Application of an Improved Diploid Genetic Algorithm for Optimizing Performance through Dynamic Load Balancing

A. SIMA UYAR, A. EMRE HARMANCI
Computer Engineering Department
Istanbul Technical University
Maslak, Istanbul TR80626
TURKEY
{etaner,harmanci}@cs.itu.edu.tr

Abstract: - The dynamic load balancing problem which can be defined as the effective redistribution of workload among the system processing units during execution time, is dynamic in nature where the load and the processing power of the system may change in time as units of work enter and leave the system and processing units are added to or removed from the processing pool. To address this problem, genetic algorithms are used in literature in different ways. In this study, promising results of applying an improved diploid genetic algorithm for load balancing a simulation of a network of processing units are reported.

Key-words: Load balancing, changing environments, evolutionary optimizaton, genetic algorithms, diploid genetic algorithms.

1 Introduction

Genetic algorithms have been applied to a diverse field of problems with promising results. While most of these mainly address stationary problems, there's another group where the problem is dynamic and is represented by a changing fitness function. This class of problems are characterized by a need for a mechanism to adapt to the change. Different characteristics of changing fitness functions can be exploited in different ways to obtain a near optimal solution.

One issue that plays an important role in distributed system performance is the effective distribution (scheduling) of workload (tasks or jobs) among the system processing units (PU) with the aim of improving system performance. Dynamic scheduling is often referred to as dynamic load balancing which can be defined as the redistribution of tasks among the PUs during execution time. The load balancing problem is dynamic in nature where the load and the processing power of the system may change in time as jobs enter and leave the system and processing units are added to or removed from the processing pool. To address this problem, new methods are explored with adaptability to the change as the main focus. Genetic algorithms have been used in different ways for dealing with the different aspects of the dynamic load balancing problem [1], [2], [3], [4], [5], [6]. In this study, a diploid genetic algorithm with an adaptive dominance mechanism for genotype to phenotype mapping, a meiosis-like process for reproduction and overlapping populations with replacement of individuals based on an aging mechanism is used. This algorithm will be referred to as *damGA* (**d**iploidy-**a**ging-**m**eiosis **G**enetic **A**lgorithm).

2 System Simulation

The system to be load balanced consists of a number of PUs connected over a network. One processor is dedicated to load balancing operations. *damGA*, which serves as the central load balancer, runs on this processor. Load information from all PUs are sent to the central processor when a change occurs. This information is used by *damGA* to find a better distribution of jobs on the PUs. When a more efficient load distribution is found by the *damGA* load balancer, all PUs in the system are notified to initiate the necessary job transfers. Task transfers among PUs cause overheads which penalize the performance value for that distribution in proportion to the number of task transfers needed and the sizes of the tasks to be transferred.

The load balancing algorithm works on an event driven simulation of the defined system. Events occur with interarrival times according to a Poisson distribution. There're four types of events in the system, namely the *new job event*, the *finished job event*, the *new PU event* and the *removed PU event.*

To simplify the simulation, some assumptions are made about the system. These assumptions are:

- All PUs in the system are equipped with the same type of resources with different capacities. All jobs may be migrated.
- If a new job arrives that will cause the current system

load to exceed the total capacity of the system, the job is refused.

- At the beginning of job execution, the average resource (CPU, I/O, Memory) requirements per unit time for each job are determined randomly. It is assumed that actual resource requirements do not deviate too much from the average values. The load value assigned to the job is a function of average requirements per unit time for all types of resources.
- The migration cost value assigned to a job is a function of the packing and unpacking loads at the host and target PUs respectively and the communication overheads as a result of job transfers over the network from host to target PUs.
- System is not initially empty and is always at least moderately loaded.

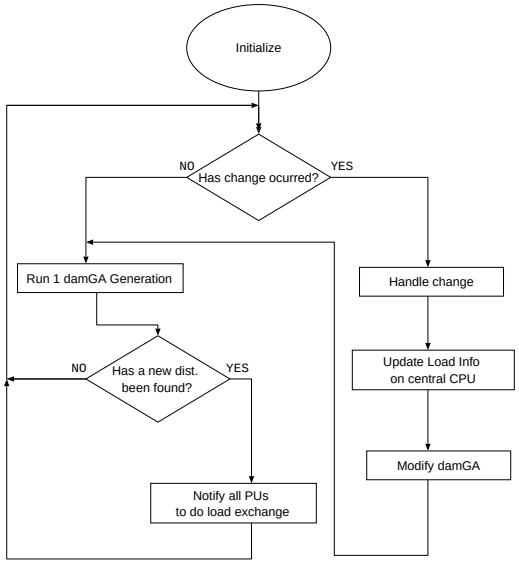

Fig. 1 The flowchart for load balancing simulation

The PUs in the system are represented by their *PU numbers*, *capacities* which are assigned randomly at the time they join the system and a value that shows their *current loads*. A job in the system is represented by its *job number*, its *average resource requirement per unit time* and the *overhead* it brings to migrate that job. In addition to these representational information, the central load balancer unit also keeps track of where each job is located and what each PU's current load is. This information is updated each time a change occurs in the system. The actual system load distribution is updated only if the distribution proposed by *damGA* will increase the system performance by a threshold percent. A predefined number of generations are required to pass until the solution candidate found by *damGA* is applied to the system. The general flow chart for the system execution is given in Fig 1.

3 *damGA* Applied to Load Balancing

The load balancer is implemented using *damGA* modified to an extent to fit the requirements of the load balancing system. The main features of the algorithm will be given in detail in the following sections. A more detailed explanation of the algorithm and performance comparisons can be found in [7], [8] and [9] by the same authors.

3.1 Representation

In *damGA*, each individual is represented with three strings, a fitness value and an age. Chromosome 1 and chromosome 2 are homologues and form the diploid genotype of the individual. The third string which is the phenotype, shows the characteristics that are expressed. In this implementation, each gene on the chromosome represents a job in the system and the alelle value of that gene shows on which PU that job is running. The chromosome length is not constant and changes as the number of jobs in the system decreases or increases. Each gene may take on a value from the set of possible alleles determined by the number of PUs present in the system. This allele value set is not constant and changes when PUs join or leave the system. A sample chromosome for four PUs and eight jobs is given in Fig. 2.

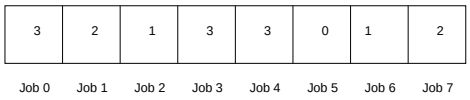

Fig. 2 A sample chromosome

3.2 Phenotype to Genotype Mapping

The phenotype of the individual is the set of characteristics that are expressed. The fitness is determined using the phenotype. This is a very important part of diploid genetic algorithms and there has been some research done most of which are explained in detail in [10] and [11]. In natural organisms the dominant allele is seen in the phenotype, so a mechanism to simulate this in artificial systems will be used. In *damGA*, a *domination matrix* is used in determining the phenotype of an individual. The number of columns in

the matrix is determined by the number of jobs in the system and the number of rows is determined by the number of PUs in the system. The pseudocode for calculating the domination matrix is given in Fig. 3.

```
for i=0 to MaxAllele
begin
  for j=0 to LastGenePos
  begin
    Val=0;
    for n=0 to LastIndividual
      if (individual[n].phenotype[j]=i)
        Val=Val+indiv[n].fitness;
    DM[i,j]=Val/TotalFitness;
  end;
end;
```

Fig. 3 Pseudocode for Domination Matrix Calculation

Each value in the matrix shows the domination factor of the corresponding allele in relation to the other alleles present on the chromosomes for that location. For example, assume that the [0,0] element of the matrix has a value of 0.82, the [1,0] element has value 0.18, [2,0] and the [3,0] elements have value 0.0. This means that in the next generation if an individual has *allele 2* or *allele 3* on either of its chromosomes at *locus* 0, the probability of that allele being expressed on the phenotype at that locus is 0.0, unless of course the individual has the same allele for that locus on both of its chromosomes in which case the corresponding phenotype value equals that of the genes. But if the individual has *allele 0* at *locus 0* on its first chromosome and *allele 1* at *locus 0* on its second chromosome or vice versa, the corresponding phenotype value becomes `0` with probability 0.82 and becomes `1` with probability 0.18. The domination matrix is recalculated at the end of each generation using the individuals in that population.

3.3 Fitness Evaluation

The fitness of an individual shows how balanced the load distribution depicted by the phenotype of that individual is. To calculate the fitness, the amount of load per unit capacity called the *unit load* (UL) is determined as in Eq. 1.

$$UL = \frac{Total\,System\,Load}{Total\,System\,Capacity} \quad (1)$$

For each PU, the amount of load for that PU under ideal conditions, which will be called *ideal load* (IL) is calculated. The ideal load for the ith PU which has a capacity of $Capacity_i$ is calculated as in Eq. 2.

$$IL_i = UL * Capacity_i \quad (2)$$

The *load imbalance* (LI) of a PU is the absolute value of the difference between the *actual load* (AL) of a PU and its *ideal load* (IL). The aim of the load balancer is to minimize the total load imbalance in the system. To normalize the total imbalance value, it is divided by the curent total load in the system. The normalized load imbalance is given by Eq. 3.

$$LI_N = \frac{\Sigma_i |AL_i - IL_i|}{Total\,System\,Load} \quad i = 0, 1, ..., NoOfPUs \quad (3)$$

In calculating the fitness of an individual the migration overhead is used as a penalty. The actual fitness value f_a of an individual is calculated as in Eq. 4.

$$f_a = \frac{1}{LI_N} \quad (4)$$

In this implementation the migration costs determined randomly for each job at start of job's execution is a real value in the [0,1] interval. The total penalty value for a distribution candidate is a function of the sum of all the migration costs for the jobs that will be transferred. When calculating the penalty value, in the worst case the new distribution candidate will require all the jobs to be transferred. Assuming there're n jobs in the system, the upper bound for the migration costs will be n. In the best case, no job transfers will be required, giving the lower bound for the migration costs as 0.0. The penalized fitness value f_p for an individual is calculated as in Eq. 5 where C_A is the actual migration costs and C_M is the upper bound on the migration costs.

$$f_p = f_a * \frac{C_M - C_A}{C_M} \quad (5)$$

3.4 Main Steps of *damGA*

The basic steps of the algorithm used (*damGA*) is given below and explained briefly in the following sections. Further details about the algorithm can be found in [7] by the same authors.

```
begin
initialize;
do
  reproduction;
  mutation;
  dom. map recalculation;
  next generation selection;
until stop;
end.
```

The *initialization* step is similar to the one in the simple genetic algorithm (SGA) [7]. Each of the genes on the two chromosomes of the individual is initialized randomly to have a value in the possible allele set. All the locations on the domination matrix is initialized to 0.5.

The *reproduction* phase consists of selection of the mating pairs, gamete formation through meiosis, pairing off and the actual mating phase to form the offspring. A roulette wheel selection mechanism is used

to determine the individuals which will go into reproduction. Gametes in natural, diploid organisms are the haploid reproductive cells. One gamete from each mating pair comes together to make up the diploid chromosome structure of the offspring. In most cases in nature, gamete formation is the result of a cell division process called meiosis. In this artificial implementation, each parent goes through a meiotic cell division phase seperately. Meiosis is implemented in three steps. In the first step a copy of each chromosome string is made during which errors may occur. The chromosome and its copy are called sister chromatids. At the end of this step, the individual has four haploid chromatids. In the second step crossing over may occur between non-sister chromatids. In this implementation, a two point cross-over approach is used. In the final step, after each mating parent completes its meiosis-like process, there are four gametes from each parent, ready to go into mating. Since in this implementation, each mating produces two offspring, two gametes from each parent are selected at random and each gamete from each parent goes to each one of the offspring.

The *mutation* operator is as defined in SGA with the modification that a mutation changes the value of a gene from one allele to another in the set of allowed alleles.

At the end of each generation, the new domination map is recalculated. This new domination map is used in the next generation to obtain the phenotypes of the individuals from their genotypes.

Offspring do not replace their parents. However since population size is kept constant, the new individuals that will survive into the next generation are determined using a fitness proportional method. The chosen individuals' age values are increased. At the end of each generation some individuals are replaced with new, randomly initialized individuals with a probability based on their ages.

4 Tests and Results

The above explained simulation of the system, with the *damGA* as the central load balancer, is run without a stopping criterion. Events occur with interarrival times distributed according to the Poisson distribution with λ. The simulation of the physical system is initialized with a default number of jobs and PUs present in the system. The parameters and default values for the system simulation are given in Table 1.

Table 1 Parameters of the system simulation

Default Number of Jobs	32
Default Number of PUs	5
Maximum Number of Jobs	1024
Maximum Number of PUs	128
Maximum Job Load	540
Maximum PU Capacity	4096
λ for Poisson Distribution	500
Prob. for New Job Event	0.64
Prob. for Removed Job Event	0.16
Prob. for New PU Event	0.1
Prob. for Removed PU Event	0.1
damGA Runs Before Update	50
Acceptable Perf. Improvement	30%

The *damGA* is initialized with values depending on the default system parameters. The *damGA* parameters and the initial values are given in Table 2. The interval to examine the system is chosen between generations 3938 and 9357 during which all four types of events occur. The changes that occur in the system during this interval and the current load and job information of the system at the time of change are given in Table 3.

Table 2 *damGA* parameters

Population Size	250
Initial Chrom. Length	32
Crossover Probability	0.9
Mutation Probability	0.009
Error Prob.in Meiosis	0.001
Initial Dom.Values	0.5
Initial Allele Set	{0,1,2,3,4}

Table 3 Change instances

Gnr.	Change	Jobs	PUs	Tot. Cap.	Tot. Load
3938	New Job	38	5	12251	10038
4437	New Job	39	5	12251	10296
4919	Rem. Job	38	5	12251	9930
5383	Rem. PU	38	4	9965	9930
5902*	New Job	38	4	9965	9930
6372*	New Job	38	4	9965	9930
6858*	New Job	38	4	9965	9930
7365	New Job	39	4	9965	9940
7870	Rem. PU	39	3	9147	9940
8394	New PU	39	4	11843	9940
8873	New Job	40	4	11843	9941
9357	New Job	41	4	11843	9993

The change entries for generations 5902, 6372 and 6858 are marked with an asterisk on Ttable 3. Even

though these are new job events, it should be noted that the number of jobs in the system stays the same. This happens because the new jobs that arrive will cause the system to be overloaded and thus they are refused entry into the system.

The results will be shown as two different set of plots showing system performance with and without load balancing respectively. System performance is determined using the fitness calculation approach without penalties on the actual physical load distribution. The first plot which can be seen in Fig. 4 shows the system performance when no load balancing is performed.

The plot of the system performance between generations 3938 and 9357 when load balancing with *damGA* is applied can be seen in Fig. 5. The system fitnesses given on the plot are the actual values calculated for the current load distribution on the physical system.

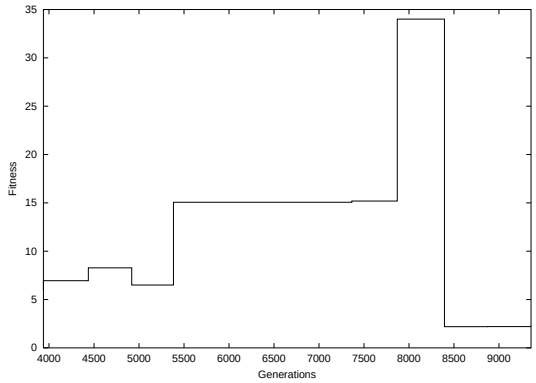

Fig. 4 Plot of system fitnesses without load balancing

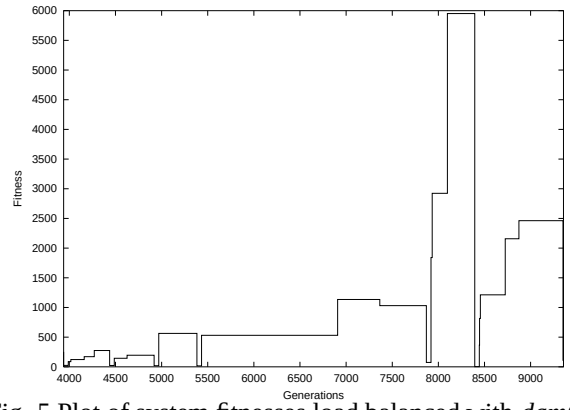

Fig. 5 Plot of system fitnesses load balanced with *damGA*

5 Conclusion

The results given in the previous section show that the application of the load balancing algorithm based on *damGA* increases system performance. A similar aproach can be found in [2] which deals with a similar system set up as the one used in this study where a central load balancing unit is run on a dedicated processing unit. In that study, a simple genetic algorithm (SGA) [7] is used for finding a suitable distribution of jobs on the PUs and the performance of the genetic load balancing approach which is called the *Genetic Central Task Assigner* (GCTA) is compared with three more classical dynamic load balancing approaches, namely *the Threshold Algorithm*, *the Central Algorithm* and the *Centex Algorithm* with promising results. When SGA is applied to the system used in the previous section with the same set of change instances and genetic algorithm parameters where applicable, the plot in Fig. 6 is obtained.

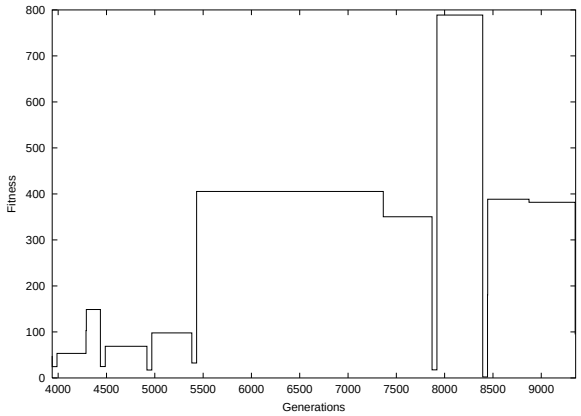

Fig. 6 Plot of system fitnesses load balanced with SGA

When compared with the system performance levels obtained when no load balancing is applied to the system (Fig. 4), the load balancing with SGA brings performance improvement. However as can be seen when comparing Fig. 5 and Fig. 6, the improvement in the levels of system performance is much greater when *damGA* is used as the load balancing algorithm with the current system configuration. These promising results encourage further study on the use of *damGA* for workload distribution optimization.

References:

[1] ALBERTO Carlos, PICO Gonzalez, WAINWRIGHT Roger L., “Dynamic Scheduling of Computer Tasks Using Genetic Algorithms”, in *Proceedings of The First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, Vol. 1, pp.829-833, 1994.

[2] BAUMGARTNER Joey, COOK Diane J., SHIRAZI Behrooz, “Genetic Solutions to the Load Balancing Problem”, in Proceedings of ICPP95 Workshop, pp. 72-78, 1995.

[3] FOGARTY Terence C., VAVAK Frank, CHENG Philip, “Use of the Genetic Algorithm for Load Balancing of Sugar Beet Presses”, in *Proceedings of the 6th International Conference on genetic Algorithms*, Morgan Kaufmann, pp, 617-624, 1995.

[4] FOGARTY Terence C., VAVAK Frank, CHENG Philip, “Load Balancing Application of the Genetic Algorithm in a Nonstationary Environment”, in Proceedings of Evolutionary Computing (AISB) Workshop, pp.224-233, 1995.

[5] MUNETOMO Masaharu, TAKAI Yoshiaki, SATO Yoshiharu, “A Genetic Approach to Dynamic Load Balancing in a Distributed Computing System”, in *Proceedings of The First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, Vol. 1, pp.1-529, 1994.

[6] MUNETOMO Masaharu, TAKAI Yoshiaki, SATO Yoshiharu, “Genetic Based Load Balancing: Implementation and Evaluation”, in *Proceedings of PPSN IV, International Conference on Evolutionary Computation, The 4th Conference on Parallel Problem Solving from Nature*, Lecture Notes in Computer Science 1141, Springer, 1996.

[7] Uyar A. Sima, Harmanci A. Emre, "Investigation of New Operators for a Diploid Genetic Algorithm", in *Proceedings of SPIE: Applications and Science of Neural Networks, Fuzzy Systems, and Evolutionary Computation II*, July 1999.

[8] Uyar A. Sima., Harmanci A. Emre, "A New Genotype to Phenotype Mapping Approach for Diploid Genetic Algorithms", in *Proceedings of the 15th International Symposium on Computer and Information Sciences*, October 2000.

[9] Uyar A. Sima., Harmanci A. Emre, “Preserving Diversity Through Diploidy and Meiosis for Improved Genetic Algorithm Performance in Dynamic Environments”, to appear in *Proceedings of the Second Biennial International Conference on Advances in Information Systems*, October 2002.

[10] Goldberg D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[11] Branke J. *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers.2002.