A Grid-oriented Genetic Algorithm *

J. Herrera[1], E. Huedo[2], R.S. Montero[1], and I.M. Llorente[1,2]

[1] Departamento de Arquitectura de Computadores y Automática, Universidad Complutense, 28040 Madrid, Spain.
[2] Laboratorio de Computación Avanzada, Simulación y Aplicaciones Telemáticas, Centro de Astrobiología (CSIC-INTA), 28850 Torrejón de Ardoz, Spain.

Abstract. Genetic algorithms (GAs) are stochastic search methods that have been successfully applied in many search, optimization, and machine learning problems. Their parallel counterpart (PGA, parallel genetic algorithms) offers many advantages over the traditional GAs, such as speed, ability to search on a larger search space, and less likely to run into a local optimum. With the advent of Grid computing, the computational power that can be deliver to the applications have substantially increased, and so PGAs can potentially benefit from this new Grid technologies. However, because of the dynamic and heterogeneous nature of Grid environments, the implementation and execution of PGAs in a Grid involve challenging issues. This paper discusses the distribution of a PGA across the Grid using the DRMAA standard API and the Grid *W*ay framework. The efficiency and reliability of this schema to solve the One Max problem is analyzed in a globus-based research testbed.

1 Introduction

Genetics algorithms are search algorithms inspired in natural selection and genetic mechanisms. GAs use historic information to find new search points and reach an optimal problem solution. In order to increase the speed and the efficiency of sequential GAs, several Parallel Genetic Algorithm (PGA) alternatives have been developed. PGAs have been successfully applied in previous works, (see for example [1]), and in most cases, they succeed to reduce the time required to find acceptable solutions.

Traditionally, PGAs have tried to efficiently exploit the performance offered by massively parallel processing systems (MPPs). The development of the Grid has opened up avenues that could lead to a dramatic increase in performance of PGAs, in terms of execution time and problem size. However, the execution and development of Grid applications requires a high level of expertise and a significant amount of effort.

The main difficulties arise from the characteristics of the Grid itself, namely: complexity, heterogeneity, dynamism and high fault rate. To overcome these difficulties, we have developed a Globus submission framework, Grid *W*ay [2],

* This research was supported by Ministerio de Ciencia y Tecnología through the research grant TIC 2003-01321 and Instituto Nacional de Técnica Aeroespacial.

that allows an easier and more efficient execution of jobs on a dynamic Grid environment. Grid*W*ay automatically performs all the job scheduling steps [3] (resource discovery and selection, and job preparation, submission, monitoring, migration and termination), provides fault recovery mechanisms, and adapts job execution to the changing Grid conditions [4].

On the other hand, the Grid lacks of a standard programming paradigm to port existing applications among different environments. The DRMAA (Distributed Resource Management Application API) [5] specification, developed within the *Global Grid Forum* (GGF)[1] framework, tries to fill this gap. The DRMAA specification constitutes a homogeneous interface to different DRMS (Distributed Resource Management Systems) to handle job submission, monitoring and control, and retrieval of finished job status.

In this work we analyze the distribution of a PGA across the Grid using the DRMAA standard API and the Grid*W*ay framework. In particular, the PGA considered here adopts a modified version of the fully connected multi-deme approach, to adapt it to the Grid characteristics mentioned above. The rest of the paper is organized as follows, Section 3 briefly reviews the main strategies proposed in the literature to distribute GAs. Then, we propose in Section 3.1 a Grid-aware distribution scheme of GAs, and we describe its implementation with DRMAA. The performance of this scheme in a research testbed is then discussed in Section 4. The paper ends with some conclusions.

2 Parallel Genetic Algorithms

The different alternatives proposed in the literature to parallelize genetic algorithms can be classified in three main categories [6]:

- **Single Population (Panmitic GA)**. This kind of GAs is usually implemented using a Master/Worker paradigm [7]. Panmitic GAs can be efficiently used when the evaluation function requires a considerable amount of computational work. The main advantage of this method is that the search behavior of the sequential GA is not altered, and therefore all the available GA theory can be applied directly. However this approach is not well suited for a Grid because of the high network requirements of its communication pattern.
- **Single Population (Fine Grain) GA**. This type of GA has only one population and its spatial structure limits the interactions between individuals. This limit can be imposed at the chromosome level (each member can only interact with their neighbors) or at the population level (only member of the same subpopulation may mate during crossover).
- **Coarse Grain GA**. The main population is divided into subpopulations (demes), each one is independently evaluated in a different node. Probably the most important characteristic of these algorithms is the communication topology used to exchange information between subpopulations. The possible communication patterns include ring model (processes can only interact with

[1] http://www.gridforum.org (2004)

their neighbors in a ring topology), master-slave (slave processes swap best individuals with the master) or all-to-all (all process swap best individuals with the others).
This kind of GA is difficult to understand because of the effects of migrations between populations are not fully understood. Moreover, coarse grain GAs introduce fundamental changes in the implementation of a simple GA.

In this work we will use a modified version of the coarse grain approach (see Section 3), since this algorithm does not imply a tightly coupled deme topology. Therefore it is more tolerant to the high latencies and dynamic bandwidths that can be expected in the Internet, unlike the single population alternatives.

3 Grid-oriented Coarse Grain Genetic Algorithms

In order to develop efficient Grid-oriented genetic algorithms (GOGAs, following the notation introduced by H. Imade et al. [8]), the dynamism and heterogeneity of a Grid environment must be considered. In this way, traditional load-balancing techniques could lead to a performance slow-down, since, in general the performance of each computing element can not be guaranteed during the execution. Moreover, some failure recovery mechanisms should be included in such a faulty environment.

3.1 Algorithm Description

Taking into account the above considerations we will use a fully connected multi-deme genetic algorithm. In spite of this approach represents the most intense communication pattern (all demes exchange individuals every generation), it does not imply any overhead since the population of each deme is used as checkpoint files, and therefore transferred to the client in each iteration.

The initial population is uniformity distributed among the available number of nodes, and then a sequential GA is locally executed over each subpopulation. The resultant subpopulations are transferred back to the client, and worst individuals of each subpopulation are exchanged with the best ones of the rest. Finally, a new population is generated to perform the next iteration [6]. The schema of this algorithm is depicted in figure 1.

However, the previous algorithm may incur in performance losses when the relative computing power of the nodes involved in the solution process greatly differs, since the iteration time is determined by the slowest machine. In order to prevent these situations we allow an *asynchronous* communication pattern between demes. In this way, information exchange only occurs between a fixed number of demes, instead of synchronizing the execution of all subpopulations. The minimum number of demes that should communicate in each iteration depends strongly on the numerical characteristics of the problem. We will refer to this characteristic as *dynamic connectivity*, since the demes that exchange individuals differs each iteration.

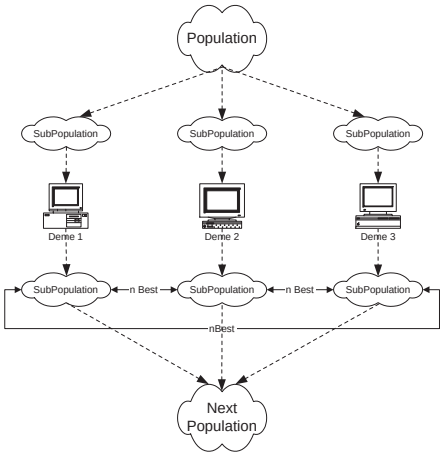

Fig. 1. Schema of fully-connected multi-deme genetic algorithm, with three computing nodes

3.2 Distributed Resource Management Application API

The Distributed Resource Management Application API (DRMAA) is an API specification for job submission, monitoring and control that provides a high level interface with Distributed Resource Management Systems (DRMS). In this way, DRMAA could aid scientists and engineers to express their computational problems by providing a portable direct interface to DRMS.

Although the DRMAA standard can help in exploiting the intrinsic parallelism found in some application domains, like GAs, the underlying DRMS is responsible for the efficient and robust execution of each job. The following aspects are considered by the Grid*W*ay framework, used in this work:

- Given the dynamic characteristics of the Grid, the Grid*W*ay framework periodically adapts the schedule to the available resources and their characteristics [4].
- The Grid*W*ay framework also provides adaptive job execution to migrate running applications to more suitable resources. So improving application performance by adapting it to the dynamic availability, capacity, cost of Grid resources, or its own requirements and preferences [4].
- Grid*W*ay also provides the application with fault tolerance capabilities by capturing GRAM callbacks, by periodically probing the GRAM *jobmanager*, and by inspecting the output of each job.

In particular, the following list describes the DRMAA interface routines implemented within the Grid*W*ay framework (see [9] for a detailed description of the C API):

- Initialization and finalization routines: Initialize and finalize a DRMAA session.

```
//Initialize a new DRMAA session.
rc = drmaa_init (contact, error)
//Execute all jobs consecutively
for (i=0; i < ALL_JOBS; i++)
   rc = drmaa_run_job(job_id, jt, err_diag)
//Execute GOGA if it dosen't rise objective_function
while (!this->objective_function()){
 //Wait for (dynamic connectivity degree) jobs
 //and store results
 for (i=0; i < NUM_JOBS; i++)
  rc = drmaa_wait(job_id, &stat, timeout, rusage, err_diag)
 this->store_results())
 //Execute (dynamic connectivity degree) jobs consecutively
 for (i=0; i < NUM_JOBS; i++)
  rc = drmaa_run_job(job_id, jt, err_diag)
}
//Finalize DRMAA session.
rc = drmaa_exit(err_diag)
```

Table 1. Implementation of the Grid-oriented coarse-grain genetic algorithm using the DRMAA standard

  - `drmaa_init`. Initialize DRMAA API library and create a new DRMAA session.
  - `drmaa_exit`. Disengage from DRMAA library and allow the DRMAA library to perform any necessary internal cleanup.
- Job template routines: These routines enable the manipulation of job definition entities to set parameters such as the executable.
  - `drmaa_set_attribute`. Adds ('name', 'value') pair to list of attributes in job template.
  - `drmaa_allocate_job_template`. Allocate a new job template.
  - `drmaa_delete_job_template`. Deallocate a job template. This routine has no effect on jobs.
- Job submission routines:
  - `drmaa_run_job`. Submit a job with attributes defined in the job template
  - `drmaa_run_bulk_jobs`. The *bulk* jobs are defined as a group of n similar jobs with a separate job id.
- Job control and monitoring routines: These routines are used to control and synchronize jobs, and monitor their status.
  - `drmaa_control`. Start, stop, restart, or kill a job.
  - `drmaa_synchronize`. Wait until all jobs specified, have finished execution.
  - `drmaa_wait`. This routine waits for a single job to finish execution.
  - `drmaa_job_ps`. Get the program status of a job.

Table 1 shows the implementation of the grid-oriented genetic algorithm describe above, using the DRMAA standard.

4 Experimental Results

In this section we will evaluate the functionality and efficiency of the Grid-oriented Genetic Algorithm described in Section 3, in the solution of the One-Max problem [10]. The One-Max is a classical benchmark problem for genetic algorithm computations, and it tries to evolve an initial matrix of zeros in a matrix of ones.

In our case we will consider an initial population of 1000 individuals, each one a 20x100 zero matrix. The sequential GA executed on each node will performed a fixed number of iterations (50), with a mutation and crossover probabilities of 0,1% and 60%, respectively. The exchange probability of best individuals between demes is 10%.

The following experiments were conducted on a research testbed made up of three different organizations, and based on the Globus Toolkit 2.4 [11]. See table 2 for a brief description of the resources in the testbed.

Name	VO	Model	Speed	OS	Memory	DRMS
hydrus	UCM	Intel P4	2.5GHz	Linux 2.4	512MB	fork
cygnus	UCM	Intel P4	2.5GHz	Linux 2.4	512MB	fork
aquila	UCM	Intel PIII	700MHz	Linux 2.4	128MB	fork
babieca	CAB	5×Alpha DS10	450MHz	Linux 2.2	256MB	pbs

Table 2. Characteristics of the machines in the research testbed.

Figure 2 shows the execution profile of 4 generations of the GOGA, with a 5-way *dynamic connectivity*. Each subpopulation has been traced, and labeled with a different number (P_{deme}). As can be shown, individuals are exchanged between subpopulations $P1, P2, P3, P4, P5$ in the first generation; while in the third one the subpopulations used are $P1, P2, P4, P7, P8$. In this way the *dynamic connectivity*, introduces another degree of randomness since the demes that communicate differ each iteration and depend on the dynamism of the Grid.

In order to study the effect of the dynamic connectivity we will consider five different executions, with different degrees of dynamic connectivity. In general, a high degree implies more demes exchanging individuals in each iteration, but also a higher execution time per iteration. On the other hand, a low degree reduces the iteration time, but deteriorates the numerical properties of the algorithm (the migration rate of individuals between subpopulations is also reduced). This effect is clearly shown in figure 3, in this case the optimum configuration is a 5-way connectivity.

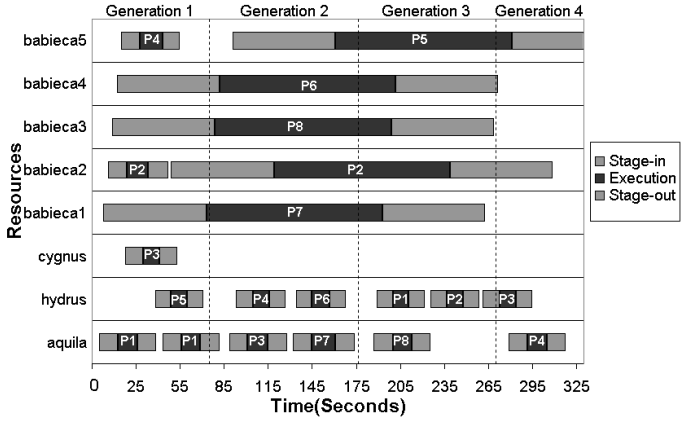

Fig. 2. Execution profile of four generations of the GOGA, each subpopulation has been labeled with P_{deme}

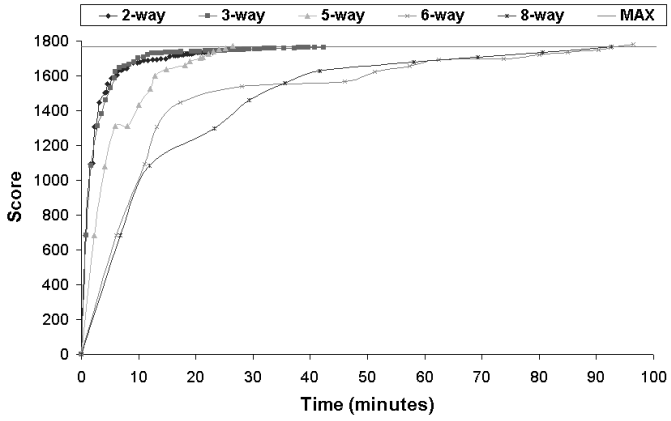

Fig. 3. Score versus execution time for five different degrees of connectivity

5 Conclusion

In this work we have presented an efficient Grid-oriented genetic algorithm. Our approach uses a fully connected multi-deme GA, with a dynamic connectivity between subpopulations to deal with the heterogeneity of the Grid. The optimum degree of connectivity depends on both, the computational characteristics of the Grid nodes, and the computational problem.

The GOGA has been developed taking advantage of the Grid*W*ay framework features and the DRMAA API. In this way, it have been shown that DRMAA can aid the rapid development and distribution across the Grid of typical genetic algorithm strategies.

References

1. Kang, L., Chen, Y.: Parallel Evolutionary Algorithms and Applications. (1999)
2. Huedo, E., Montero, R.S., Llorente, I.M.: A Framework for Adaptive Execution on Grids. J. of Software – Practice and Experience **34** (2004) 631–651
3. Schopf, J.M.: Ten Actions when Superscheduling. Technical Report GFD-I.4, Scheduling Working Group – The Global Grid Forum (2001)
4. Huedo, E., Montero, R.S., Llorente, I.M.: Adaptive Scheduling and Execution on Computational Grids. J. of Supercomputing (2004) (in press).
5. Rajic, H., Brobst, R., Chan, W., Ferstl, F., Gardiner, J.: Distributed Resource Management Application API Specification 1.0. (2004)
6. Cant-Paz, E.: A Survey of Parallel Genetic Algorithms (1999)
7. Alba, E., Nebro, A.J., Troya, J.M.: Heterogeneous Computing and Parallel Genetic Algorithms. (2002)
8. Imade, H., Morishita, R., Ono, I., Ono, N., Okamoto, M.: A Grid-oriented Genetic Algorithm Framework for Bioinformatics. New Generation Computing **22** (2004) 177–186
9. Haas, A., Brobst, R., Geib, N., Rajic, H., Tollefsrud, J.: Distributed Resource Management Application API C Bindings v0.95. (2004)
10. Schaffer, J., Eshelman, L.: On Crossover as an Evolutionary Viable Strategy. In Belew, R., Booker, L., eds.: Proceedings of the 4th International Conference on Genetic Algorithms, Morgan Kaufmann (1991) 61–68
11. Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. International Journal of Supercomputer Applications **11** (1997) 115–128