Job-Scheduling with Resource Availability Prediction for Volunteer-Based Grid Computing

Jun Zhang† and Chris Phillips†

† Queen Mary, University of London

Abstract: In a volunteer-based grid computing environment, one big challenge for effective job allocation is resource availability. As resources in this environment are volatile and may become frequently unavailable, matching guest jobs to suitable resources is very important. To improve scheduling performance in such an unreliable computing environment, especially in terms of avoiding job completion failure due to resource unavailability, this paper proposes a new job-scheduling algorithm. This scheduling algorithm is based on an existing resource availability prediction technique that anticipates future availability of resources to help make better job allocation decisions. Simulation results shows this new job scheduling algorithm provides better results in terms of ensuring jobs are processed successfully.

1. Introduction

Grid computing has evolved to a stage that it can comprise many heterogeneous computing resources owned by different individual users and institutions. This brings benefits for large-scale computing projects as more computer resources are available to exploit. On the other hand, as the owners of the resources usually have the right to decide when and how to donate idle CPU cycles of their computers, these resources may be volatile; they may appear and disappear at any time.

The volatility of computing resources brings a big challenge for allocating jobs effectively. If a job is allocated to a resource that becomes unavailable before finishing the processing of the job, then this job will be suspended or may even fail, requiring the job to be sent to another resource for processing again from the start. This is a waste of resource CPU cycles and it lengthens job's *makespan*, i.e. the time to "make" or complete a job.

In general, there are two approaches to solve the problem of job process failure caused by resource unavailability. The first one is to change the computing resource behaviour to make them more available and stable. However, as mentioned above, these computing resource are owned by different users and institutions and they usually have rights to decide when and how to donate their computing resources. It is therefore difficult to achieve the necessary control. The second and the more practical approach is to recognise what will happen to the computing resources and make reasonable job allocation decisions accordingly.

To anticipate what will happen to the computing resources, some prediction techniques have to be proposed. In [1][2], the authors propose a technique for predicting resource availability. According to their simulation results, their technique is more accurate than many existing prediction techniques. To improve scheduling performance in volunteer-based grid computing environment, especially in terms of avoid guest jobs process failure caused by resources' unavailability, our paper proposes a new job scheduling algorithm based on this resource availability prediction technique.

According to [3], for a job-scheduling algorithm, there is a trade-off between speed, minimising average job makespan, and reliability, minimising the number of job failures caused by resource unavailability. This means that it is impossible to achieve both objectives at the same time in most cases. As extensive research has been carried out in minimising the average job makespan, the scheduling algorithm proposed in this paper focuses on the second objective.

The rest of the paper is organised as follows: Section 2 reviews some related work, especially the background to the prediction technique proposed in [1][2] and adopted within this paper. Section 3 describes the prediction technique proposed in [1][2]. Section 4 introduces the proposed job scheduling algorithms. Next, Section 5 presents our experiment approach and results. Finally, conclusions are given in Section 6.

2. Related Work

Considerable attention has been paid to job scheduling and many well known approaches have been proposed, e.g. First-Come-First-Served (FCFS), Earliest-Deadline-First (EDF) and Backfilling [4]. However, these traditional scheduling approaches lead to poor grid schedulers as many assumptions that hold for traditional scheduling scenarios cannot be applied in the context of grid computing [5]. Therefore, much research has also been devoted to job scheduling in a grid computing environment.

The most related work in grid computing is that undertaken by Brent Rood and Michael J.Lewis' [1][2]. In their approach, they propose a multi-state model to describe a resource's availability and they use this model with their proposed prediction techniques. Based on these techniques, they propose a job scheduling approach named Production Prediction Score (PPS) Scheduler. They claim that their prediction results are better than many existing techniques. Therefore, we adopt one of their prediction techniques and propose a new job-scheduling algorithm. Further details are given in Section 3.

Another related work is that of Derrick Kondo *et al* [6][7]. In their work, they propose a number of resource prioritisation/exclusion methods for resource selection in a grid computing context. Some of these methods simply use static information, e.g. resources' clock rate to priorities resources and the some use resources' past performance to predict its future performance and prioritise the resources accordingly. Based on the prioritisation results, the job scheduler allocates jobs to the resource that has the highest priority.

Aleksandar Lazarevic *et al* [5] propose a new deadline-scheduling algorithm, called Latest Time To Run first (LTTR). LTTR was motivated by extensive research in the scheduling of the real-time systems using the EDF algorithm and, in essence, it is a modified version of EDF. In LTTR, users not only submit the deadline of their jobs, but they also submit the estimated execution time of their job. With this extra information, LTTR calculates the difference between the job requested deadline and the predicted job run time; it then sorts the job queue in ascending order. Later, the first job in the queue will be allocated to the first available idle resource.

3. Prediction Techniques

In this section, some background to the prediction technique adopted in this paper is briefly described, regarding the work of Brent Rood and Michael J.Lewis [1][2]. As mentioned in Section 2, they proposed a multi-state model to describe a resource's state. These states are: Available to Grid, User Present, CPU Threshold Exceeded, Job Eviction or Graceful Shutdown and Unavailable. With this resource availability model, they then consider a number of multi-state prediction algorithms. In brief, a multi-state algorithm works as follows: it takes a length of time as an input and uses a resource's availability history to predict the probabilities of that resource next exiting the available state into each of the non-available states, and to remain available throughout the interval. These probabilities sum to 100%. The predictor outputs four probabilities, one each for entering: User Present, CPU Threshold Exceeded and Unavailable state next and one for the probability of completing the interval without leaving the Available state.

To calculate these probabilities, they employ several techniques and the one used in this paper is Transitional N-Day with Equal transition weights (TDE). "Transitional" means the prediction technique calculates the output probabilities by counting both the number of transitions from Available to other states and how many times the job could be processed between two transitions. "N-Day" means checking the most recent past N days' transitions. "Equal transition weights" means the prediction technique considers each transition within the different checking days equally. According to their results, TDE is the most successful technique among all the prediction techniques they tested when the prediction length is no longer than 42 hours, especially when it is shorter than 19 hours. According to research in [5], the job execution time in a grid context is typically less than 10^5 seconds (around 27.8 hours), so TDE was adopted in this paper to improving the performance of our job-scheduling algorithm.

4. Proposed Job-Scheduling Algorithm

As discussed above, resources' unavailability is a big challenge in grid computing environment. We propose a job-scheduling algorithm based on the prediction technique described in Section 3. This algorithm makes the following assumptions: Firstly, there is a centralised job scheduler in the grid and users submit their self-contained executable jobs to the job scheduler. When the job arrives at the job scheduler, the job scheduler puts the job into a job queue. Later, the job scheduler decides where to allocate these jobs. Jobs can fully exploit resources' CPU cycles donated to the grid. After execution, resources return the results to the original users that submitted those jobs. The job scheduler is assumed to know the execution time for each job before making allocation decisions.

In general, the process of a job-scheduling algorithm can be summarised as follows:

1. When a job is submitted to the job scheduler, the job will be put onto a queue and sorted by a scheduling method, e.g. FCFS or EDF.
2. At regular interval, if the job queue is not empty and idle resource(s) exist(s) in the grid, the job scheduler tries to find a suitable resource for the first job in the job queue. It uses the TDE prediction method to calculate the reliability of the first idle resource. Here, reliability is the probability of completing the prediction interval without leaving the available state calculated by TDE.
3. The job scheduler makes a job allocation decision based on this result. If the resource is considered to be suitable - reliability is over a predefined threshold, the job scheduler will allocate the job to that resource. If

not, the job scheduler will try to find an alternative idle resource and repeat step 2 and 3. If there is no suitable resource available, the job scheduler will hold the job for a period of time before returning to step 2.

In step 2, the job scheduler only looks for an available resource that is idle. This is to avoid the case of allocating many jobs to a particular resource as this will slow down the job that is already running on that resource and all jobs allocated to this resource will fail to complete when that resource becomes unavailable. In addition, rather than comparing all idle available resources before allocating a job, the job scheduler checks one resource at a time. This is to reduce the computation cost and speed up allocation decisions. For example, if many candidate resources are available at the same time, calculating each resource's reliability and comparing them could consume considerable time. Therefore, to accelerate job allocation decision speed, the scheduler looks for a suitable resource, which is not necessarily the best.

In step 3, not allocating jobs to resources whose reliability is lower than a pre-determined threshold reduces the risk of a job failing to complete.

5. Simulation Results

To observe the performance of our job-scheduling algorithm, we run a series of simulations and compare our results with the job scheduling algorithms PPS proposed in [1]. PPS with prediction, uses its own equation to calculate the availability score for each resource and select the one with the highest score. The equation is:

$$RS(i) = C_i \cdot M_i \cdot (1 - L_i) \tag{1}$$

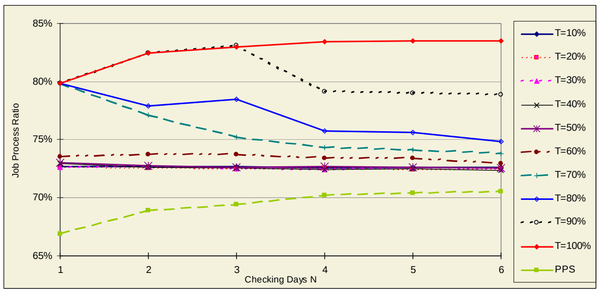
where C_i is the resource's predicted completion probability during the application's expected execution interval as given by the predictor, M_i is it's MIPS score, and L_i is the CPU load currently on resource i.

The UCB grid resource availability traces were downloaded from http://xw01.lri.fr:4320/dg/. Each file records the number of CPU operations delivered to the grid between different epoch times. If the value of CPU operations delivered is below 0, then that means the resource is not available. According to [6][7], unavailable gaps of less than 2 minutes can be neglected as these gaps are caused by their measurement method, and are not real periods of unavailability. Therefore, these gaps are neglected in our simulations. In each run, trace data for seven days worth of resource availability are used from a particular set. Jobs are submitted to the job scheduler at regular intervals and job execution times follow an exponential distribution ranging from 1 second to 10^4 seconds. The average job execution time is about 10^3 seconds. As with [1], the prediction length is three times the job execution time. For example, if a job lasts for an hour from 8am to 9am, then the prediction algorithm will check resources' transitions for three hours - from 8am to 11 am in the most recent past N days. The job scheduler sorts the job in the queue according to FCFS order.

Two important variables were tested in the simulations: the first one is the reliability threshold, T, used for job allocations. As discussed in Section 4, the job scheduler makes allocation decisions based on a reliability threshold. So if the reliability threshold is set too low, then jobs may be allocated to resources which are not very reliable and they cannot be processed before the resources become unavailable. On the other hand, if the reliability threshold is too high, then a lot of idle CPU cycles may be wasted and jobs' makespan time may become very long. The second one is the number of checking days, N, used by prediction algorithm. The prediction techniques check resources' most recent past N days' states transitions to make a prediction, so if N is too small, then prediction results may not be accurate and the scheduling results may be affected.

Figures 1 and 2 were plotted by using average values of five replications of each data point. Figure 1 shows job process ratio (overall number of jobs processed successfully by resources divided by overall number of jobs processed successfully plus overall number of jobs that failed to be processed) for different reliability threshold values, T. In general, our job-scheduling algorithm performs better than PPS. In PPS, the job scheduler allocates jobs to the resource which has highest score after comparisons, no matter whether the resource is idle or not. So more than one job may be allocated to the same resource. This can lead to many jobs failing to complete by the time the resource becomes unavailable. In the results of our scheduling algorithm, when T is lower than 50%, the job process ratio is quite stable. That is because resource has a reliability of over 50% in most cases, so the job scheduler will allocate the first job in the queue to the first available idle resource. It behaves more like a round-robin algorithm when T is lower than 50%, but when T is over 50%, job process ratio becomes higher along with T. The highest job process ratio is 83.43% when N is 6.

Figure 2 shows the job process ratio for different checking days N. Though N varies from 1 to 6, the results have similar trends and similar results. When the probability threshold is lower than 50%, the job process ratio is quite stable, varying from 72.6% to 72.7%. The job process ratio increases when T is over 50% and reaches the peak when T is 100%. Interestingly, larger values of N bring better results when T is 100% but larger N does not necessarily bring better results when T is between 50% and 90%. For example, "N = 1" has the highest performance when T is 70% and "N = 3" has the highest result when T is 90%. That is because of the calculation methods used by the prediction technique. For example, when the reliability threshold is 70% and if N is 1 and the first day has reliability 40%, then this resource is unreliable in this case. But if N is 2 and the second day has reliability 100%, then the overall reliability will be (40% + 100%) / 2 = 70%, so this resource will consider as reliable in this case. Therefore, this causes the differences when T is between 50% and 90%. But when the reliability threshold is 100%, it requires each day to be 100%. As a result, the deviation caused by observations over different days disappears.



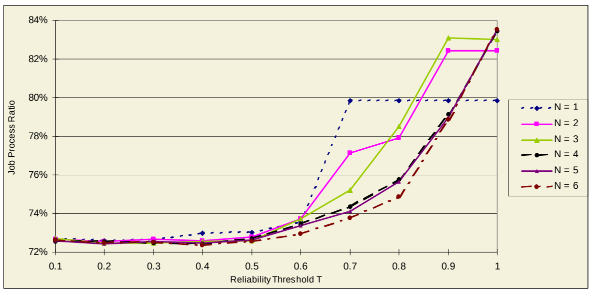


Figure 1. Job Process Ratio for Different Reliability Thresholds T

Figure 2. Job Process Ratio for Different Checking Days N

6. Conclusions

In this paper, we propose a job-scheduling algorithm. This algorithm is aimed for ensuring jobs process successfully in unreliable grid computing environments. According to the simulation results, this algorithm performs better than PPS. The results also indicate that checking more days does not provide better prediction performance for job scheduling.

Acknowledgments

The authors thank Dr. Derrick Kondo for the downloadable desktop grid trace (http://xw01.lri.fr:4320/dg/)

References

[1] Rood, B. Lewis, M.J., Proceedings of eScience, 2008. eScience '08. IEEE Fourth International Conference, ISBN: 978-1-4244-3380-3

[2] Rood, B. Lewis, M.J., Proceedings of Grid Computing, 2007 8th IEEE/ACM International Conference, ISBN: 978-1-4244-1560-1

[3] A, Dogan. F, Ozguner., Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems, 13(3):308–323, 2002.

[4] David A, Lifka., The ANL/IBM SP scheduling system, Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, volume 949 of Lecture Notes in Computer Science, pages 295–303, London UK, 1995. Springer-Verlag.

[5] Lazarević A, (2008) Autonomous grid scheduling using probabilistic job runtime scheduling. Doctoral thesis, University of London.

[6] Derrick Kondo, Dissertation, University of California, San Diego, Scheduling Task Parallel Aplications For Rapid Turnaround on Desktop Grids, La Jolla, CA 92093, 2005.

[7] Derrick Kondo, Gilles Fedak, Franck Cappello, Andrew A. Chien, Henri Casanova, Resource Availability in Enterprise Desktop Grids, Journal of Future Generation Computer Systems, 2007, ISSN:0167-739X