Slowdown-Guided Genetic Algorithm for Job Scheduling in Federated Environments

Eloi Gabaldon(✉), Josep L. Lerida, Fernando Guirado, and Jordi Planes

Department of Computer Science, Universitat de Lleida, Lleida, Spain
{eloigabal,jlerida,f.guirado,jplanes}@diei.udl.cat

Abstract. Large-scale federated environments have emerged to meet the requirements of increasingly demanding scientific applications. However, the seemingly unlimited availability of computing resources and heterogeneity turns the scheduling into an NP-hard problem. Unlike exhaustive algorithms and deterministic heuristics, evolutionary algorithms have been shown appropriate for large-scheduling problems, obtaining near optimal solutions in a reasonable time. In the present work, we propose a Genetic Algorithm (GA) for scheduling job-packages of parallel task in resource federated environments. The main goal of the proposal is to determine the job schedule and package allocation to improve the application performance and system throughput. To address such a complex infrastructure, the GA is provided with knowledge based on slowdown predictions for the application runtime, obtained by considering heterogeneity and bandwidth issues. The proposed GA algorithm was tuned and evaluated using real workload traces and the results compared with a range of well-known heuristics in the literature.

Keywords: Resource federation · Scheduling · Co-allocation · Genetic Algorithms · Slowdown-execution predictions

1 Introduction

The computing requirements of scientific applications are continuously growing as is the amount of data that those applications produce and process. The use of new and sophisticated infrastructures is necessary to cover these requirements. One of the earliest infrastructures for covering scientist requirements was the emergence of cluster systems that integrate a number of standalone computers together to work as a single system. Later, despite the reduction in resource costs and the sprawl of infrastructures in organizations and institutions, the requirements still outweighed the local resources. To overcome the problem, grid and cluster federation systems have been developed to enable federated resource sharing logically or physically distributed in different administrative domains. Nowadays, most of the scientific work-flows and applications are deployed in these systems due to their high performance and large storage capacity. Recently, the attention has switched to Cloud computing, a new paradigm for distributed

P.C. Vinh et al. (Eds.): ICTCC 2014, LNICST 144, pp. 181–190, 2015.
DOI: 10.1007/978-3-319-15392-6_18

computing, which transfers local processing to centralized facilities operated by third-party utilities. This paradigm provides on-demand access to thousands of computers distributed throughout the world, with different levels of services and driven by economies of scale, applicable to completely new problems that are beyond the aim of this work. The present paper is focused on federation systems environment, which will allow to take profit for the idle computing resources present in any organization.

The amount of available computing resources in federated systems, the heterogeneity and co-allocation of tasks between different administrative domains, turns job scheduling into an NP-hard problem.The job scheduling optimization methodologies can be mainly categorized as Deterministic Algorithms (DA) and Approximate Algorithms (AA). DAs [1] can find good solutions among all the possible ones but do not guarantee that the best or the near optimal solution will be found. These methodologies are faster than traditional exhaustive algorithms but inappropriate for large-scale scheduling problems. AAs [2,3] employ iterative strategies to find optimal or near optimal solutions. The Genetic Algorithms (GAs) especially find excellent solutions by simulating nature. Although they are less efficient than deterministic algorithms they can find better solutions for large-scale problems in a reasonable time.

In this paper, we focus on the batch-scheduling optimization of parallel applications in heterogeneous federated environments. Specifically, we design different GAs to minimise the Makespan of parallel batch jobs. The first proposal is to use a random strategy to create the initial population in the initialization stage. The second one uses the knowledge produced by a heuristic based on the estimation of the execution slowdown to guide the GA search process. The model of execution slowdown used by the GA was previously proposed by the authors in [4]. The model envisages the resource heterogeneity and also the contention of the communication links to estimate the execution slowdown. This model is used for objective function evaluation in the proposed GA algorithms.

The reminder of this paper was organized as follows. Section 2 presents related work. The proposed genetic algorithm with its variants are elaborated in Section 3. Section 4 demonstrates the performance analysis and the simulation results for real Workload traces. The conclusions and future work are presented in Section 5.

2 Related Work

The potential benefit of sharing jobs between independent sites in federated environments has been widely discussed in previous research [5,6].However, the resource heterogeneity, data transferring and contention in the communication-links have a large influence on the cost of execution of parallel applications, becoming critical aspects for the exploitation of resources and application performance [7–9]. To improve the performance of co-allocation frameworks many policies and models have been proposed. Mohamed et al. [10] proposed the co-allocation of tasks in resources that are close to the input files with the aim

of reducing communication overhead. Jones et al. [8] proposed minimising the communication link usage by maximizing the grouping of tasks in clusters with available resources, but without considering the heterogeneity. The performance of different scheduling strategies using co-allocation based on job queues was analyzed in [6]. This work concludes that unrestricted co-allocation is not recommended and limiting the component sizes of the co-allocated jobs improves performance. The performance for large-scale grid environments was explored in [11,12], concluding that workload-aware co-allocation techniques are more effective at reducing the mean response time and obtaining better load-balance.

Traditional algorithms on job scheduling have in common that jobs are treated individually [1,13]. Allocating jobs without taking into account the rest of the jobs can reduce the performance of future allocations and could decrease overall system performance [14]. More recent research has proposed algorithms that consider later jobs in the queue when making scheduling decisions. Shmueli et al. [14] proposed a backfilling technique in which later jobs are packaged to fill in holes and increase utilization without delaying the earlier jobs. Tsafrir et al. [13] proposed a method to select the most suitable jobs to be moved forward based on system-generated response time predictions. These techniques are based on predetermined order, moving some jobs that accomplish specific deadline requirements forward only on certain occasions. Another point of view had been proposed by Blanco et al. [4], presenting a new technique that tries to schedule the set of jobs in the queue based on the prediction of execution time slowdown.

A common issue in the previous works is that they are based on deterministic heuristics that obtain good results but do not guarantee the best solution. Other techniques based on exhaustive algorithms were explored in the literature [7,15,16]. However, they are impractical for large-scale environments due to their time cost. Alternatively, approximate techniques, such as Simulated Annealing, Tabu Search, Genetic Algorithms, Particle Swarm, etc., have emerged as effective for complex large-scale environments. Particularly, GA are well known for their good results and robustness and are being applied successfully to solving scheduling problems in a wide range of fields [1,17,18].

Our proposal overcome previous works as it is designed to treat large complete set of jobs identifying their resources allocation, and in case that not enough free resources are available, it also determines the best job execution order that minimizes the global makespan.

3 Genetic Algorithm Meta-Heuristic

A Genetic Algorithm (GA) is a stochastic search heuristic used to find nearly-optimal solutions with the use of nature-based techniques. It starts by creating an initial population of solutions known as individuals, each one encoded using a chromosome. To create a new generation, four steps are performed: ranking the individuals driven by a fitness function, a ranking-based selection, the crossover and the mutation. The algorithm is motivated by the hope that, after several generations, the new population will be better than the older ones.

One of the key decisions in GA is the chromosome design, which represents each individual in the population. In order to reduce the chromosome size, and thus the offspring generation time, the chromosome corresponds to the job order in which the jobs have to be executed. Given a chromosome, the final allocation is decided by a deterministic method described in Algorithm 1. It has shown good results for improving the system overall system performance [4]. The method first searches for the most powerful nodes available lines 2-4, where Power(n) is the computational power of the node n. However, for the parallel jobs, their execution time is denoted by the slowest computational node used. So, some of these powerful nodes will not be used at their full. Then, the heuristic tries to make these nodes free, wich can be achieved by using slower ones without loosing performance in the job execution time lines 5-10.

Algorithm 1.. Allocation Algorithm

Require: $\mathcal{Q}$: Set of jobs
Ensure: $\mathcal{A}$: Set of $(Task, Node)$
1: **for** $Job \in \mathcal{Q}$ **do**
2: **for** $Task \in Job$ **do**
3: $\mathcal{A} \leftarrow \mathcal{A} \cup (Task, \text{argmax}_{n \in FreeNodes}(Power(n)))$
4: **end for**
5: **for** $(Task, Node) \in \mathcal{A} : Node = \text{argmax}_{(t,n) \in \mathcal{A}}(Power(n))$ **do**
6: $Node' \in FreeNodes : Power(Node') \geq \min_{(t,n) \in \mathcal{A}}(Power(n))$
7: **if** $\exists Node'$ **then**
8: $\mathcal{A} \leftarrow \mathcal{A} \setminus (Task, Node) \cup (Task, Node')$
9: **end if**
10: **end for**
11: **end for**

To start the evolutionary process, it is necessary to have an initial population composed of a varied set of chromosomes to facilitate a thorough exploration of the search space. In our first proposal, named *GA-Random*, the chromosomes that make up the initial population are randomly generated by using different permutations of the set of jobs.

Next, the GA uses the heuristic described in Algorithm 1 to allocate the jobs to the computational nodes. If we run out of computational nodes, GA predicts the first job to finish using a execution slowdown model and releases its allocated nodes for the subsequent jobs.

The individuals in the population of each generation are evaluated to score the scheduling solutions. In the present work, such a score depends on the makespan.

The makespan is defined as the elapsed time between the submission of the first job until the finalization of the last one. It is calculated as $\max(F_i) - \min(I_j)$ for all the jobs in the workload, F_i being the time when job i finishes, and I_j being the time when job j starts.

The crossover operator combines the information about the different individuals in the current generation to create new individuals as offspring. First a mask of random binary values is generated. For every position with value 1, the job of the first parent is placed into the offspring. For the missing jobs, the order of the second parent is chosen. Additionally, some parents are not crossed but copied to the next generation.

The selection operator is used to choose which individuals should mate. The population is ordered by using the standard tournament selection algorithm.

Finally, the mutation is the operation used to find new points to evaluate the search space. In our case, a mutation is the swapping of two jobs in a given assignment.

In our second proposal, named *GA-METL*, we decided to add some knowledge into the genetic algorithm to speed-up the search: one of the individuals in the initial population is created by a systematic search solution, based on the heuristic presented in [4] named METL (Minimum Execution Time Loss), that considers heterogeneity and bandwidth contention for predicting the execution slowdown and chooses the job with less time lose for execution. This addition of knowledge helps the GA by starting the search with a good solution in the initial population.

4 Experimentation

In this section we have conducted an experimental study with the aim to determine the best GA parameters. Finally we compared the effectiveness of the GA with other heuristics present in the literature.

An important contribution of our proposals is the ability to obtaining better scheduling solutions by means an effective packing of the jobs in the queue. The package size can have a great impact on the effectiveness of the scheduling algorithm, and by this, first was conducted an experimental study to analyze the performance of the GA proposals for different package sizes.

Figure 1 shows the results of the makespan with bars (primary Y axes) and their time-cost with lines (secondary Y axes) obtained by our proposed meta-heuristics. The package size was ranged from 100 to 1000 jobs. As can be observed, the time cost of both proposals GA-METL and GA-Random, is very similar irrespectively of the package size. Instead, when the number of jobs in the package increases the GA-METL is able to reduce the makespan while for the GA-Random the results worsen. This is because the initial random population needs more iterations to evolve to an adequate solution when the search space increases. These results show that providing the GA with some knowledge (GA-METL) allows to improve the algorithm effectiveness.

Other parameters critical for the effectiveness of the genetic algorithms and directly related with the package size are the number of iterations and the population size. To evaluate how these parameters affect the effectiveness of the proposed algorithms a new experimentation was conducted varying them separately. The package size was fixed to 1000 jobs, the value which provides bigger

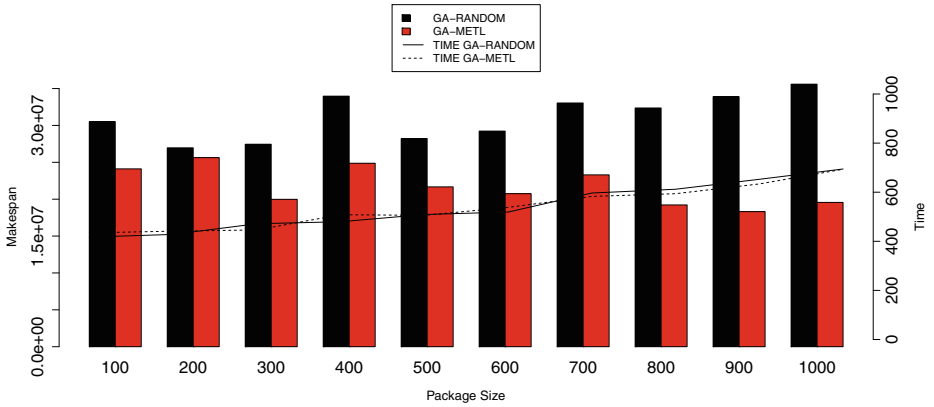

Fig. 1. Method performance for workload of 2000 jobs by package size. Population and Iterations fixed to preliminar values.

differences between GA-Random and GA-METL in the previous experimentation.

Figure 2 shows the results varying the number of iterations from 100 to 500 with the population size fixed to 500. As can be observed, the GA-Random benefits from this increase reducing the makespan. This results show the ability to achieve better solutions when the population evolves. On the other side, the GA-METL that starts with certain knowledge in the initial population produce a significant improvement in makespan. However, the execution times increase largely.

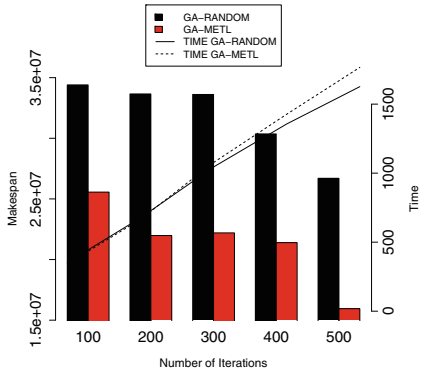

Fig. 2. Performance incrementing the number of iterations when the population size is fixed to 500

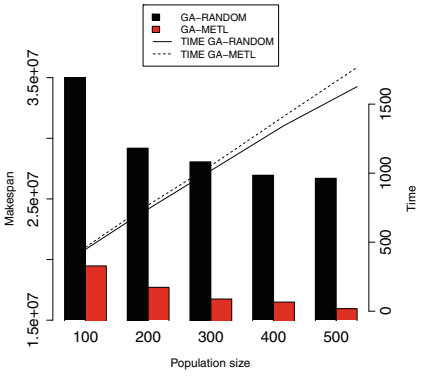

Fig. 3. Performance incrementing the population size when the number of iterations is fixed to 500

The results when varying the population size can be seen in Figure 3. In the case of the GA-Random, larger populations allows major number of crossover and mutations obtaining better results. In the other hand, the behavior for the GA-METL algorithm is similar to the above but with much higher gains of makespan, due to the initial knowledge produces faster convergence towards better solutions. As can be seen, the execution time has the same tendency that in the previous experimentation.

Finally, we compared the performance of the GA proposals with diverse heuristics with co-allocation capabilities from the literature. These heuristic are *JPR*, a variant of Naik's heuristic [16], where the tasks are matched with the most powerful available resources to take advantage of the heterogeneity in multi-cluster resources. *CBS* (*Chunk Big Small*) [8], which tries to allocate a "large chunk" (75% of the job tasks) to a single cluster in an attempt to avoid inter-cluster link saturation. Both of them evaluates individual jobs from the workload. We also used the heuristic *METL* (*Minimum Execution Time Loss*) [4], that is able to consider a set of jobs with the aim to minimize the global job execution slowdown based on the available resources. The set of parameters that guide our proposals were adjusted as shown in Table 1, selected by taking a trade-off between the performance improvement and the computational cost according to the previous experimentation.

Table 1. Settings of GA proposals key parameters

Parameter	GA-Random
Num. Iterations	500
Population Size	200
Mutation Frequency	1%
Crossover Frequency	80%

We have evaluated six different workloads {Wk-1,..Wk-6}, composed of 2000 jobs, from the HPC2N. They were evaluated for two different package sizes, small packages (100 jobs) and big packages (1000 jobs). When evaluating the workload with small packages, Figure 4, we can observe that GA-Random proposal obtained better results than the METL technique (by about 20%), even without using extra knowledge in the initial population. This is because the search space is limited being easier to find a better solutions. Furthermore, GA-METL proposal performs better than any other, as it was expected. The single job traditional heuristics, *JPR* and *CBS*, obtained worst results because they scheduled the jobs individually allocating them in the best available resources without taking into account the following jobs in the queue.

The results obtained for big packages are shown in Figure 5. The GA-Random obtained worse results than the experimentation with small package sizes. By contrast, the GA-METL maintained its behavior obtaining in all cases good makespan results. When the size of the packages are huge, the solutions search

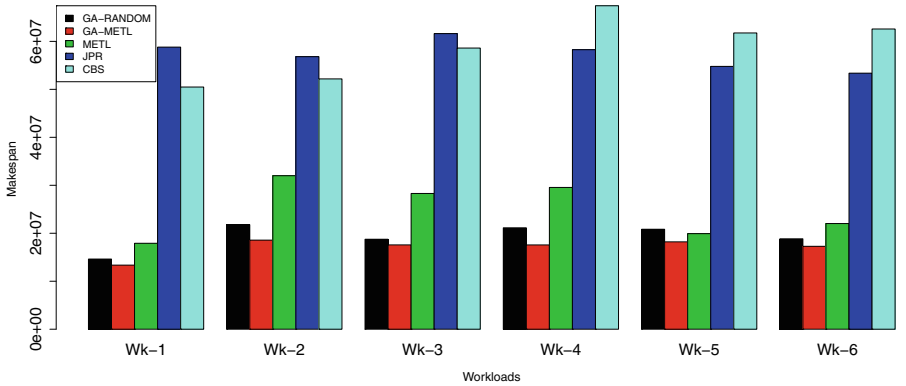

Fig. 4. Makespan results for different workloads composed of 2000 jobs, evaluated with package size of 100

space grew exponentially and both GAs, with the previously chosen parameters, have difficulties to find better results. However, this did not occur with the METL heuristic that was able to obtain good results because its behavior does not depend on the size of the package, and also the best solution it is not guaranteed. In conclusion, to improve the GAs effectiveness for huge packages it is necessary not only to increase the iteration and population parameters but also to redefine the GA functions such as the crossover, mutation, etc.

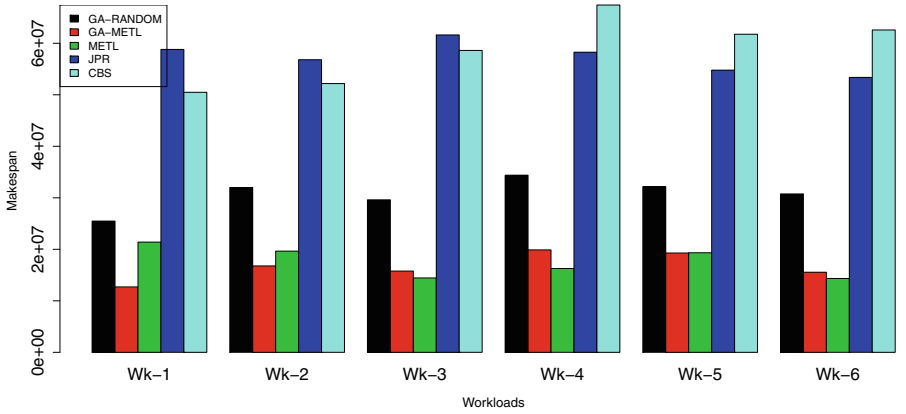

Fig. 5. Makespan results for different workloads composed of 2000 jobs, evaluated with package size of 1000

5 Conclusions

The research for new ways to schedule the jobs in federated resource environments is a critical issue for improving the performance of these systems. We can see that such sophisticated heuristic methods such as the METL heuristic

achieve very good results, outperforming other heuristics that only take into account the optimization of a simpler criteria such as computational power, bandwidth, etc. However, these deterministic heuristics have proven to be inappropriate for large-scale and dynamic environments. In this paper, the authors present a GA that obtains better results than the heuristic methods. The results also showed that providing some knowledge to guide the GA gives better performance results, and also helps the algorithm to converge quickly and reduce the time cost. However, when the package size increase largely the GA showed difficulties to find solutions without modifying the configuration parameters.

In a future work, we are interested in exploring new evolutionary functions applied to our GA proposal in order to obtain better results with lower computational cost when evaluating large packages of jobs. We also aim to study different meta-heuristics that have shown good performance for large-scale problems such as Particle Swarm Optimization (PSO), Ant Swarm Optimization (ASO), or hybrid techniques.

References

1. Braun, T.D., Siegel, H.J., Beck, N., Bölöni, L.L., Maheswaran, M.-C., Reuther, A.I., Robertson, J.P., Theys, M.D., Yao, B., Hensgen, D., Freund, R.F.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing **61**(6), 810–837 (2001)
2. Kolodziej, J., Xhafa, F.: Enhancing the genetic-based scheduling in computational grids by a structured hierarchical population. Future Generation Computer Systems **27**(8), 1035–1046 (2011)
3. Mathiyalagan, P., Suriya, S., Sivanandam, S.N.: Hybrid enhanced ant colony algorithm and enhanced bee colony algorithm for grid scheduling. Int. J. Grid Util. Comput. **2**(1), 45–58 (2011)
4. Blanco, H., Llados, J., Guirado, F., Lerida, J.L.: Ordering and allocating parallel jobs on multi-cluster systems. In: CMMSE, pp. 196–206 (2012)
5. Ernemann, C., Hamscher, V., Schwiegelshohn, U., Yahyapour, R., Streit, A.: On advantages of grid computing for parallel job scheduling. In: CCGRID, pp. 39–39. IEEE (2002)
6. Bucur, A.I.D., Epema, D.H.J.: Scheduling policies for processor coallocation in multicluster systems. IEEE Transactions on Parallel and Distributed Systems **18**(7), 958–972 (2007)
7. Blanco, H., Lerida, J.L., Cores, F., Guirado, F.: Multiple job co-allocation strategy for heterogeneous multi-cluster systems based on linear programming. The Journal of Supercomputing **58**(3), 394–402 (2011)
8. Jones, W.M., Ligon III, W.B., Pang, L.W., Stanzione Jr., D.C.: Characterization of bandwidth-aware meta-schedulers for co-allocating jobs across multiple clusters. The Journal of Supercomputing **34**(2), 135–163 (2005)
9. Liu, D., Han, N.: Co-scheduling deadline-sensitive applications in large-scale grid systems. International Journal of Future Generation Communication & Networking **7**(3), 49–60 (2014)
10. Mohamed, H.H., Epema, D.H.J.: An evaluation of the close-to-files processor and data co-allocation policy in multiclusters. In: IEEE CLUSTER, pp. 287–298 (2004)

11. Finger, M., Capistrano, G., Bezerra, C., Conde, D.R.: Resource use pattern analysis for predicting resource availability in opportunistic grids. Concurrency and Computation: Practice and Experience **22**(3), 295–313 (2010)
12. Wang, C.-M., Chen, H.-M., Hsu, C.-C., Lee, J.: Dynamic resource selection heuristics for a non-reserved bidding-based grid environment. Future Generation Computer Systems **26**(2), 183–197 (2010)
13. Tsafrir, D., Etsion, Y., Feitelson, D.G.: Backfilling using system-generated predictions rather than user runtime estimates. IEEE Transactions on Parallel and Distributed Systems **18**(6), 789–803 (2007)
14. Shmueli, E., Feitelson, D.G.: Backfilling with lookahead to optimize the packing of parallel jobs. Journal of Parallel and Distributed Computing **65**(9), 1090–1107 (2005)
15. Blanco, H., Guirado, F., Lerida, J.L., Albornoz, V.M.: Mip model scheduling for bsp parallel applications on multi-cluster environments. In: 3PGCIC, pp. 12–18. IEEE (2012)
16. Naik, V.K., Liu, C., Yang, L., Wagner, J.: Online resource matching for heterogeneous grid environments. In: CCGRID, pp. 607–614 (2005)
17. Carretero, J., Xhafa, F., Abraham, A.: Genetic algorithm based schedulers for grid computing systems. International Journal of Innovative Computing, Information and Control **3**(6), 1–19 (2007)
18. Zomaya, A.Y., Teh, Y.-H.: Observations on using genetic algorithms for dynamic load-balancing. IEEE Transactions on Parallel and Distributed Systems **12**(9), 899–911 (2001)
19. Garg, S.K.: Gridsim simulation framework (2009). http://www.buyya.com/gridsim
20. Feitelson, D.: Parallel workloads archive (2005). http://www.cs.huji.ac.il/labs/parallel/workload