8: end while
9: optimal = findBestFitnessInPop(pop)
10: emit(optimal)

11: nextGeneration:
12: for (unit : population)
13:        parents = selection (population)
14:        offsprings = crossower (parents)
15:        offsprings = mutation (offsprings)
16:        addOffspringsToNewGeneration(offsprings)
17: end for
```



```
Algorithm 3 Reduce phase of GA job
Reduce:
1: for (unit : optimals)
2:         checkOptimal(unit)
3: end for
```

Figure 3: Pseudo code of GA model which uses different population for each node in cluster

As shown in pseudo code for this model most of processing has been moved from reduce phase to map phase. This change reduces amount of IO footprint because all processing data is kept in memory instead of HDFS, but drawback of this is that we have different population for each node which leads to species problem in genetic algorithm.

IMPLEMENTATION OF PARALLEL GA OVER MAP/REDUCE FRAMEWORK FOR ONE MAX PROBLEM

For testing of genetic algorithm we have been using One Max problem because same problem was solved in [1]. Our implementation of genetic algorithm supports any type of function to be optimized because function class is plug-able using configuration of genetic algorithm.

Class which implements One Max function for distributed genetic algorithm model is

```
public class OneMax extends AbstractFitnessFunction {
        public OneMax(List<VariableRange> variableRanges) {
                super(variableRanges);
        }
        @Override
        public Double evaluate(List<Double> args) {
                if(args!=null && args.size() < 1 ){
                        throw new
IllegalArgumentException("OneMax is multivariable function,
at least one argument is required!");
                }
                this.args = args;
                Double value = 0.0;
                for(Double arg : args){
                        value += arg;
                }
                return value;

        }
}
```

This function is configured while map reduce job is executed using ga.optimizing.function configuration property.

SCALING MEASUREMENTS ON HADOOP CLUSTER

In this section we are testing performances of two different implementations of genetic algorithms. For those purposes we have been using 10 nodes cluster (i7 - 4 cores 2.6 GHz, 4GB

DDR3 RAM, 300GB HDD) with CentOS 5.6 operating system and Hadoop CDH3u0 distribution. We have performed two tests of both models are compared results

- Convergence of genetic algorithm with constant number of map reduce tasks,
- Scalability of genetic algorithm with constant load per node in cluster.

Convergence of genetic algorithm with constant number of map reduce tasks

With this test we have track best fitness in population of One Max problem over iterations of genetic algorithm and compared results from both models. Results of comparison are presented on figure 4.

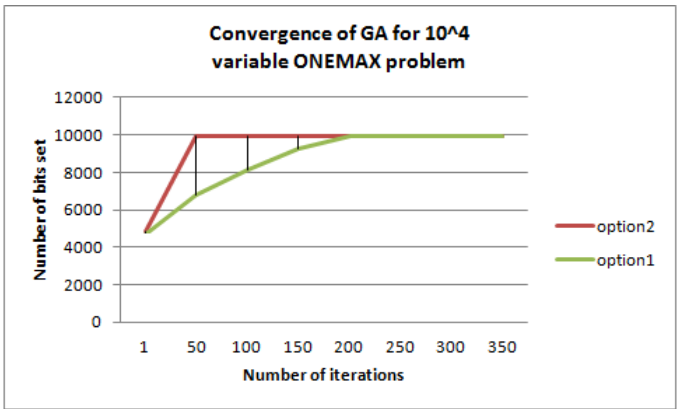


Figure 4: Comparison of two models of parallel GA from solution convergence aspect; option 1 – All nodes uses same population; option 2 – each node has its own population

In this test parameters of genetic algorithm are:

- crossover probability = 0.7
- mutation probability = 0.01
- population size = 5000
- number of mappers/reducers = 20

As results show option 2 has better convergence of fitness because it has multiple mappers, which are working on different populations, which causes that solution is found much earlier.

Scalability of genetic algorithm with constant load per node in cluster

In second test we compared scalability of two models described in this paper. Results of comparison are presented on figure 5.

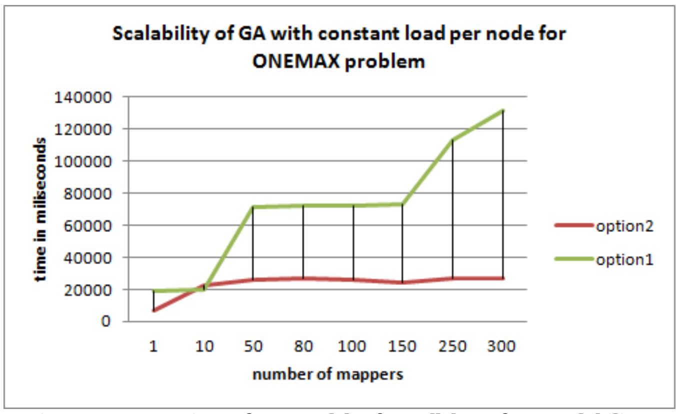


Figure 5: Comparison of two models of parallel GA from scalability aspect; option 1 – All nodes uses same population; option 2 – each node has its own population

In second test parameters of genetic algorithm are:

- crossover probability = 0.7
- mutation probability = 0.01
- population size = 5000
- number of variables for One Max problem = 10000
- number of iterations = 200

As presented on figure 5 option 2 is much faster than option 1 because IO footprint is reduced because data is not written to HDFS. Option 1 and 2 can scale to infinity by adding more hardware resources into cloud.

Acknowledgments of people, grants, funds, etc. should be placed in a separate section before the reference list. The names of funding organizations should be written in full. DO NOT place acknowledgment on the first page of your paper or as a footnote.

Conclusion and Future Work

Both of models which are described in this paper can easily scale and those models could be used for solving any complex problem just by adding more hardware resources. First model which is presented in [1] is slower but doesn't have species problem like other model. In future work both models should be used for solving different problems, like TSP (Traveling Salesman Problem). Some other parts of map reduce model, like combiner, are not used in these models of GA but those parts could be used to improve models even more.

REFERENCES

Both of models which are described in this paper can easily scale and those models could be used for solving any complex problem just by adding more hardware resources. First model which is presented in [1] is slower but doesn't have species problem like other model. In future work both models should be used for solving different problems, like TSP (Traveling Salesman Problem). Some other parts of map reduce model,

like combiner, are not used in these models of GA but those parts could be used to improve models even more

[1]Scaling Genetic Algorithms using MapReduce - Abhishek Verma, XavierLlor'a, David E. Goldberg, Roy H. Campbell

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

[3]MapReduce: Simplified Data Processing on Large Clusters - Jeffrey Dean and Sanjay Ghemawat

[4]Scaling Populations of a Genetic Algorithm for Job Shop Scheduling Problems using MapReduce - Di-Wei Huang, Jimmy Lin

[5]Scheduling divisible MapReduce computations - J. Berlińska, M. Drozdowski

[6]MRPGA: An Extension of MapReduce for Parallelizing Genetic Algorithms - Chao Jin, Christian Vecchiola and Rajkumar Buyya

[7]Parallel Genetic Algorithms - R. Shonkwiler

[8]Map Reduce web page - http://hadoop.apache.org/mapreduce/

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

[10]Raghuraman, R., Penmetsa, A., Bradski, G., and Kozyrakis, C. Evaluating mapreduce for multi-core and multiprocessor systems. Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture.