Job Scheduling in a Heterogenous Distributed Computing System using a Competitive Matchmaking System

David Mendez Lopez[1], **Fabian Garcia Nocetti**[1]
Applied Mathematics and Systems Research Institute, UNAM, Mexico City, Mexico
[1]mendezd.lopez@ciencias.unam.mx , [2]fabian.garcia@iimas.unam.mx

Abstract— *Every day services that requiere the use of distributed computing systems, like Grid and Cloud services, are getting more popular and necessary. One of the core functions of these systems is finding a good scheduling of computing jobs in order to minimize their execution time. However, scheduling jobs in distributed systems that are made from widely different processors and hardware is a very difficult problem. In addition, in recent years execution time is not always the metric we need to optimize in a scheduling. Cost, energy, bandwidth, response time, among others, are needed as well. Sub-optimal solutions are given, but these are based on predictions and estimates about performance and cost that not necessarily reflect a real-time execution. In other cases a heavy monitoring system needs to be implemented in order to get the necessary data for good schedulings. We present our solution, a competitive matchmaking system for job scheduling. Our goals are to: 1) provide sub-optimal solutions with data that reflects the current execution; 2) require no external performance monitoring system; 3) a flexible enough system that we can schedule jobs based on different metrics.*

Keywords: Distributed Computing, Job Scheduling, Resource Allocation, Competitive Matchmaking

1. Introduction

In the last decade there has been an increase in the popularity and necessity of high performance distributed computing services, like Grid and Cloud systems. This systems are implemented by several geographically separated data centers and autonomous systems connected by the internet. The machines used can be completely different in terms of speed, bandwidth and manufacturer. We call this a heterogeneous distributed computing system.

One of the core functions in this systems is the scheduling of user computer jobs to the processors that will execute them in a manner that the execution time of these jobs is minimized. Job scheduling is a very old and recurring problem in computer science. The problem has been approached by the Operating Systems and Data Base communities since the 1970s. In [15] it is defined as the distribution of limited resources (machines) among activities (jobs) so as to achieve certain objectives. More recently, with the growth of the distributed computing systems the problem is still a major issue engineers and scientist have to address. The difficulty of this problem comes with the fact that it is NP-hard in its most simple form, and NP-complete in several other variations [1], [6]. Thus we have to come up with sub-optimal algorithms that take us as close to the optimal solution as possible.

Currently there are many different approaches to this problem in distributed computing systems [8], [2], [5]. Traditionally, the only thing that mattered in a good schedule was the execution time of the jobs. Now different parameters must be taken into consideration while scheduling such as: economic cost of the execution, the priority of the user, the bandwidth of specific parts of the network, and minimum response time to the user.

In order to handle scheduling with this new parameters, preferred solutions require either prediction mechanisms, heuristic functions or robust monitoring systems[9]. Unfortunately, the predictions are based on data that may not reflect the current system's behavior, the heuristic functions get too complicated, or robust monitoring systems have to be implemented.

2. Related Work

Heuristic algorithms have been proposed as solutions since the 70s[7]. More recently, in [5], [12] several heuristics are used to achieve dynamic scheduling in heterogenous computing systems. Dynamic scheduling refers to having the ability to reschedule jobs to better machines. The problem here is that the heuristics used require complicated algorithms and data structures. It also makes predictions on machine behavior that don't necesarily reflect a real time execution. Our solution doesn't need any offline predictions about the system.

Dynamic scheduling has also been proposed as a toolset for computational Grids [3]. This approach is effective in improving performance but requires a monitoring system to collect the required measurements. Our system does not require the use of external monitors, it only requires data obtained after a job is complete.

Adaptive solutions have also been proposed like [4], [11]. This approach shows a system that can reschedule tasks based on the long term performance of the system. It is primarily intended for large applications that can stay in

execution for an indeterminate amount of time. The problem with this method is that it has to log the execution of all the jobs on all the machines, or again calculate complicated predictions and estimates of various system parameters. Our solution requires very little data to be stored into memory, at the very least it just requires one variable and a bounded number of lists of machine identifiers.

The algorithm we use is based on competitive matchmaking systems. This systems are used to calculate the expected result in a competition between two or more participants. They use a numerical rating to measure the ability of the competitors. The most popular one is the Elo model used in chess to measure the relative ability between players [13]. The model reflects the increase of skill between players, where there is a bigger change in rating if there is a bigger gap in skill, or a small change if the skill gap is low. The idea of our work is to look at machines in a distributed system as participants in a contest.

3. The Competitive Matchmaking Algorithm

The basis of the algorithm is looking at the execution of jobs in a distributed system as a competition between the distinct machines of the system. Winning in this competition means a machine completed a job below a predetermined performance threshold. We can use performance metrics like execution time, execution cost, response time, or similar. The scope of this paper only covers testing and implementation for the execution time metric.

Each machine is assigned a numerical rating that we are going to call CR *(Competitive Rating)*. This rating reflects the past performance of each machine. A higher CR means the machine has completed more jobs succesfuly in the past. At the beginning of the algorithm all machines have the same base CR.

We keep track of the machines' performance by grouping them according to their CR in a *Tier List*. The tier list is list of lists and it is handled by the job scheduler. Each tier corresponds to a list of machine identifiers with CR between a predetermined range. Each tier is assigned a number called *Tier Level*. Each machine knows the number of the tier where they belong. The number of tiers and the range of CR are system implementation dependent. In our implementation we chose tier level 1 as the fastest, ascending levels are slower. We want the number of tiers to be done in a way that each tier contains machines that have speed that is around a fraction of the maximum machine speed of the system. For example if lowest tier level is 5 we want that the machine in that tier have speed around 1/5 of the maximum machine speed.

Before the scheduling starts, jobs are given a requirement level, either by the user of the system or the size of the job. We call this *Job Level*. The levels go hand in hand with the tier number. This way the higher the job level the higher the tier the job needs. The job scheduler tries to assign the task to any available machine in the tier as the job level requires. If there are no machines available it tries to assign to the next best tier. A predetermined performance threshold is set for each tier. We call this threshold the winning condition. The details of the winning condition formula are explained in section 3.1. Each assigned machine will use their own CR, the tier level and the job level to predict the probability of achieving the winning condition. Everytime a machine completes a job it measures how well it performed (Ex. how long it took to complete the job). With this measurement and the prediction, it can now modify its CR. The prediction formula is explained in section 3.2.

We calculate the new CR the following way:

$$CR_n = CR_o + C * (S - S_e) \quad (1)$$

- CR_n: is the new competitive rating.
- CR_o: original rating.
- C: growth adjusting positive constant.
- S: result of the competition, 0 if winning condition was not met, 1 if it was.
- S_e: expected result. It will be a value between 0 and 1 (explained section 3.2).

This formula is based on the Elo formula for chess [13]. It reflects that if a machine wins the CR will increase, if it loses it will decrease. As the formula shows, the change depends on how far the expected value was from the result.

We update the new CR and update the *Tier List* at the job scheduler. If the CR drops below the current tier range, then the machine will be removed from the current tier and added to the one below. CR cannot go to a 0 value or less. Conversely if the CR grows above the tier range the machine will move to the upper tier. If the machine is already at the top tier the CR will increase but it will stay at the top tier. We can now schedule new jobs.

3.1 Winning Condition Formula

The winning condition is set for each tier. This way higher tiers have stricter conditions. We now give the winning condition formula using the job execution time metric. In our current test build the data we need is the following: the execution time of the job, the machine's tier level, the maximum machine speed of the distributed system and a programmer defined execution time overhead. The maximum machine speed of the system can be an estimation but this is normally something we can know beforehand. The execution overhead is used as variable to adjust how strict the win condition can be.

$$WinCond = \frac{(MAXSPEED - overhead)}{tierlevel} \quad (2)$$

If the execution time is less or equal than $WinCond$ then we can decide that the machine won.

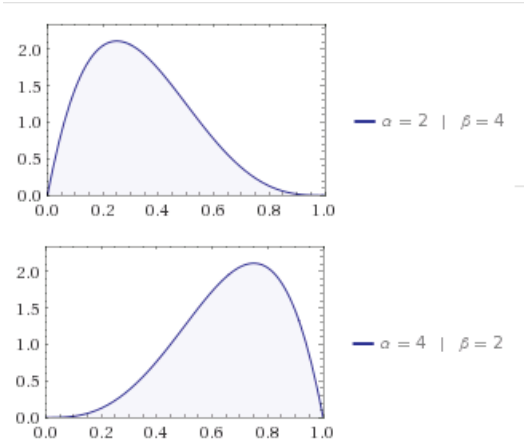

Fig. 1: An example of two beta distributions one with $\alpha = 2$ and $\beta = 4$, the other $\alpha = 4$ and $\beta = 2$. The first shows a distribution where losing is the most likely outcome, the second shows one where winning is the most likely outcome.

3.2 Prediction Formula

We want each machine to calculate the probability of it completing the job below or above the performance threshold. We use machine tier level and the job level to calculate. We utilize a beta distribution to derive the formula we use. The beta distribution is a family of probability distributions defined in the [0,1] interval. It is parametrized by two positive values denoted as α and β. This distribution is useful because it can model the behavior of random variables defined between two values [14]. In our case the two values we need are lose and win (0 and 1). It is also a good model that allows for variability in the probability of success among individuals [10]. This also closely models the behavior of jobs executing in a given machine and how likely they are to be completed, mentioned in [16].

In (3) we show the formula of the beta distribution, where θ is the random variable. We use the expected value of the probability distribution that is easily calculated as shown in (4).

$$P(\theta|\alpha,\beta) \propto \theta^{\alpha}(1-\theta)^{\beta-1} \tag{3}$$

$$E(\theta) = \frac{\alpha}{\alpha+\beta} \tag{4}$$

We choose 2 different beta distribution families. One using $\alpha = 2$ and $\beta > \alpha$, the second using $\beta = 2$ and $\alpha > \beta$. An example of the shape of both distribution families is shown in Figure 1.

The parameters α and β are chosen the following way:

- Case Job Level $\leq$ Tier Level: We call this positive perspective and so $\alpha = 2+1+|TierLevel-JobLevel|$ and $\beta = 2$.

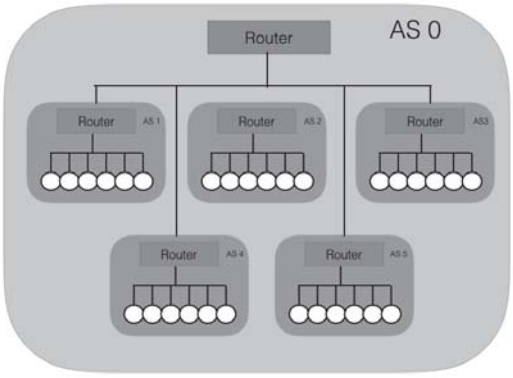

Fig. 2: A hierarchical Autonomous System based network

- Case Job Level $>$ Tier Level: We call this negative perspective and so $\alpha = 2$ and $\beta = 2+1+|TierLevel-JobLevel|$

4. Experiments

Testing and experimentation is currently underway and several scenarios are still untested.The simulation test suite is still under development. Only job execution time metric has been tested. We have tested our competitive algorithm against a random scheduling algorithm and against the min-min heuristic algorithm The min-min heuristic algorithm tries to schedule the minimum size job to the machine with the minimum expected job completion time. The experiments are being made using the distributed system simulation platform called SimGrid 3.14. In particular we use the SimDag framework, which provides functionalities to simulate parallel job scheduling. Jobs are represented using directed acyclic graphs, and also reprecedent precedence constraints and data movements between jobs.

We have made simulations of 30 and 60 machine systems, representing a centralized hierarchical autonomous system network used in several Grid and Cloud systems [8] as shown in Figure 2.

Machine speeds fluctuate between 5 and 9 Mflops, and network bandwidth is constant since only job execution time is being measured right now. $MAXSPEED$ was set at 9Mflops. The number of tiers is 5 with a 500 CR range per tier. The growth constant for the CR formula was set at 20. In Fig. 3 we show the result of our algorithm compared with a the min-min heuristic algorithm and a uniformly random scheduling algorithm measuring the execution time of sets of 10, 15, 20, 40, 60 independent jobs with execution times varying between 1000 and 9,000,000,000 operations.

5. Preliminary Results and Discussion

We developed a new scheduling algorithm and have shown it is comparable to other heuristic based scheduling algorithms. This algorithm can still be improved. We assume we

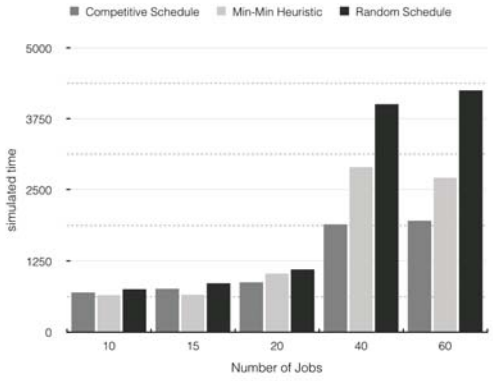

Fig. 3: Comparison on total job execution time between our competitive algorithm, the Min-Min heuristic algorithm and random assignment.

know beforehand data about the system like the maximum machine speed and the size of the jobs. In practice this information is not always available. This assumption comes from the fact that the systems we are modelling resemble Grid and Cloud systems. A setback this method clearly has is that the CR of each machine has to converge to a certain value before we can have a precise tier list of machines.

The use of a beta distribution in calculating the expected performance of a machine is still a very little researched subject. We feel that the use of this distribution can be more precise. As we gather more data about machine performance the distribution families we use can be more refined.

Other test cases are being examined with bigger job sets, with different levels of job dependence, and more constrains on job sizes. We are also looking into testing dynamic job scheduling with this method so we can handle rescheduling jobs to better machines as they become available.

Different heuristics are planned to be tested such as the ones referenced in [5].

Since this scheduling algorithm works better the longer it is running, we expect the system will show better results in long term applications.

References

[1] Horowitz E.,Sahni S., *Exact and approximate algorithms for scheduling nonidentical processors*. , Journal of ACM 23,3., 1976.

[2] Casavant, T., Kuhl J., *A taxonomy of scheduling in general-purpose distributed computing systems*., IEEE Transactions on software engineering 14.2, 1988.

[3] Costa G., Sikora A., Jorba J., Margalef T., *GMATE: Dynamic Tuning of Parallel Applications in Grid Environment*, Journal of Grid Computing, Volume 12, Issue 2, pp 371-398, Springer, 2014.

[4] Wu M., Sun XH, *A general self-adaptive task scheduling system for non-dedicated heterogenous computing*, Cluster Computing, 2003. Proceedings. 2003 IEEE International conference on. IEEE, 2003.

[5] Maheswaran M., Ali S., Siegel J., Hensgen D., Freund R., *Dynamic Mapping of a Class Independent Tasks onto Heterogeneous Computing Systems*, Journal of Parallel and Distributed Computing 59.2, 1999.

[6] Ullman J. D., *NP-Complete Scheduling Problems*, Journal of Computer and System Sciences 10, 1975.

[7] Ibarra O., Kim C., *Heuristic algorithms for scheduling independent tasks on nonidentical processors*, Journal of the ACM 24(2), 1977.

[8] Hameed Hussain, Saif Ur Rehman Malik, Abdul Hameed, Samee Ullah Khan, Gage Bickler, Nasro Min-Allah, Muhammad Bilal Qureshi, Limin Zhang, Wang Yongji, Nasir Ghani, Joanna Kolodziej, Albert Y. Zomaya, Cheng-Zhong Xu, Pavan Balaji, Abhinav Vishnu, Fredric Pinel, Johnatan E. Pecero, Dzmitry Kliazovich, Pascal Bouvry, Hongxiang Li, Lizhe Wang, Dan Chen, Ammar Rayes, *A Survey on Resource Allocation in High Performance Distributed Computing Systems*, Journal of Parallel Computing, 2013.

[9] Qureshi M.B., Hussain H., Dehnavi M.M., Qureshi M. S., Rentifis I., Tziritas N., Xu C. Z., Loukopoulos T., Khan S. U., Zomaya A.Y. *Survey on Grid Resource Allocation Mechanisms*, Journal of Grid Computing, Springer, 2014.

[10] Schuckers E. Michael, *Using Beta-binomial distribution to assess performance of a biometric identification device.* International Journal of Image and Graphics 3.03 (2003):523-529.

[11] Saleh, A.I., Sarhan, A.M., Hamed, A.M., *A new grid scheduler with failure recovery and rescheduling mechanisms: discussion and analysis*. Journal of Grid Computing 10(2), 2012.

[12] Thaman J., Singh M., *Current Perspective in Task Scheduling Techniques in Cloud Computing: a Review*, International Journal in Foundations of Computer Science and Technology 6(1), 2016.

[13] Elo Arpad *The Rating of Chessplayers, Past and Present.* ISBN 0-668-04721-6, 1978.

[14] Navarro D. Perfors A., *An introduction to the Beta-Binomial model*, Computational Cognitive Sciencelecture Notes. University of Adelaid, https://www.cs.cmu.edu/Ĩ0701/lecture/technote2_betabinomial.pdf.

[15] Wu T., Ye N., Zhang D., *Comparison of distributed methods for resource allocation.*, International Journal Prod. Res 43(3) pp 515-536, 2005.

[16] Harchol-Balter M. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge Universtiy Press, February 2013.