A Fast Genetic Algorithm Based Static Heuristic for Scheduling Independent Tasks on Heterogeneous Systems

Gaurav Menghani
Department of Computer Engineering,
Thadomal Shahani Engineering College, Mumbai - INDIA
gaurav.menghani@gmail.com

Abstract

Scheduling of tasks in a heterogeneous computing (HC) environment is a critical task. It is also a well-known NP-complete problem, and hence several researchers have presented a number of heuristics for the same. The paper begins with introducing a new heuristic called Sympathy, and later a variant called Segmented Sympathy. A new Genetic Algorithm based heuristic using the Segmented Sympathy heuristic is proposed, which is aimed at improving over the speed and makespan of the implementation by Braun et al. Finally, the results of Simulation reveal that the proposed Genetic Algorithm gave up to 8.34% and on an average 3.42% better makespans. The new heuristic is also about 160% faster with respect to the execution time.

1. Introduction

A Grid is a collection of heterogeneous systems, which may belong to different organizations. Grid computing helps researchers solve complex computational problems without the need to set up one single very powerful system, but instead aggregating the computational resources of several low-cost member systems. A problem is made of several tasks, which are heterogeneous with respect to the computational time required to solve them. Also, the grid is made of several systems with different computational resources. [6]

To fully utilize the tremendous potential of the grid, it is often desired that an optimal mapping of tasks to machines be prepared. However, it has been shown that the problem of finding such a mapping is NP-Complete [2].

Several heuristics have been proposed by researchers, which try to optimize the mapping of tasks [1, 3], including several nature-based heuristics [1,5,7]. These heuristics can be categorized as either on-line mode or batch mode. In on-line mode, a task is mapped to a machine as soon as it arrives. In batch-mode, mapping happens at certain scheduled times called mapping-events. At such mapping events, a set of independent tasks called a meta-task is considered for mapping. A meta-task can include newly arrived tasks, or tasks which were mapped earlier, but could not be executed.

A new batch-mode heuristic, Sympathy is introduced in this research, along with a variant called Segmented-Sympathy. In section 5, the genetic algorithm based heuristic in [1] is discussed, and a modified genetic algorithm based heuristic is proposed, which is designed to improve over the original heuristic with respect to both, the makespan as well as the time of execution.

In section 6 the simulation procedure is discussed which is based on work done in [1]. Finally, the simulation results are discussed.

2. Related Work

The terms ‘scheduling’ and ‘mapping’ are used interchangeably in literature. Several researchers have worked on the problem of mapping independent tasks onto diverse computing resources.

[1] Discusses and compares eleven static heuristics. These include MCT, MET, Min-Min, Genetic Algorithm, Simulated Annealing, A*, OLB, Genetic Simulated Annealing, Tabu Search, etc.

[3], presents the Sufferage heuristic, which is based upon the idea of assigning a particular machine to a task which 'suffers' the most, if the machine is not assigned to that task. This is followed by comparison with Min-Min and Max-Min, and discussion of the results.

A new heuristic called Segmented Min-Min (SMM) with four variants is presented in [4]. The SMM-Avg differs from Min-Min by applying Min-Min into separate 'segments', which are sorted by the average makespan of the segment. SMM-Avg is then shown to improve over Min-Min by up to 12.9%.

An implementation of the GA-based heuristic for the problem can be found in [1]. Selection, Crossover and Mutation operators are defined. Also, population seed, initial population, probability of crossover and mutation, and stopping conditions are defined.

A detailed analysis on genetic algorithm based schedulers is provided in [5]. They present comparison between various crossover and mutation operators.

[1], also provides a comprehensive simulation procedure. It also takes care of different level of heterogeneities of tasks and machines. Overall, it makes use of 12 different types of test cases for assessing the heuristics.

3. Previous Heuristics

Several heuristics have been proposed in the past. In this section, we briefly review some of them.

OLB (Opportunistic Load Balancing): It assigns the tasks, in arbitrary order, to the next available machine. [1]

MET (Minimum Expected Time): It assigns a task to the machine with the best-expected execution time for that task. [1]

MCT (Minimum Completion Time): It assigns a task to the machine with the best-expected completion time for that task. [1]

Min-Min: In Min-Min, the task with the overall minimum completion time in the meta-task, is assigned to the corresponding machine. The selected task is removed from the meta-task and the process continues. [1]

GA (Genetic Algorithm): GA is used for searching large search spaces. It starts with a randomly generated population, or is seeded using a chromosome from some other heuristic. [1, 5] A chromosome is nothing but the mapping of the tasks.

SA (Simulated Annealing): SA is an iterative method. It considers only one solution to the problem at a time. It stochastically accepts or rejects a new solution based upon its fitness and the system temperature. With decrease in system temperature, better solutions are more likely to get accepted. [1]

GSA (Genetic Simulated Annealing): It is a hybrid of GA and SA techniques. [1]

Tabu Search: It is a local search procedure, which keeps track of the search areas already tracked, so as not to repeat them again. [1]

A* Search: It makes use of the A* search technique for generating a mapping. [1]

Sufferage: For every task, its sufferage value (the difference between its second best completion time and best completion time) is computed. The task with the highest sufferage value is assigned to the machine with its minimum completion time. The process continues until all tasks are mapped. [3]

SMM (Segmented Min-Min): SMM sorts the tasks according to the minimum/maximum/average expected completion time of each task with respect to each machine. The list is then partitioned into a fixed number of segments. Subsequently, Min-Min is applied individually applied on each segment, in order. [4]

4. Sympathy and Segmented Sympathy

The sympathy heuristic prefers to map tasks, which have a big variation in their completion times before others. The metric for ranking tasks is the sympathy metric, which is defined as $s_i = E(c_i)$ x $V(c_i)$. Where $E(c_i)$ is the mean completion time of task i on all machines, and $V(c_i)$ is the variance of the completion times of task i on all machines. The sympathy metric gives a rough idea of the improvement possible in the makespan, if that particular task is assigned to a machine where it will achieve the best completion time. Thus, the heuristic will 'sympathize' with tasks with high sympathy metric values.

Sympathy

(1) **start**
(2) **do**
(3) Compute s_i for all tasks in M
(4) Find the task i with the highest sympathy metric
(5) Map task i to machine j, with the lowest c_{ij}
(6) Remove i from M
(7) Recompute completion times for all tasks
(8) **while** all tasks in meta-task M are not mapped
(9) **end**

If there are t tasks initially in the meta-task and m machines, it can be easily interpreted that the complexity of the heuristic is $O(t^2m)$.

Segmented Sympathy is a variant of the Sympathy heuristic.

Segmented Sympathy

(1) **start**
(2) Sort each task by their respective sympathy metrics.
(3) Partition the tasks evenly into N segments.
(4) Starting with the segment with the highest values of sympathy metric, apply Min-Min individually on the segments.
(5) **end**

We note that, unlike Sympathy, here the sympathy metrics are not updated for tasks after one task is mapped. Thus, the complexity of the heuristic is $O(t^2)$.

5. Modified GA

The implementation of GA provided in [1] (referred to as Braun GA, henceforth), was found to be somewhat inefficient. The modified GA (referred to as GA, henceforth) implementation has the following features

(1) **Seeding**: Seeded with chromosomes of Min-Min, Segmented Min-Min and Segmented-Sympathy. Moreover, it was noticed that unseeded iterations did not provide makespans of good quality; hence only four seeded iterations were retained.

(2) **Population** & **Elitism**: The population size was fixed at 100. The top 5% chromosomes were allowed to survive to the next generations, instead of just the best. This allowed solutions, which had the potential to come close to optimal solutions, to be retained.

(3) **Swap & Fitness based Crossover**: The crossover is implemented in two steps. A one-bit swap operation with 40% probability, swaps the machine mapping of a single task between two chromosomes. Next, a crossover, which is based upon fitness with 20% probability, takes place. In this, two parents produce a single offspring. The new chromosome has machine assignments based upon the fitness of the parents. If one of the parents is more fit than the other, the offspring is more likely to have the assignments according to that parent.

(4) **Stopping Condition**: The stopping condition was changed to, '100 steps without a change in the elite chromosome'.

(5) **Local Search**: To improve the solutions, a final local search was performed on the chromosome with the best makespan. This was done by swapping task assignments between the more loaded half of the machines and lesser-loaded half of the machines. In the simulation, this often led to about 0.2 - 0.3% improvement in the makespan.

6. Simulation Procedure

The heuristics chosen for simulation were, MCT, Min-Min, Sufferage, Segmented Min-Min Average, Sympathy, Segmented Sympathy, Braun GA and GA. The simulation was conducted to find the most efficient heuristics amongst the chosen few, with makespan as the performance metric. The simulation was conducted as specified in [1]. Three types of ETC matrices, Consistent, Semi-Consistent and Inconsistent were created. Each with four types of heterogeneity, low task - low machine, low task - high machine, high task - low machine and high task - high machine heterogeneities. Each type of matrix had 10 meta-tasks, with t = 512 and m = 16 (i.e. 512 tasks and 16 machines).

After the execution of all heuristics on a particular meta-task, the makespan of the solutions provided were noted. Each heuristic was given a 'penalty' depending upon its performance, which was evaluated as follows:

P_{it} = (makespan of heuristic i for t) / (maximum makespan amongst all heuristics in H for t)

Where, P_{it} is the penalty received by the heuristic i, for the task t, and H is the set of heuristics. Thus, the penalty given to a heuristic i for the task t, lies in the range (0, 1]. P_{it} can be referred to as the normalized makespan.

7. Results

The results of the simulation are as follows. C, S and I refer to Consistent, Semi Consistent and Inconsistent ETC matrices respectively. The tables list the cumulative penalty assigned to each heuristic after 10 meta-tasks of three different types of consistencies each. Every table clubs the results under different types of machine and task heterogeneities.

Low task – Low machine Heterogeneity

Heuristic	C	S	I
MCT	9.46	9.86	9.82
Min-Min	9.49	8.21	8.18
Sufferage	9.67	9.72	9.89
Sympathy	9.50	9.43	9.21
SMM-Average	8.73	8.05	8.20
Seg-Sympathy	9.00	8.48	8.35
Braun GA	9.14	7.91	7.88
GA	8.66	7.83	7.86

Low task – High machine Heterogeneity

Heuristic	C	S	I
MCT	9.80	9.83	9.83
Min-Min	9.53	8.41	8.41
Sufferage	9.77	9.82	9.86
Sympathy	9.48	9.33	9.44
SMM-Average	8.77	8.04	8.09
Seg-Sympathy	9.04	8.36	8.48
Braun GA	9.18	8.00	7.95
GA	8.74	7.78	7.81

High task – Low machine Heterogeneity

Heuristic	C	S	I
MCT	9.59	9.74	9.92
Min-Min	9.60	8.45	8.19
Sufferage	9.33	9.65	9.84
Sympathy	9.32	9.37	9.31
SMM-Average	8.43	8.02	8.12
Seg-Sympathy	8.72	8.38	8.39
Braun GA	9.09	7.94	7.94
GA	8.39	7.75	7.79

High task – High machine Heterogeneity

Heuristic	C	S	I
MCT	9.64	9.65	9.66
Min-Min	9.59	8.18	8.31
Sufferage	9.84	9.72	9.83
Sympathy	9.33	8.94	9.51
SMM-Average	8.74	7.65	8.10
Seg-Sympathy	9.04	8.14	8.34
Braun GA	9.20	7.69	7.88
GA	8.64	7.42	7.74

Average execution time of the two GA-based heuristics was also found. The table below lists the time taken by both heuristics (in seconds), on a 1.73 GHz Pentium M machine with 1 GiB RAM.

Average Execution Time

Heuristic	C	S	I
Braun GA	20.17	26.86	24.41
GA	9.90	8.09	9.53

8. Discussion

It is seen from the results that MCT was comparable to Min-Min in the consistent matrices. However, in other cases, it fared badly and its makespans were one of the worst. Sufferage also did not perform well. It was better than Min-Min only in the High-Low Consistent case.

Sympathy performed better than Sufferage. It matched Min-Min in all consistent cases, and bettered it in the High-Low Consistent and High-High Consistent cases.

Segmented Sympathy fared much better than Sympathy. It outperformed Min-Min in seven of the twelve cases. It was beaten by Min-Min in all four inconsistent cases. But it provided makespans, which were up to 9.1% better than Min-Min (in the High-Low Consistent case) in other cases. Also, Segmented Sympathy was better than Braun GA in all the consistent cases.

Braun GA bettered Min-Min by roughly 3.6% in all the Low-Low cases. Its best performance was when it provided makespans, which were around 6.4% better than Min-Min in the High-Low Semi-consistent case.

Segmented Min-Min Average was the second best heuristic in the results. It was better than Min-Min in all the cases except the Low-Low Inconsistent and

High-High Inconsistent case where it was behind Min-Min by 0.24% and 0.3% respectively. In the Low-High Consistent and High-Low Consistent cases, it outperformed Min-Min by 8.6% and 13.1% respectively.

The proposed GA turned out to be the best heuristic. It did better than Min-Min in all cases. In the High-Low Consistent case it was better than Min-Min by 14.42%. It was also better than Braun GA in all the test cases. In the High-High Consistent and High-Low Consistent cases it got 6.48% and 8.34% smaller makespans. In the Low-Low Inconsistent case, GA scored better than Braun GA even though none of the seed chromosomes of GA were better than the Min-Min chromosome seed of Braun GA, and had a smaller population and lesser iterations. Thus, proving that the crossover and mutation operators were superior to Braun GA. It was also observed that the GA outperforms Braun GA most in the consistent cases (around 5%). On an average, GA gave makespans, which were 3.42% better than Braun GA.

GA also turned out to be much faster than Braun GA. In one of the meta-tasks, it was about 14 times faster. On an average it was observed to be around 2.60 times quicker than Braun GA.

9. Conclusion

In this paper, I began with two new heuristics, Sympathy and Segmented Sympathy. Sympathy does better than the Sufferage heuristic. Segmented Sympathy is an improved variant of Sympathy, and it did better than Min-Min in seven out of twelve cases. It also obtained better makespan than Braun GA in consistent cases.

Finally, the GA based heuristic was introduced, which uses Sympathy and Segmented Sympathy. It has a few differences in implementation when compared to Braun GA (crossover, mutation, population, etc). It performed upto 8.34% better than Braun GA and 3.42% on an average. It also was about 2.60 times faster than Braun GA.

10. References

[1] Tracy D. Braun, Howard Jay Siegel, Noah Beck, Ladislau L. Boloni, Muthucumaru Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, Bin Yao, Debra Hensgen and Richard F. Freund, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems", *Journal of Parallel and Distributed Computing*, 2001.

[2] D. Fernandez-Baca, "Allocating modules to processors in a distributed system", *IEEE Trans. Software Engg.* 15, Nov. 1989, pp. 1427-1436.

[3] Muthucumaru Maheswaran Shoukat and Muthucumaru Maheswaran and Shoukat Ali and Howard Jay Siegel and Debra Hensgen and Richard F. Freund, "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems", *Journal of Parallel and Distributed Computing*, 1999, pp. 107-131.

[4] Min-You Wu, Wei Shu and Hong Zhang, "Segmented Min-Min: A Static Mapping Algorithm for Meta-Tasks on Heterogeneous Computing Systems," *9th Heterogeneous Computing Workshop*, 2000, pp.375-385.

[5] Javier Carretero, Fatos Xhafa and Ajith Abraham, "Genetic Algorithm Based Schedulers For Grid Computing Systems", *International Journal of Innovative Computing, Information and Control,* Vol. 3, No.5, pp. 1-19, 2007.

[6] I. Foster and C. Kesselman., *The Grid - Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1998.

[7] Stefka Fidanova and Mariya Durchova., "Ant Algorithm for Grid Scheduling Problem", *Lecture Notes in Computer Science*, Volume 3743/2006, 2006, pp. 405-412.