Scheduling Parameter Sweep Applications on Heterogeneous Compute Resources

Ryan Tanaka Dept. of Information and Computer Science
University of Hawaii
Honoulu HI, 96822
Email: ryant@hawaii.edu

I. ABSTRACT

***Abstract*—The optimization problem of scheduling tasks onto heterogenous resources in distributed computing environments has been shown to be an NP-complete problem in some cases and is the subject of ongoing research in the field of distributed computing. A number of dynamic and static task-scheduling algorithms have been proposed to tackle this problem. Of the static based scheduling algorithms, there exist heuristic and random based search methods, all of which can provide ideal task schedules in different application and hardware configurations. The problem of accurately benchmarking these algorithms on varying hardware configurations has proved to be time consuming for researchers, and as a result, highly optimized simulation software has been developed over the years to aid in the advancement of high performance and distributed computing research. We first develop a genetic algorithm for the paremeter sweep scheduling problem that uses that uses WRENCH, a workflow management simulation framework, as a means of individual fitness evaluations. Then we evaluate the performance of the schedule output by the genetic algorithm against schedules produced by the Max-Min and Min-Min heuristics. Findings show that, the heuristics, although simple to implement, provide good scheduling performance. Additionally, the genetic algorithm's schedule performance approaches that of the heuristics. However, 150 generations with a population size of 200 is not enough to match the performance of those heuristics as movement through the search space was slow, likely a result of the genotype implementation.**

***Keywords*—Scheduling, Distributed Computing, Workflow Management Systems, Genetic Algorithm**

II. INTRODUCTION

Distributed computing environments have become the platform of choice for executing large scale scientific applications because they afford researchers the ability to execute such applications in a fraction of the amount of time it would take to execute on a single machine. A "fraction of time" in this context could mean an application makespan (execution time) of months instead of a year assuming hardware resources are utilized in a clever manner. This paper focuses on one specific type of scientific application known as a *parameter sweep application*, where the entire application is composed of independent tasks which can require a number of input files [1], [2]. Tasks can require a varying amounts of computation to complete and each task may use one or more files also used by other tasks within the application. In order to execute a parameter sweep application on a set of distributed computing resources, one must specify a schedule with the following information: 1. when and where to send what file, 2. when and where to execute what task. This decision making/scheduling process has been proven to be NP-complete in many cases [3]. Heuristic based, list scheduling approaches such as *Max-Min, Min-Min* are some efficient and effective ways to schedule file transfers and assign tasks to compute hosts [1], [2], [3], [4]. Previous research has suggested that using genetic algorithms as a viable meta-heuristic for the task scheduling problem [5] , [6] where inter task dependencies exist. The purpose of this project is to benchmark a schedule created by a genetic algorithm (GA) and compare the schedule's performance with the afformentioned *Max-Min and Min-Min* heuristics. To accomplish this we use WRENCH, a workflow management simulation framework [7], to simulate a parameter sweep application, the cyberinfrastructure it is to be executed on, and the scheduling logic necessary to map portions of the application onto compute and storage resources.

This paper is organized as follows. Section III begins by formally defining the problem of scheduling parameter sweep applications on heterogenous resources. Then the Max-Min and Min-Min heuristics are introduced. Section IV covers the implementation details of the genetic algorithm used. Section V begins with a description of the experimental scenario used to evaluate the GA against the Max-Min and Min-Min heuristics followed by a discussion of the results.

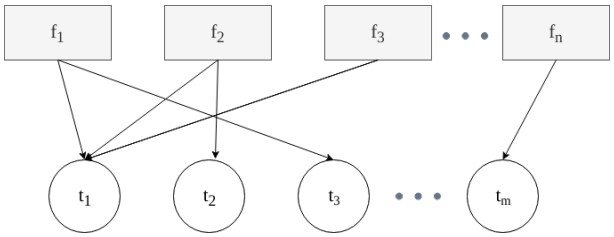

Fig. 1: Graph representation of a parameter sweep application.

III. BACKGROUND

Parameter Sweep Applications

Scientific applications are often represented as directed acyclic graphs (DAGs) where nodes represent computational tasks and input/output files. Links generally will start at a file and extend to a task or vice versa. A link from a file to a task denotes that the task requires the file as an input. Conversely, a link from a task to a file signifies that the task creates that file as its output. Here we focus on one such scientific application known as a parameter sweep, where there are an independent set of n files f_j, $1 \leq j \leq n$ and m tasks t_k, $1 \leq k \leq m$ as depicted in Figure 1. These files have links to tasks, thus forming a bipartite graph. Depending on the application, this bipartite graph may be arranged in a number of ways. For example, consider an application with 10 files and 20 tasks. And say, each file is used by a pair of tasks. This type of application may be easily parallelized as the graph can be partitioned into sections. However, in the case of irregular applications, the dependencies between files and tasks can make it difficult to effectively schedule it onto a set of hardware resources. Scheduling the execution on a single compute resource is trivial, but beyond that, the scheduling problem is shown to be NP-Complete in many cases [3]. Even when executed on modern day, massively parallel compute architectures, these scientific applications can be so large that they take days or months to execute.

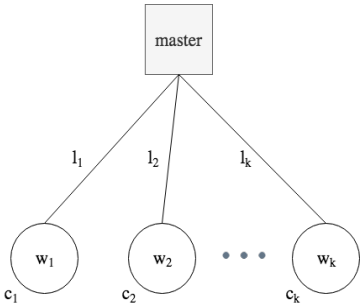

Fig. 2: Master worker infrastructure.

Cyberinfrastructure

The focus of this project is scheduling a parameter sweep application onto a cyberinfrastructure arranged in a *master worker* layout shown in Figure 2. The master node is connected to k worker nodes, w_i where $1 \leq i \leq k$. The links between master and worker nodes, l_i where $1 \leq i \leq k$ have a bandwidth of bw_i, denoted in megabytes per second (MBps). Worker node w_i has a compute speed c_i denoted in floating point operations per second (flops).

The master is responsible for assigning tasks to workers. Depending on the application graph, before the master can map a task to a worker, that tasks's required input files must be present at a persistant storage service located on that same worker. Here we restrict the master from sending more than one file at a time to a worker. Additionally the master may start up a task at any moment on a worker if the required files are present at the worker's persistant storage. Once a file has been sent from the master to some worker, that file will be retained by the worker and may be used by multiple tasks. Furthermore, the master may send the same file to multiple workers such that multiple tasks that require the same file may be executed in parallel by different workers.

Consider a simple application comprised of a single task t that requires some number of floating point operations to complete. t also requires the set of F files. The master will synchronously send each file in F to w_1 over l_1 then instruct w_1 to execute t. The expected makespan of this particular application can be modeled by the following equation:

$$makespan_{expected} = \frac{\sum_{f \in F} size(f)}{bw_1} + \frac{num\_flops(t)}{c_1}$$

Modeling expected execution times is used by a number of scheduling heuristics including Max-Min and Min-Min.

List Scheduling Heuristics

Two of the most common online list scheduling algorithms are Max-Min and Min-Min. Both of these algorithms use estimated minimum completion times (MCT) of each task to determine which set of files and tasks to map to what resource by computing the following:

Algorithm Estimate MCT

$ECTS = [|pending\_tasks|][|workers|]$
for $t \in pending\_tasks$ **do**
  for $w \in workers$ **do**
    $ect = estimate\_completion\_time(t, w)$
    $ECTS[t][w] = ect$
  end for
end for

Max-Min: first selects the task with the maximum of all the minimum estimated completion time. Then, that task is assigned to the worker which will give it the largest estimated completion time out of all the possible workers it could have run on. This is to start long tasks early on slower resources such that smaller tasks can be executed faster on fast resources.

Min-Min: selects the task, worker pair with the fastest estimated completion time and assigns that task to that worker.

IV. Genetic Algorithm

Chromosome Representation

The chromosome representation of a schedule for this problem must be able to capture two things: the mapping of files to hosts and the order in which files are sent from the master. The mapping of files to hosts dictates at which host(s) a task can run on. This mapping is described by the following relation which yields a set of nk $(file, worker)$ mappings.

$$domain = \{f_j \mid 1 \leq j \leq n\}$$
$$range = \{w_i \mid 1 \leq i \leq k\}$$

The order in which files are sent from the master affects when a task can start as no task may start on a host which does not yet have the file(s) it requires. Therefore, a list of nk $(file, worker)$ mappings is used as the genotype such that the order of each mapping determines the order of file transfers. This genotype is specific to the application and cyberinfrastructure specifications. For example, an application with 10 input files and a cyberinfrastructure with 10 workers would yield a genotype with a 100 $(file, worker)$ pairs. One possible instance of a genotype for this scenario may appear as follows:

$$[(f_1, h_1), (f_1, h_2), (f_2, h_1), (f_{10}, h_3)...]$$

Because file transfers dictate where a task may be executed, we ignore task, worker mappings because this can ultimately be derived from file, worker mappings (as well as from simulation the simulation data).

Fitness Evaluation

The objective function is to minimize the application makespan (execution time), therefore individual fitness is based soley on the application makespan given its schedule. Ideally, it would be useful to execute this type of schedule on a real platform, however obtaining the resources to do so may not be possible for certain architectures (they may not exist or could very well be too expensive). For that reason, we use WRENCH simulations to emulate aribtrary applications and cyberinfrastructures. Given an individual's schedule as input to the simulation, we can evaluate its fitness by obtaining the application makespan using that schedule. The lower the application makespan, the higher the individual's fitness.

In many cases, the application will complete (all tasks have been executed) before all file transfers have completed. When evaluating an individual's fitness, the application makespan is determined by the time at which all tasks complete, and not the time at which all file transfers complete (every file worker mapping in the genotype has been processed) because the remaining file transfers contribute nothing to the application makespan.

Selection and Recombination

Selection is done by taking the top 50% of the population. Through recombination, the remaining individuals are then replaced by offspring produced by the top 50%. Because the objective function is to minimize application makespan, binary tournament selection is used twice to obtain two parents for crossover rather than fitness proportional selection. Given a random cross over point k, recombination is done by taking the first k ordered pairs in the first parent and copying them into the the first k ordered pairs of the offspring. The remaining ordered pairs that the offspring is missing is copied from the second parent in the order that they appear there. For example, consider two parents $p1$, $p2$, the offspring os, a random crossover point 3, and a schedule of length 5.

$$p1 = [(f_1, h_1), (f_2, h_2), (f_3, h_2), (f_3, h_1), (f_4, h_1)]$$
$$p2 = [(f_4, h_1), (f_1, h_1), (f_3, h_2), (f_3, h_1), (f_2, h_2)]$$
$$os = [(f_1, h_1), (f_2, h_2), (f_3, h_2), (f_4, h_1), (f_3, h_1)]$$

The idea behind this method of crossover has been used in the development of genetic algorithms for scheduling tasks on a multiprocessor machines [6] where task ordering must be considered and therefore has been adopted here. By using this method of crossover, we retain the order of file, worker mappings for both of the parents in the offpsring.

Mutation

If an individual is selected for mutation, the mutation operator simply swaps two random mappings in the schedule. For example, say $p1$ from the previous example is mutated. The result is $p1\_new$.

$$p1 = [(f_1, h_1), (f_2, h_2), (f_3, h_2), (f_3, h_1), (f_4, h_1)]$$
$$p1\_new = [(f_4, h_1), (f_2, h_2), (f_3, h_2), (f_3, h_1), (f_1, h_1)]$$

V. Experimental Details

Parameter Sweep Application and Infrastructure

A parameter sweep application containing 50 files and 50 tasks is randomly generated. Connections between files and tasks are random. File sizes range from 1 to 1000 megabytes and task computation requirements range from 1 to 1000 flops.

The master worker infrastructure containing 10 workers is also randomly generated. Each worker has a compute speed ranging from 1 to 1000 flops per second and the link between the master and each worker ranges from 1 to 1000 megabytes per second.

When executing the GA and heuristics, the same parameter sweep application and master worker infrastructure is used. The scheduling logic for both the GA and the heuristics is implemented in C++ using the WRENCH API.

Genetic Algorithm

Using the variation operators described in Section IV, the GA itself is implemented in Python. Additionally, Python C++ bindings were developed to enable the GA to pass individuals into the WRENCH simulation for fitness evaluation. Parameters for the GA used in the experiment is described below.

GA Parameters	
population size:	200
elitism:	0.5
parent selection:	binary tournament
mutation rate:	0.5
stopping condition:	150 generations

Results

After evolving over 150 generations, the GA produced an indivdidual with a max fitness (minimum makespan) of 919.221 seconds. Max-Min resulted in an application makespan of 493.90 seconds while Min-Min resulted in an application makespan of 494.684 seconds. Both

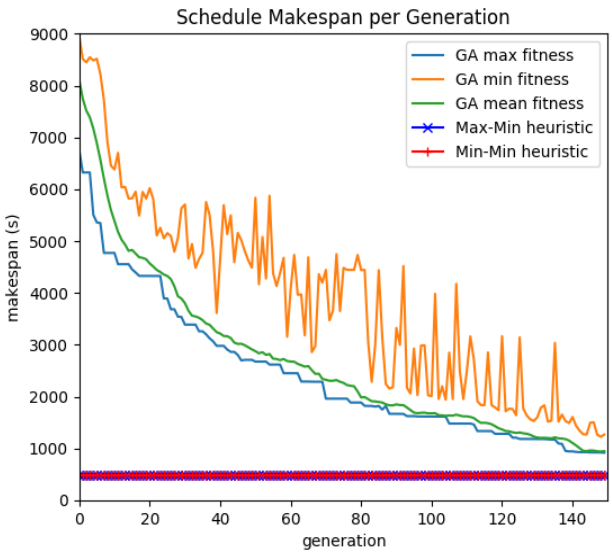

Fig. 3: Comparing results of the genetic algorithm against the Max-Min and Min-Min heuristics.

heuristics produced schedules that performed about twice as good as the best individual produced by the GA, shown in Figure 3. As the GA moved from generation to generation, it takes a longer period of time between generations for an improvement to be shown in the individual with the maximum fitness (shown by the slope of then blue line). It is also important to note that when the population as a whole begins to converge onto the hightest fit individual, there are some noticeable improvements in the hightest fit individual in generations that immediately follow. This can be seen around generations 20, 70, and 135. An implication of this could be such that if the GA ran for more than 150 generations, we would likely witness the population fitness converge onto where Max-Min and Min-Min currently stand.

VI. Conclusion

Based on the results in this study, we can conclude that Min-Min and Max-Min may perform well when given random applications and master worker infrastructures. Additionally, given enough iterations, the GA could produce a schedule that performs just as well, or better than these list scheduling heuristics. In practice, online and offine heuristics generally aren't compared, however in the case of scheduling parameter sweep applications, this can be okay as real world applications may run for days or months while the evolutionary algorithm may run for a fraction of that time thanks to simulation.

Both the GA and the heuristics have their advantages and disadvangtages. First, the genotype representation

for this specific problem scales in the size of the number of files used in the application along with the number of worker nodes. Real world parameter sweep applications have a number of files and tasks orders of magnitude greater than the number of worker nodes available. Using this specific GA in a real world scenario requires a genotype representation that could be thousands of times larger than what was used in this study. If the length of the genotype is N, then we end up with a search space that is $N!$. Thus scaling GAs for larger problems similar to this have been pointed out to be a challenge [6]. The search space in this study is $500!$ and the GA came close to the heuristics after only 30000 fitness evaluations, a minute portion of the total search space, suggesting that this could be a viable approach for smaller problems. Scaling aside, another issue with the GA is that it produces a static schedule. Static schedules do not account for things such as network or host failures and so online heuristics such as the ones tested here are more adaptable to real world systems.

To further explore the viability of this GA and its affectiveness on a range of application types and infrastructures, it will be useful to test a number of different application configurations. For example, one that is data intensive versus compute intensive. Furthermore, the topology of the bipartite application graph can take many different forms and so those must be tested as well. In addition, improvement can be made with the chromosome representation to decrease the search space size as some file, host mappings contribute nothing to the application makespan.

In conclusion, this GA could be useful for smaller problems, but the current set of heuristics available provide simple, easy to implement methods of achieving ”close to optimal” performance.

REFERENCES

[1] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, “Heuristics for scheduling parameter sweep applications in grid environments,” in *Proceedings 9th Heterogeneous Computing Workshop (HCW 2000) (Cat. No.PR00556)*, May 2000, pp. 349–363.

[2] H. Casanova, F. Berman, G. Obertelli, and R. Wolski, “The apples parameter sweep template: User-level middleware for the grid,” in *SC ’00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, Nov 2000, pp. 60–60.

[3] A. Giersch, Y. Robert, and F. Vivien, “Scheduling tasks sharing files on heterogeneous master–slave platforms,” *Journal of Systems Architecture*, vol. 52, no. 2, pp. 88 – 104, 2006, parallel, Distributed and Network-based Processing. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762105000767

[4] M. Maheswaran, S. Ali, H. J. Siegal, D. Hensgen, and R. F. Freund, “Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems,” in *Proceedings. Eighth Heterogeneous Computing Workshop (HCW’99)*, April 1999, pp. 30–44.

[5] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, “Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach,” *Journal of Parallel and Distributed Computing*, vol. 47, no. 1, pp. 8 – 22, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731597913927

[6] A. Wu, H. Yu, S. Jin, K. Lin, and G. Schiavone, “An incremental genetic algorithm approach to multiprocessor scheduling,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 9, pp. 824–834, Sept 2004.

[7] H. Casanova, S. Pandey, J. Oeth, R. Tanaka, F. Suter, and R. Ferreira da Silva, “Wrench: A framework for simulating workflow management systems,” in *13th Workshop on Workflows in Support of Large-Scale Science (WORKS’18)*, 2018.

APPENDIX

Note that the pseudocode below describes how Max-Min and Min-Min work but does not include implementation details in regards to how a task is executed or how files are sent.

Algorithm Max-Min

```
pending_tasks = application.getTasks()
while pending_tasks.size ≠ 0 do
  if canScheduleTasks() then
    ECTS = [|pending_tasks|][|workers|]                    # estimated completion times
    for t ∈ pending_tasks do
      for w ∈ workers do
        ect = estimate_completion_time(t, w)
        ECTS[t][w] = ect
      end for
    end for
    target_task = {t | max_{∀t∈pending_tasks} ( min_{∀w∈workers} (ECT[t][w]))}
    target_worker = {w | max_{∀w∈workers} (ECT[target_task][w])}
    assign(target_task, target_worker)
  end if
end while
```

Algorithm Min-Min

```
pending_tasks = application.getTasks()
while pending_tasks.size ≠ 0 do
  if canScheduleTasks() then
    ECTS = [|pending_tasks|][|workers|]                    # estimated completion times
    for t ∈ pending_tasks do
      for w ∈ workers do
        ect = estimate_completion_time(t, w)
        ECTS[t][w] = ect
      end for
    end for
    target_task, = {t | min_{∀t∈pending_tasks} ( min_{∀w∈workers} (ECT[t][w]))}
    target_worker = {w | min_{∀w∈workers} (ECT[target_task][w])}
    assign(target_task, target_worker)
  end if
end while
```