JOB SCHEDULING IN COMPUTATIONAL GRID USING GENETIC ALGORITHMS

Mohsin Saleem, Savitri Bevinakoppa
School of Computer Science and Information Technology, RMIT Univeristy, Mlebourne,Australia

Keywords: Grid Computing, Scheduling, Genetic Algorithms

Abstract: The computational Grid is a collection of heterogeneous computing resources connected via networks to provide computation for the high-performance execution of applications. To achieve this high-performance, an important factor is the scheduling of the applications/jobs on the compute resources. Scheduling of jobs is challenging because of the heterogeneity and dynamic behaviour of the Grid resources. Moreover the jobs to be scheduled also have varied computational requirements. In general the scheduling problem is NP-complete. For such problems, Genetic Algorithms (GAs) are reckoned as useful tools to find high-quality solutions. In this paper, a customised form of GAs is used to find suboptimal schedules for the execution of independent jobs, with no inter-communications, in the computational Grid environment with the objective of minimising the makespan (total execution time of the jobs onto the resources). Further, while using the GA-based approach the solution is encoded in the form of chromosome, which not only represents the allocation of the jobs onto the resources but also specifies the order in which the jobs have to be executed. Simple genetic operators i.e., crossover and mutation are used. The selection is done on the using Tournament Selection and Elitism strategies. It was observed that the specification of order of the jobs to be executed on the Grid resources played a significant role in minimising the makespan. The results obtained from the experiments performed were also compared with other heuristics and the GA-based approach by other researchers for job-scheduling in the computational Grid environment. It was observed that the GA-based approach used in this paper was able to achieve much better performance in terms of makespan.

1 INTRODUCTION

The objective of scheduling is to minimise the execution time of the jobs on the Grid resources and to balance the load across the resource as much as possible. The scheduling techniques depend heavily on the nature of the jobs to be scheduled. Jobs can be independent or dependent on each other. Independent jobs may require an input file and can execute in any sequence according to their resource requirements. In the case of dependent jobs, the results are communicated to other jobs as inputs and this communication pattern can be represented using task graphs. The scheduling of both kind of jobs mentioned above, can be static or dynamic. In the case of static scheduling all the job requirements and the information about the resources in the Grid are already known before applying the scheduling technique. However in the case of dynamic scheduling, assumptions about the jobs and performance predictions about the Grid resources are used to take scheduling decisions at run time of the jobs. This paper addresses the static scheduling problem of the independent jobs in the Grid computing environment.

The scheduling problem is NP-complete (Hachbaum, 1997), so many heuristics have been suggested to find an optimal solution to this problem in the Grid computing environment (Casanova et al, 2000). In this paper a simple form of GAs is applied to the scheduling problem in the Grid computing environment. Special chromosome representations are used to produce the schedules, which then help in minimising the makespan (total execution time of the jobs onto the resources) and assist in balancing the load on resources.

2 PROPOSED SOLUTION

This model contains an information keeper, which provides information about the jobs to be scheduled as well as the resources in the Grid. Jobs are

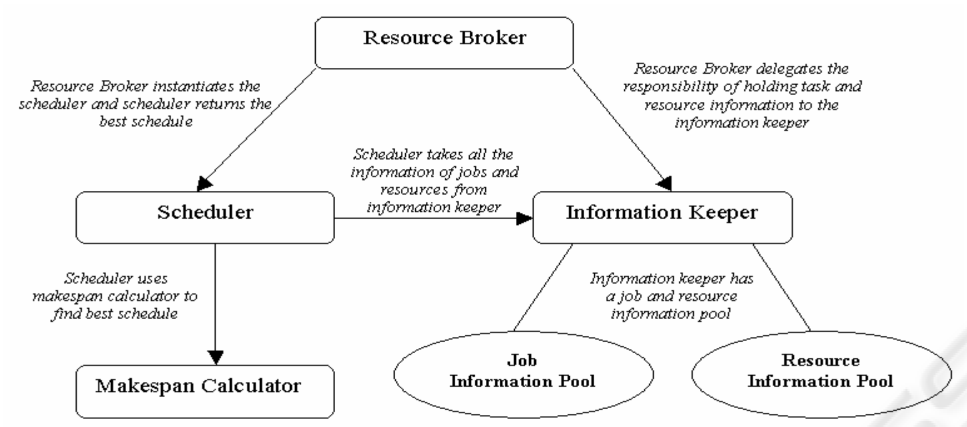
Resource Broker
Resource Broker instantiates the scheduler and scheduler returns the best schedule
Resource Broker delegates the responsibility of holding task and resource information to the information keeper
Scheduler takes all the information of jobs and resources from information keeper
Scheduler
Information Keeper
Scheduler uses makespan calculator to find best schedule
Information keeper has a job and resource information pool
Makespan Calculator
Job Information Pool
Resource Information Pool

Figure 1: Proposed Model

specified by a number of requirements such as required computing power, total execution time, required architecture, required operating system, and size and location of data input file. The users of the Grid submit jobs to a resource broker. The resource broker uses the information from the information system and the jobs' requirements as submitted by the users to generate a schedule.

To create a schedule, the resource broker will ask the scheduler component to find an appropriate schedule based on the information of jobs and resources. The scheduler can use any algorithm to produce a schedule. The scheduler makes use of the makespan calculator component, which estimates the total run time of the schedule. The purpose of the scheduler is to produce a schedule with a minimum makespan. The makespan includes not only the execution time of the jobs but also the data transfer time for the input files.

A customised implementation of Simple Genetic Algorithms (Holland, 1975) is used in this paper to create a schedule for independent jobs to run on the Grid resources with the objective of minimising the makepsan. A random method is used to create initial population of schedules and then applied standard genetic operators to produce new schedules. Two forms of chromosome representation are used and the results are compared. First form only considers the allocation of jobs on the resources. The second form considers the order of the jobs on each resource along with the allocation of jobs on the resource. Single-point, two-point, uniform and cycle crossovers are used for reproduction of individuals. The selection is based on tournament selection (Srinivas et al, 1994) technique. Elitism (Mitchell, 1996) is also used to make sure that the best chromosome remains in the population. Scheduling with GAs has been proved a good tool for providing very good solutions to large scheduling problems. The use of a simple model reduces the complexity of the solution and makes it easier to integrate the solution into the other models.

The application model chosen for this study was that of independent jobs with no communication between them. This means that the jobs have no precedence constraints and can be scheduled in any possible order. The performance criterion was to reduce the makespan of the schedule. The GAs provide a flexible framework for this kind of scheduling problems; in this work a customised form of GAs is used. The main focus of the paper is on the scheduling using GA, so few assumptions were made regarding the information about the jobs and the resources. The details of the solution architecture and the implementation of GA used for the scheduling are presented below. Every job in the job pool gets unique identification number Job ID, Resource ID in XML format.

3 GA SCHEDULER ARCHITECTURE

The model used to solve the scheduling problem is given in Figure 1.
The model used consists of following basic components:

3.1 Resource Broker

The resource broker is responsible for getting requests for resources to allocate jobs on them. The resource broker, with the help of the information keeper component obtains the job requirements for the resources. This component is not only responsible for allocation of jobs on the resources but also for monitoring their progress and returning the results to the user(s).

Once the information has been updated in the information keeper component, the resource broker asks another component called scheduler to use this information and come up with a good schedule. The good schedule in this paper is the one that helps in finishing the job execution on the resources as early as possible and keeps a balance of the load on the resources at the same time.

3.2 Information Keeper

The information keeper is responsible for keeping the information about the resources and the jobs to be allocated on those resources. It contains two sub-components called job information pool and resource information pool.

The information keeper stores the information about the jobs in the job information pool. The resource information pool is updated when the resource broker has to allocate job to the feasible resources. This information can be obtained from services such as MDS-2 (Zhang et al, 2004) (a part of Globus toolkit (Foster et al, 1998)) or NWS (Lowekamp, 1999).

3.3 Scheduler

This component is instantiated by the resource broker once the information has been collected by the information keeper. The scheduler assumes that all the information in the job information pool and the resource information pool of the information keeper is the latest and builds a good schedule based on that information.

In this paper, a Genetic algorithm based approach is used to find a good schedule. In GAs the new generation of solutions go through an evaluation process. The scheduler component evaluates the schedules produced using information from the information keeper. The best schedule made by the scheduler is returned to the resource broker.

3.3.1 Unordered Schedule

In one implementation of the scheduler, the schedule is produced without any specification of the order in which jobs have to be executed. The order is determined by the resource on which jobs have to run. Here we assume that when the scheduler does not specify the order the jobs run on the resource in a First Come First Served (FCFS) fashion.

3.3.2 Ordered Schedule

In the other implementation of the scheduler, the order of the jobs on the resources is also specified. By specifying order, makespan can be minimised. If jobs 1, 2 and 3 are allocated on a resource, allocating them in a different order affects the total execution time. It is assumed that the resources can run jobs in parallel. In the example shown in Figure 2, the computing power is shown on the horizontal side of the graph and time spent to execute the job is shown on the vertical side. In the first case, job 2 and 3 are allocated first in parallel. Job 1, which is a slightly larger job has to wait until job 3 finishes execution, thus increasing the makespan. In the other case if job 1 is allocated first with job 2 running in parallel, job 3 can start execution in parallel with job 1 when job 2 is over. This reduces the makespan significantly as shown in Figure 2.

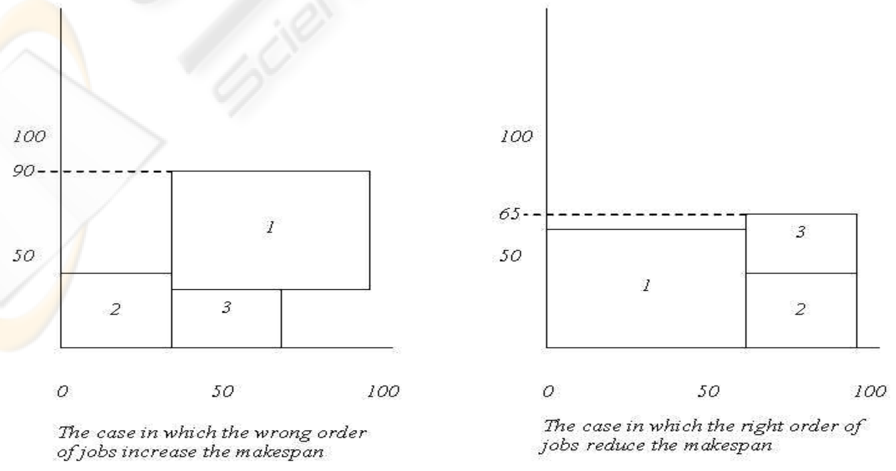
100
90
50
1
2
3
0
50
100
The case in which the wrong order of jobs increase the makespan
100
65
50
1
3
2
0
50
100
The case in which the right order of jobs reduce the makespan

Figure 2: Example to show how order affect the makespan

3.3.3 Makespan Calculator

A makespan calculator is used by the scheduler to calculate the makespan of a schedule produced. It virtually allocates the jobs on the resources as specified by the schedule and returns the makespan. To get information about the jobs and the resources the makespan calculator contacts the information keeper, whenever it needs it. The scheduler uses this makespan to determine how good or bad is the schedule produced.

4 GENETIC ALGORITHMS BASED APPROACH FOR SCHEDULING

4.1 Chromosome Representation

In this paper, two representations of chromosomes have been used, the first representation considers the assignment of jobs on the resources while the other considers the order of jobs on the resources as well. In the first representation, the chromosome consists of a string where the positions in the string represent the jobs to be scheduled and the value at each position represents the resource on which job has to be executed. This string is termed a mapping string as it gives mapping of jobs on the resources. In this case the order of jobs on the resources is not considered and is left to the resource to decide the order as shown in Figure 3.

In the second representation, another string is used in addition to the mapping string to consider the order of jobs to be run on the resources. In this string the positions in the string represent the jobs and the values at each position in the string represent the sequence in which the jobs are submitted to the resources. This string is termed a sequence string as it gives the sequence of jobs submission on the resources as given in Figure 4.

In this example, if we consider the allocations to the resource 4 we find that jobs 4, 5 and 9 are allocated to this resource. If we do not consider the sequence string, then the jobs 4, 5 and 9 will be executed on resource 4 in the order {4, 5, 9}. On the other hand, the sequence string enforces the order {9, 4, 5}.

Jobs	1	2	3	4	5	6	7	8	9	10
Resources	3	1	2	4	4	1	2	3	4	2

Figure 3: Chromosome with mapping string.

Jobs	1	2	3	4	5	6	7	8	9	10
Resources	3	1	2	4	4	1	2	3	4	2
Sequence	8	2	4	6	10	3	7	5	1	9

Figure 4: Chromosome with mapping and sequence strings

4.2 Initial Population Generation

In the initial population step, a fixed number of chromosomes are generated. In GA used here, random methods are used to generate mapping strings and sequence strings. While creating the initial population of sequence strings, the values in the genes are kept unique at the chromosome level. To create the initial population of mapping strings of the chromosomes, the list of feasible resources is used, which every job-information unit contains. The information keeper generates this list for every job in the job information pool. In this step the values are assigned in the mapping strings randomly, from the list of feasible resources for each job. For example, if job 2 has resources 1, 2 and 4 in the list of feasible resources then while creating an initial population of mapping strings, any value can be assigned at position 2 out of 1, 2 and 4.

To create the initial population of sequence strings, for each string the values are assigned based on the following two constraints: first, the values should be unique and second, the values should be less than the total number of jobs. A simple method is used to create these strings. For each string a list L is created containing number ranging from 1 to j, where j is the total number of jobs to be scheduled. A value k is picked up from L and put into the sequence string at its starting index. The value k is then removed from L. After that, each time k is picked randomly from L, put in the first available index in the sequence string and then removed from L. This process is repeated to generate every sequence string in the initial population. The reason for using random methods to generate an initial population is to introduce diversity into the population - the more diverse the initial population, the more chances there are to reach an optimal solution.

4.3 Evaluation

In the evaluation step the population under evaluation is checked for fitness. The smaller the makespan, more fit the chromosome is. To determine the fitness the GA makes use of the makespan calculator. In case the order is not important, the makespan calculator calculates makespan in a First-Come-First-Served (FCFS) way on the resources given by the mapping string. In the other case, the order of execution of jobs is given by a sequence string and the makespan calculator calculates makespan based on order given by the sequence string. The GA used here also determines the best chromosome and the average fitness of the generation under evaluation.

4.4 Selection and Crossover

In this step, the parents' chromosomes are selected to take part in reproduction to produce offsprings. One option is to use the top half chromosomes sorted in the order of fitness, however this may lead to a local optimum and ignore the global optimum (Wilkinson, 1999). To avoid this situation, tournament selection (Wilkinson, 1999) is used. The GA, in this step, chooses pairs of chromosomes randomly from the population and selects the chromosome with better fitness as a parent of the next generation. Tournament selection provides a fairly good chance for individuals to take part in reproduction. The problem with this selection method is that it produces parents half in number than the number of individuals in the population. This problem is addressed later in the crossover step. Uniform Crossover is used for crossover for both mapping and sequence strings.

5 EXPERIMENTS AND ANALYSIS OF RESULTS

In the following sub-sections, the results obtained from all the test cases used are given. These results are obtained using following parameters:
Number of individual in a population = 1000
Total number of generations = 500
Uniform Crossover for mapping strings
Cycle Crossover for sequence strings.
Mutation operators
Crossover Probability = 0.95
Mutation Probability = 0.5
Tournament Selection with Elitism
Both representations of chromosomes
The termination condition is the completion of 500 iterations
These parameters were chosen after running a series of experiments using the test cases given below and found out to be best for this solution of the scheduling problem. In the case of crossover for mapping strings, the single-point and the two-point crossovers produced almost similar results. In the following sections the test cases and the analysis of the results are given.

5.1 Test 32-4

This experiment had 32 jobs with different computational requirements to be scheduled on 4 resources. In this experiment, the first 24 jobs can run on all the 4 resources, but the other 8 jobs can run only on resource 3 and 4.

Figure 5 shows the makespans achieved by the scheduler when it is run 20 times. The scheduler was able to get the makespan of 180 on 4 occasions when the ordering of jobs was specified. It can be inferred that scheduler resulted in a degradation of 6% from the optimal solution on 20% of occasions when ordering is used. The most frequently occurring makepsan in this case was 190 - occurring 12 times in 20 attempts of the scheduler. It can be concluded that the scheduler was able to find a solution which has a degradation of 12% from the optimal solution on 60% of occasions. In short, in 90% of attempts, the makespan degradation was in the range of approximately 6% to 12% when ordering is specified. In the case of ordering, not being specified, the makespan degradation was in the range of 12% to 18% in all attempts.

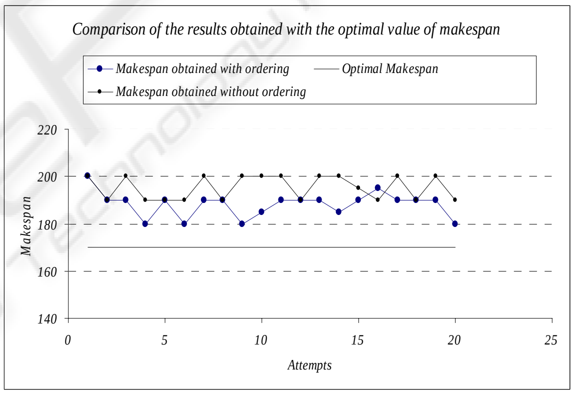

Figure 5: Comparison of results obtained with the optimal makespan

In the case of random scheduling the jobs are assigned randomly on the feasible resources. The makespan given in the graph is obtained by running the random scheduling 20 times and considering the most frequently recurring makespan. In the case of best node first, the makespan is obtained by allocating a job on a resource which can guarantee the minimum completion time for that job. This greedy approach for scheduling is also described in (Kwok et al, 1999) as Minimum Completion Time (MCT) heuristic. The makespans achieved by the scheduler using GA based approach with and without ordering information, are compared with the heuristics defined above.

Figure 6 shows that the approach used in this paper outperformed other heuristics stated earlier in the performance. Random scheduling performed the worst in all the test cases. The best node first heuristic performed better than random scheduling in all experiments. In the case of Test 9-1, best node first had the same performance as the GA and was able to achieve the optimal makespan. In all other test cases, this heuristic was not able to compete with GAs and was far removed from the optimal makespan. The GA with ordering information achieved better performance than the GA with no ordering information. The GA with ordering information was able to produce a schedule with a makespan closer to the optimal makespan.

Figure 7 shows the comparison of standard deviations observed with different algorithms. It was observed that in Test 9-1 all the algorithms performed the same as far as the stability of algorithm was concerned. In all other cases the random heuristic was found to be the most inconsistent. The Best Node First heuristic was the most consistent as it always came up with the same solution. Both the GA approaches were almost identical in consistency, but the approach with no ordering information performed slightly better i.e., less standard deviation. But this fact does not make this approach better than that with ordering information. To elaborate more on the performance of the algorithms, another comparison is made.

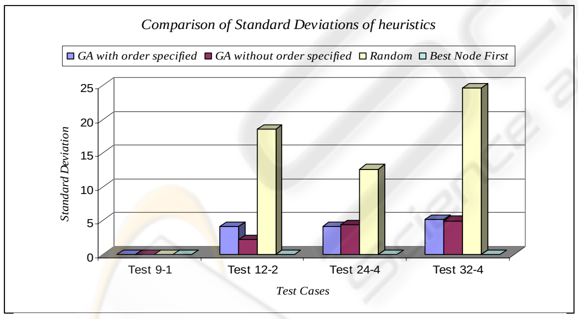

Figure 6: Comparison of performance of GA scheduling with other scheduling heuristics

It is evident from the graph that the GA with ordering information has the best performance as compared to the other heuristics. In all the test cases this approach was observed to come up with a schedule having a makespan which had an average deviation of 12% to 16% from the optimal. In the case of the GA with no ordering information, the average deviation range was observed to be 16% to 24%. In the case of random scheduling the average deviation range was found to be 40% to 50%. The

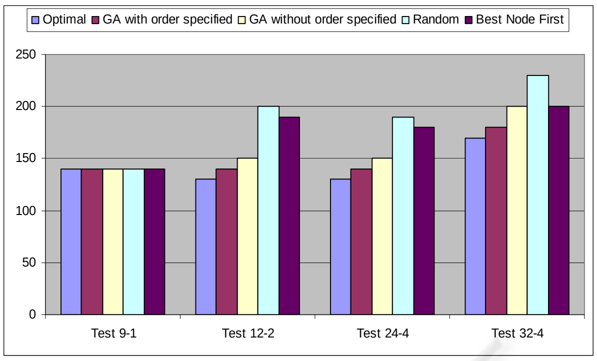

Figure 7: Standard Deviation of the different algorithms.

average deviation range of the best node first scheduling heuristic was observed approximately 18% to 46%. In short, the GA based scheduling using ordering information had the best average performance.

6 CONCLUSION

In this paper, the application model used had independent jobs with no inter-communication. The performance criterion chosen was to reduce the makespan. In the experiments, the GA was run with and without the sequence strings in the chromosomes. It was found that if the GA is used, not only to determine the job allocations on the resources but also for the order in which jobs will run, the performance of the schedule becomes better. The approach used in this paper was also compared with the one used in (Martino et al, 2002) and (Martino, 2003). It was observed that the approach used in this paper performs better that of Martino et al in terms of performance and execution time of the GA. The results of experiments wer also compared with the other heuristics such as random allocation and best node first. The GA based approach had significantly more execution time but it outperformed these heuristics in achieving sub optimal makespans.

REFERENCES

Hachbaum D. S., 1997. *Approximation Algorithms for NP-Hard Problems*, PWS Publishing Company, ISBN 0-534-94968-1.

Casanova H., Legrand A., Zagorodnov D. and Berman F., Heuristic for Scheduling Parameter Sweep Applications in Grid Environments, In *Proceedings of Heterogeneous Computing Workshop 2000*, page(s) 349-363.

Holland J.H., 1975. *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Mich.

Srinivas M., Patnaik L. M., Genetic Algorithms: A Survey, In *IEEE Computer, Jun. 1994*, page(s) 17-26.

Mitchell M., 1996. *An Introduction to Genetic Algorithms,* MIT Press. ISBN 0-262-13316-4.

Zhang X. and Schopf J., Performance Analysis of the Globus Toolkit Monitoring and Discovery Service, MDS2. In *Proceedings of the International Workshop on Middleware Performance (MP 2004), part of the 23rd International Performance Computing and Communications Workshop (IPCCC),* April 2004.

Foster I., Kesselman C., The Globus Project: A Status Report. In *Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop,* page(s) 4-18

Lowekamp B., Miller N., Sutherland D., Gross T., Steenkiste P., and Subhlok J. A Resource Query Interface for Network aware applications, In *Cluster Computing, no. 2, 1999,* page(s) 139-151.

Wilkinson B. and Allen M., 1999, *Parrallel Programming Techiniques and Applications Using Netwroked Workstations and Parallel Computers,* Prentice Hall, ISBN 0-13-671710-1.

Kwok Y. K. and Ahmad I., Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors, *ACM Computing Surveys (CSUR), Volume 31, Issue 4, December 1999,* page(s) 406-471.

Martino V. D., Mililotti M., Scheduling in a Grid Computing Environment Using Genetic Algorithms, In *IPDPS 2002 Workshops, International Parallel and Distributed Processing Symposium, April 15 - 19, 2002,*

Martino V. D., Sub Optimal Scheduling in a Grid using Genetic Algorithms, In *IPDPS'03 International Parallel and Distributed Processing Symposium April 22 - 26, 2003*