Fault-aware Scheduling in Grid Environment based on Linear Programming

Mehdi Sarikhani, Bahman Javadi, Askari Parichehre
Islamic Azad University of Qazvin, Iran
Sarikhani.mehdi@Gmail.com
Javadi@aut.ac.ir
apmastermail@gmail.com

POSTER PAPER

ABSTRACT

Grid system provides the sharing, selection and aggregation on distributed autonomous resources while it is an error prone environment. So, grid component like scheduler must provide the user's Quality of Service (QoS) requirements by selecting the appropriate resources for user's jobs. In this paper, we have proposed a fault-aware economy scheduling model based on binary integer programming (FES-BIP) to allocate resources to application jobs such that the users' requirements are met while we know the distribution of resource failure in the grid environment. FES-BIP algorithm guarantees optimal resource selection.

KEYWORDS: Quality of Service, economy model, fault-aware scheduling, binary integer programming.

1. INTRODUCTION

Grid becomes more and more popular in large-scale computing, because they enable the sharing of computing resources that are autonomous resources and distributed throughout the world. We assume the market-based grid should consider the economy factors as a QoS user requirement. In market-based mechanisms, time, cost, and number of Processing Elements (PE) which a user needs are the important QoS factors. Consequently, the scheduling algorithms try to meet QoS factors and to optimize these factors. Grids are more error prone than traditional parallel machines as there are potentially thousands of resources that are heterogeneous and sharing among various applications [12]. So fault tolerance is an important characteristic in grid as the dependability of individual grid resources may not be guaranteed and resources are autonomous. It becomes increasingly difficult to guarantee that a resource being used is not malicious in some way [17]. So providing appropriate resource selection is vital to use grid systems perfectly. Also resource selection and scheduling in grid environment is always a controversial subject especially when fault tolerant scheduling is crucial due to the lots of failure. Therefore, the fault tolerant scheduling based on economic factors is so complicated (NP-hard) and finding the optimal answer is so difficult. There are several approaches to deal with scheduling but one of the best approaches to prove the scheduling result which is affirmative and optimal is using mathematical model and especially linear programming model. So by modeling fault-aware scheduling system which can provide optimal scheduling results, the scheduler can meet the user's QoS requirement and prepare fault tolerant grid resource. Albeit linear programming scheduling model has been used in previous works to solve scheduling problems, all of them work only on resource and job but obviously such models cannot prepare a real grid model[9][10]. Because they have to assume that all resource PEs has same cost and computational power. They can't take any decision about the tasks of jobs, in particular about the length of tasks and have to take all tasks with the same length. So it is crucial to put another parameter into model which is the resource PEs information, having assumed that the number of PEs in each resource (machine) and cost of each PE per second is possible. Our proposed scheduling algorithm by adding such information about PEs of resource on economy grid meet requirement of the time, cost, and number of PEs which users need. In addition it optimizes the cost and ensures reliability.

The rest of the paper is organized is as follows. In section 2, we review some related works. In section 3, we put forward our linear programming model. In section 4, we proposed the algorithm to solve the problem. We present the experiments and discuss the results in section 5. Finally, we conclude the paper and outline future lines of investigation.

2. RELATED WORK

Because the scheduling algorithms are dominant parts of grids environment, there are lots of algorithms which have been proposed in this area of research.

The scheduling algorithm can be divided in to two broad categories of knowledge-free and knowledge-based approach [6]. Canonico [6] in his knowledge-based approach took 3 states of information for scheduling which is availability awareness, computational power or both of them. Further, he used replication and checkpoint to provide fault tolerant scheduling. Fault-aware scheduling is one kind of knowledge-based scheduling which Verboven, Lan and Sun used. Verboven [19] recommended a two-phased fault-aware scheduling strategy, which are firstly dispatches the job to resources when the resource up-time is more than job execution time until all resource queue have just one job. In second phase, all resources are ordered according to the amount of computing time which is left, then that job is assigned to a resource with lowest remaining execution time while it's up-time is more than approximate execution time. Similarly, Lan [12] proposed a failure-aware resource selection framework which is intended to identify an optimal resource for a given application by considering the reliability characteristics of the available resources. He contends that an intelligent resource selection can save tremendous overhead that may be introduced by job migration or rescheduling after job allocation. So he sets forh a resource selection framework which on the basis a special formula calculates the minimum expected job completion time on all resources. At the end, it selects the resource that provides the minimum expected job completion time. A similar fault-aware structure that has three important parts including fault prediction, fault management, and system support which evaluate the time completion and execution of jobs by some probability equations was presented by Sun [18].

A scheduler's strategy varies according to time of job dispatching. As such we have two varies of job dispatching time, namely, immediate and batch mode. In the immediate mode, a task is assigned to a resource as soon as it arrives at the scheduler and this decision will not change once it is computed. On the contrary, in the batch mode, tasks are not mapped onto the resources as they arrive. Instead, they are collected into a set and the scheduler computes some parameters as a pre-scheduling step. Some classical scheduling algorithms are based on these two categories as presented by Maheswaran [14]. As regards the immediate mode there are several algorithms such as first-come-first-service, minimum completion time, minimum execution time, switching algorithm, k-percent best and opportunistic load balancing. Concerning batch mode we have Min-Min, Max-Min and Suffrage types [14].

Another strategy for grid scheduling is using replication and checkpoint which utilizes a lot in traditional distributed systems. Nazir [16], for example, applied fault tolerant job scheduling by keeping the history of resources. For resources that had several failures in their previous jobs and according to the number of failures the scheduler took some checkpoints. Anglano [2] also by adding replication and checkpoint to work queue algorithm presented WQR-FT algorithm that by taking checkpoint on replica of task and restarting from last checkpoint and producing a replica when another replica failed, improved WQR algorithm. Divide-and-conquer is another approach that divides a job into smaller tasks until it can solve these smaller tasks more easily, so at the time of task failure, it can redo these tasks. Owing to the fact that these tasks are smaller the redundant computation is also smaller. In addition, by storing partial results, this approach can be improved [20].

The market-based or economy-based model in grid is a very applicable approach. The market-based grid should consider the economy factors as a QoS user requirement. There are a number of market-based models which can be used in grid environment such as the commodity market, the posted price, the bargaining, the tendering/contract-net, the auction, the bid-based proportional resource sharing, the community / coalition / bartering and the monopoly/oligopoly model [3]. Deadline and budget are the most common economy factors as QOS user requirements. There are several algorithms for minimizing the cost, time or both of them. Venugopal [18] presented an scheduling algorithm for distributed data intensive Bag-of-Task applications using a deadline and budget constraint. This algorithm built a resource set for a job that minimized the cost or time based on the user's request.

A scheduling is an NP-hard in which finding an optimal result is of import. In this regards, genetic algorithm is the one which can serve the propose of finding such an optimal result. For example, Lee [13] introduces the architecture of fault detector, fault manager and resource manager which used a genetic algorithm to select the optimal result. The scheduling approach which similarly can prove the optimality of scheduling result while has not the problems of genetic algorithm, however, suffers from the problem of large search space and huge time consumption. As such, linear programming can be offered as an alternative to find the optimal result of scheduling which has been proved not to the aforementioned problems. There have been a number of studies in literature which have utilized linear programming in their approach to scheduling. For example, Dogan [8], Garg [10], Feng [9] all used linear programming. But they were limited on way or another. Dagan's [8] used meta scheduling algorithm to achieve optimal solution in scheduling problems. However it was limitation in the sense that his modeling rested on the assumption of one task for each independent application with many concurrent users. Although Feng [9] could provide a cost and time optimization by combining time and cost as optimization parameter, the approach taken to combination of these two variables does not appear to be legitimate. In a similar vein, Garg [10] proffered a linear programming cost model which is for space and time share resources. However, his heuristic solution to solve the linear programming was based on the limited assumption that all PE resources have same cost and computational power and that all tasks have the same length. In addition, the heuristic solution provided as a genetic algorithm suffers from the problems mentioned above.

In order to deal with the preceding limitations, we have the FES-BIP model which builds on real grid environment. FES-BIP is a fault-aware economy scheduling model which rest on binary integer linear programming with the advantage of an increased attention to resource information.

3. PROBLEM DEFINITION

3.1. System model

Grid users submit their application jobs to the Scheduler with their QoS requirements. The QoS requirements consist of budget, deadline, number of processing elements that user required and size of each job with unit of MI (million instructions). Each application job consists of independent tasks with each task requiring one PE. Each resource includes a number of heterogeneous PEs meaning that the PEs can have different characteristics like Computational Power, Cost, etc [10]. Figure 1 shows the interaction of the Scheduler (Meta Broker) with resource providers and users.

Two types of approach can be considered for job:

1. Space share: every task of a job must be assigned to PEs of single resource.
2. Time share: tasks of each job can assign to each PEs of each resource. So task of a job can assign to PEs of different resources.

In this paper, we supposed that works on space share jobs. The Scheduler gathers information about resources such as number of PEs available, cost of PEs per unit time for each user, PEs computational power with a unit of Millions of Instructions per Second rating (MIPS) and means time to repair of each resource (MTTR), from the resource providers who have agreed to rent their computational resources to Scheduler. The Scheduler run scheduling program then assign the job or in other word assign the task to PEs in appropriate resource. At the end, gathering the information of finished and failure job. For failure job updated the QoS requirements then if possible for them, run jobs again with new QoS requirements [10].

3.2. Problem formulation

Minimize job completion time and cost is the goal of the Scheduler with m resources and n jobs.

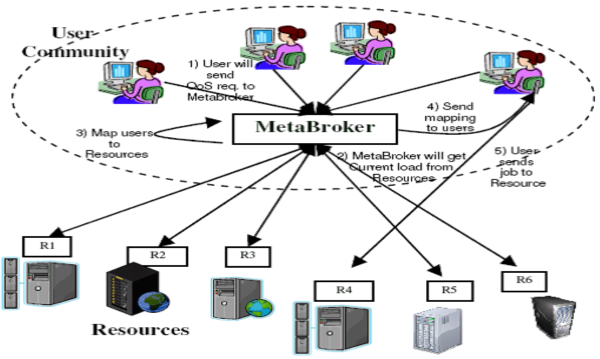
User Community
1) User will send QoS req. to Metabroker
MetaBroker
4) Send mapping to users
3) Map users to Resources
2) MetaBroker will get Current load from Resources
5) User sends job to Resource
R1
R2
R3
R4
R5
R6
Resources

Figure 1.Grid Model with Scheduler (MetaBroker)[10]

The Scheduler receives information from both users and resources. We consider this information into following parts:

1. The information that a Scheduler receives from a resource at instance time T in grid environment.

$$P_i : \{i, n_i, c_{ik}, v_{ik}, MTTR_i\}$$

i : Resource ID

n_i : Number of available PEs in resource i.

c_{ik} : Cost of PE number k in resource i consumption per second for any tasks of any jobs.

v_{ik} : Computational power of PE k in resource i with million instructions per second (MIPS) unit.

$MTTR_i$: Mean Time To Repair for resource i. each resource has different MTTR, maybe same distribution, we supposed Weibull distribution.

$$i \in I = \{1...m\}, k \in K = \{1...n_i\}$$

2. The QoS information that a Scheduler receives from a user at instance time T in this environment.

$Q_j = \{j, u_j, b_j, d_j, M_t, m_j\}$

j : Job ID

u_j : User ID

b_j : Budget constraint that user determined for jth job

b_j : Deadline constraint that user determined for jth job

M_t : Size of each task of job with unit of million instructions (MI). Each job has t tasks means that the number of tasks is equal to t for each job.

m_j : Number of PE that user need for job execution.

$j \in J = \{1 \ldots n\}$

The mathematical System modeling for space Share resources for cost optimization are:

Objective function:

$$\sum_{i=1}^{m} \sum_{k=1}^{n_i} \sum_{j=1}^{n} x_{ikj}.c_{ik}.(\max_t (M_t / v_{ik})) \qquad (1)$$

The variable x_{ikj} is a binary variable and denoted that whether job j assigned to Processing Element K on resource i or not. The $[\max_t (M_t / v_{ik})]$ denotes the maximum execution time of each task on every PEs. Because each task execute on every PE in paralleled manner, so for finding the time of a job execution, we must find the maximum execution time of tasks. Generally, Equation (1) calculates the cost of job j which executed in PE k on resource i.

Constraint:

$$\sum_{i=1}^{m}\sum_{k=1}^{n_i} \max_t (M_t / v_{ik}) + s_i < d_j , \quad \forall j \in J \qquad (2)$$

$$\sum_{i=1}^{m}\sum_{k=1}^{n_i} (M_t / v_{ik}).c_{ik} < b_j , \quad \forall j \in J \qquad (3)$$

$$\sum_{j=1}^{n}\sum_{k=1}^{n_i} x_{ikj} < n_i , \quad \forall i \in I \qquad (4)$$

$$\sum_{i=1}^{m}\sum_{k=1}^{n_i} x_{ikj} = m_j , \quad \forall j \in J \qquad (5)$$

Equation (2) denotes the time Constraint that time of execution and sum of MTTR of resource must be less than the deadline of the job. The $s_i = \sum_{i=1}^{m_j} MTTR_i$ indicates the sum of MTTR of all PEs which used. Equation (3) denotes the budget Constraint that cost of PE and duration of using this PE must be less than deadline of the job. Equation (4) explain the resource‘s capacity constraint which resource‘s capacity must be greater than number of PE that is need. Finally, Equation (5) shows the PE requirement of user‘s job.

4. PROPOSED ALGORITHM

In our proposed model, the scheduler can be aware of both availability and computational power of the resources, enabling it to decide if a task can be executed on a specific resource without failure. It's obvious that by having full information about the resources, we can calculate the execution time and approximate failure time of resource, so that we can determine whether the task completion time is before the fault event or not [6].Our FES-BIP algorithm are shown in Figure 2. We used branch-and-bound algorithm to search for optimal solution and solve this binary integer programming model. Branch-and-bound algorithm verifies that no better integer feasible solution is possible. This algorithm could potentially search all 2^n binary integer vectors, where n is the number of variables. As a complete search might take a very long time, we can limit the search by using maximum amount of time in the algorithm runs.

The objective function calculates the cost of execution of job j on PE k in resource i. Here we are multiplying the cost of execution by the maximum length of time of execution of all tasks. The maximum length of time is computed through dividing the length of job in a given unit of MI by the computational power of PE k on resource i whose unit is MIPS. When job j can satisfy all constraints of PE K on resource i the value of x_{ikj} is equal to 1. So the cost of job j which is executed in PE k on resource i can be computed by objective function. But if the value of x_{ikj} is equal to 0, this means that at least one constraint cannot be satisfied. So this job is not executed on PE K from resource i and the cost of job j does not need to be computed by objective function. In PE requirement of user's jobs constraint, the numbers of PEs which are allocated to tasks (the sum of x_{ikj}) are equal to the number of PEs requirement of user's job; meaning that PE in resource i assigned to job j, in accordance with the number of user's PE requirements. In resource's capacity constraints, we seek for a resource for job j whose available numbers of PEs are more or at least equal to the number of required PEs of user's job. Next

```
  % Computing objective function
--------------------------------------------------------

for j <= 1 to number of jobs
  for i <= 1 to number of resources
    for k <= 1 to number of available PE in resource i
        obj_func= (cost of $k_{th}$ PE in resource i)*
        (max((length of job j) / (computational power of $k_{th}$ PE
        in resource i)));
    endfor
  endfor
endfor
--------------------------------------------------------
  % Computing constraints
--------------------------------------------------------
    % PE requirement of user's job constraint
  for j <= 1 to number of jobs
    for i <= 1 to number of resources
      for k <= 1 to number of available PE in resource i
              $X_{ikj=1;}$
      endfor
    endfor
     sum of $X_{ikj}$=number of PE that user need for job j execution
  endfor
--------------------------------------------------------
    %  Resource's capacity constraint
  for i <= 1 to number of resources
    for j <= 1 to number of jobs
      for k <= 1 to number of available PE in resource i
             $X_{ikj=1}$
    endfor
  endfor
  sum of $X_{ikj}$<= number of available PE in resource i;
 endfor
--------------------------------------------------------
    % Budget Constraint
  for j <= 1 to number of jobs
    for i <= 1 to number of resources
      for k <= 1 to number of available PE in resource i
        budget_const()=(cost of $k_{th}$ PE in resource i)*
        ((length of job j) / (computational power of $k_{th}$ PE in
        resource i));
      endfor
    endfor
    sum of budget_const() <= budget of job j;
  endfor
--------------------------------------------------------
    % Time Constraint
 for j <= 1 to number of jobs
   for i <= 1 to number of resources
     for k <= 1 to number of available PE in resource i
          deadline_const= max((length of job j)/
          (computational power of $k_{th}$ PE in resource i)) + MTTR of
          resource i;
    endfor
  endfor
  sum of deadline_cons() <= deadline of job j;
 endfor
--------------------------------------------------------
```

Figure 2. Pseudo Code of FES-BIP Algorithm

constraint is the budget constraint which calculates the execution cost through multiplying cost of execution in a given unit of time by the length of time of execution. It is to be mentioned that the execution cost should be less than the considered budget for this job. The last constraint, time constraint, determines the maximum execution time for job and then adds this to failure time of PEs in resource i. If the calculated time is less than the deadline, it tells us that, to a great extent, we can be sure of the completion of job execution.

Complexity of our algorithm can be explained with respect to two aspects, time and memory complexity. Complexity of memory for m resources and n jobs and k PEs in each resource is $O(nmk)$. Complexity of memory is the number of linear programming variables. Time complexity is equal to the number of algorithm iterations as indicated in Figure 3.this Figure shows an experiment with the sample with a deadline of 400 seconds and different budgets. This graph shows that the number of iterations increased sharply from 120 iterations to 3353 iterations in a range of 10 to 50 resources while the iterations decreased genteelly from 3353 iteration to 1967 in a range of 50 resources to 200 resources. The rationale behind of this decrease is obvious, in this circumstance with more resources finding the optimal result of scheduling is easier as compare to a situation with fewer resources while the number of completed job is the same. For resources fewer than 50 a few jobs are completed while the completed jobs for resources more than 50 are approximately equal. So the accepted range of resources for discussing about time complexity is from 50 to 200 resources. In addition, for jobs that are more relaxed QoS had a small number of iterations because finding the result is faster and easier.

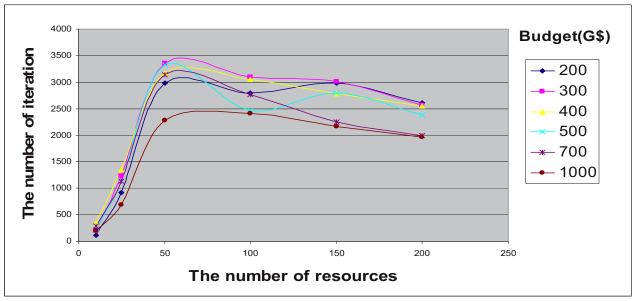

Figure 3. The Number of Iteration vs. Number of Resources for Deadline of 400 and Different Budget.

5. EXPERIMENTS AND RESULTS

We use Bintprog algorithm from MATLAB optimization toolbox to solve FES-BIP algorithm. The Bintprog algorithm is based on branch-and-bound algorithm. The parameter and configuration of grid environment in simulation of FES-BIP algorithm is presented in table 1. The number of completed jobs vs. different amounts of budget and deadline which were used in simulation are shown in Figure 4(a), Figure 4(b), Figure 4(c), Figure 4(d), Figure 4(e) and Figure 4(f).

As illustrated in Figure 4(a), Figure 4(b), Figure 4(c) and Figure 4(d), (which are concerned with resources of 200, 150, 100 and 50 respectively) for a deadline of 200 seconds the number of completed jobs regardless of the amount of budgets is around 5. Interestingly, no matter what the number of resources is, for deadlines of 400, 500

Table 1.
The Parameter and Configuration of Grid Environment

Parameter	Value of parameter
Number of job	100 jobs
Number of resources	200, 150, 100,50 and 25
Deadline	From 200 to 600 seconds in steps of 100
Budget	From 200 to 1000 G$
Cost of each PE	From 1 to 8 G$ per second by Gaussian distribution
Computational power	Between 277 and 577 MIPS.
The number of available PE in each resource	From 2 to 4
The number of tasks in each jobs	From 1 to 3
The size of each task	10000 + 10% MI by Gaussian distribution
MTTR factor	The Weibull distribution by 50 as a scale Parameter and 5 as a shape parameter [11].

and 600 seconds there is a slight rising trend of the number of completed jobs from a range 80-90 to a range of 90-100 as the amounts of budget varies from 200 to 1000 G$. However, as indicated in Figure 4(a) and Figure 4(b), for the deadlines of 300 seconds with 200 and 150 resources respectively, the number of completed jobs rises moderately from around 30 to 50 as the amounts of budget changes from 200 to 1000 G$. AS revealed in Figure 4(c) and Figure 4(d), related to resources of 100 and 50 respectively, there is a similar slight rises of the number of completed jobs from 59 to 70 as the amounts of budget changes from 200 to 1000 G$. Figure 4(e) concerns 25 resources, for deadline of 400, 500 and 600 seconds, there is moderate increase from 41 to 51 the number of completed jobs varies from 200 to 1000G$. Like the former graphs the number of completed jobs for the deadline of 200 seconds remains constant at 5 regardless of amount of budget. However, the number of completed jobs for deadline of 300 seconds rising from 35 to 45 and then leveled off as the amount of budget changes from 200 to 1000 G$. Figure 4(f) presents the number of completed jobs for 10 resources. As indicated in this figure, the number of completed jobs for the deadline of 200 is around 5 again no matter how much the amount of budget varies. For the rest of deadlines the sharp increasing trend of the number of completed jobs is similar starting from 10 to 19.

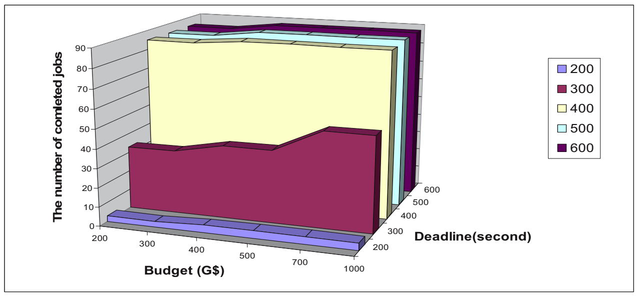

Figure 4(a). The Number of Completed Jobs for 200 Resources vs. the Amounts of Budget, Deadline and 100 Jobs.

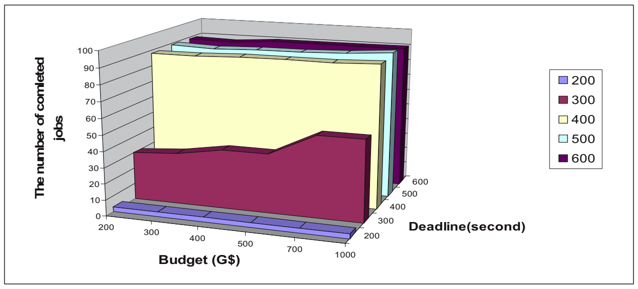

Figure 4(b). The Number of Completed Jobs for 150 Resources vs. the Amounts of Budget, Deadline and 100 Jobs.

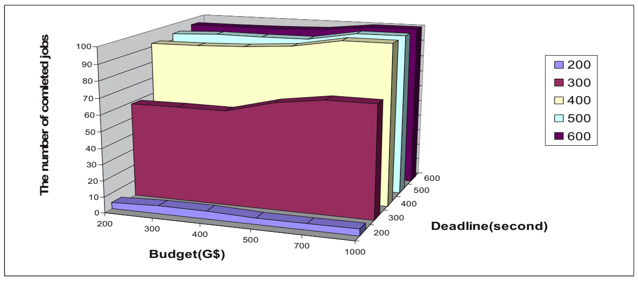

Figure 4(c).The Number of Completed Jobs for 100 Resources vs. the Amounts of Budget, Deadline and 100 Jobs.

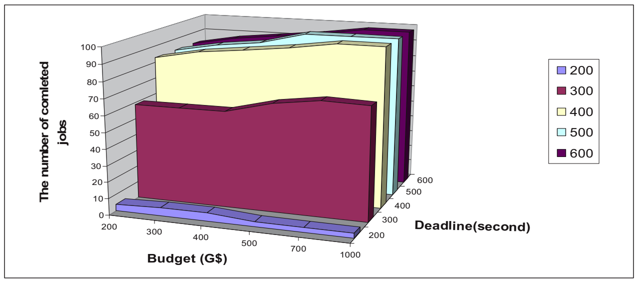

Figure 4(d) The Number of Completed Jobs for 50 Resources vs. the Amounts of Budget, Deadline and 100 Jobs.

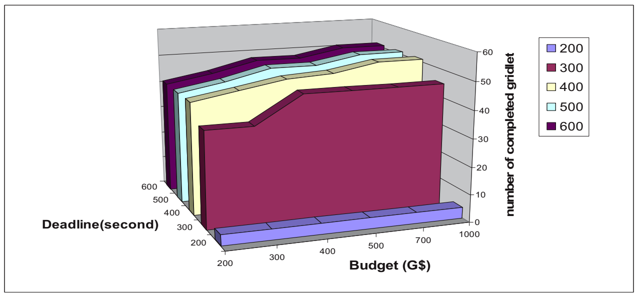

Figure 4(e). The Number of Completed Jobs for 25 Resources vs. the Amounts of Budget, Deadline and 100 Jobs.

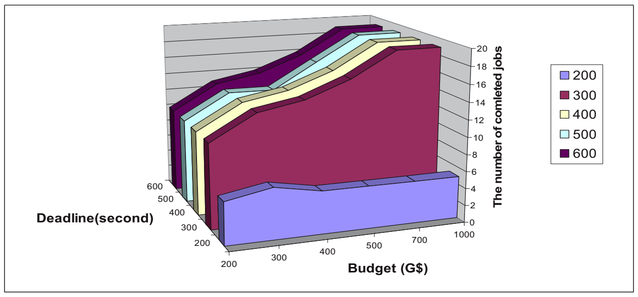

Figure 4(f). The Number of Completed Jobs for 10 Resources vs. the Amounts of Budget, Deadline and 100 Jobs.

The number of completed jobs vs. different number of resources which used in simulation is shown in Figure 5(a), Figure 5(b), Figure 5(c) and Figure 5(d). Figure 5(b), Figure 5(c), Figure 5(d) reveal the number of completed jobs of deadlines 400, 500 and 600 seconds respectively. As it can be seen there is a similar trend in all these graphs regardless of the deadlines. The number of completed jobs in all these graphs increases rapidly from a range of 10 to 19 to a range of 83 to 98 as the number of resources changes from 10-50. Then, for a range of 50 to 100 resources, the number of completed jobs remains constant. There is a slight fall of the number of completed jobs from a range of 92 to 100 jobs to a range of 85 to 88 jobs when the number of resources increases to 200. While in the real world the number of completed jobs is expected to increase or at least to stay constant as the number of resources rising. We have witness in our experiments that the number of completed jobs declines just as the number of resources exceeds 100. The reason behind this phenomenon is that as the number of variables (resources and jobs) increases the search space, consequently, enlarge, making finding a feasible and optimal answer more time consuming. It should also be noted that we assumed 60 seconds, as the maximum solving time in FES-BIP algorithm to select the resources. As a result, we can't perform as many jobs expected. To see what happens if we free the limitation of 60 seconds (the maximum solving time), we performed an experiment with no solving time limitation to solve FES-BIP algorithm. In our experiment the scheduler took 1655 seconds to solver FES-BIP algorithm to select out of 150 resources to executed 74 jobs, far beyond the plausibility of jobs' deadline. Figure 6, as an example, illustrate the result of our experiment for the deadline of 300 seconds. As it can be seen, the number of completed jobs likes that in the former Figures (5(a)-5(d)) noticeably increase from 35 to 90 for the range of 10-100 resources and from this point, as expected, the trend leveled off regardless of the rise of the number of resources.

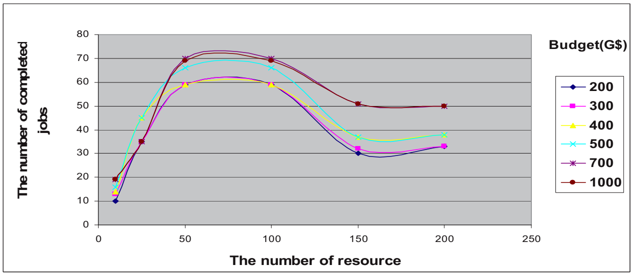

Figure 5(a). The Number of Completed Jobs by Deadline of 300 Seconds for Different Number of Resources.

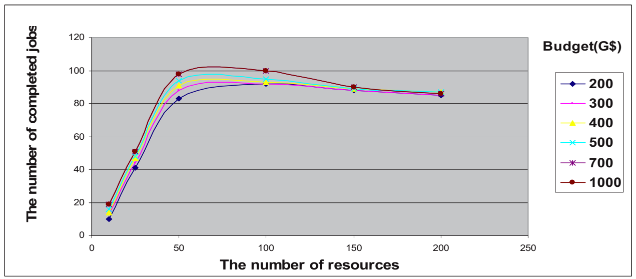

Figure 5(b). The Number of Completed Jobs by Deadline of 400 Seconds for Different Number of Resources.

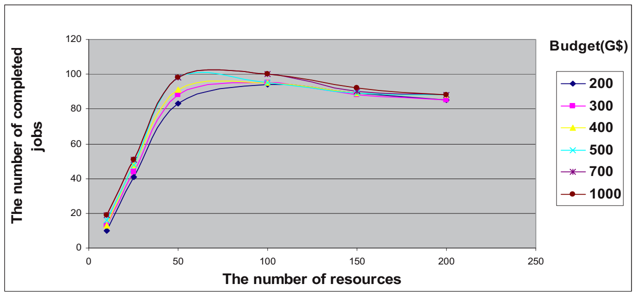

Figure 5(c). The Number of Completed Jobs by Deadline of 500 Seconds for Different Number of Resources.

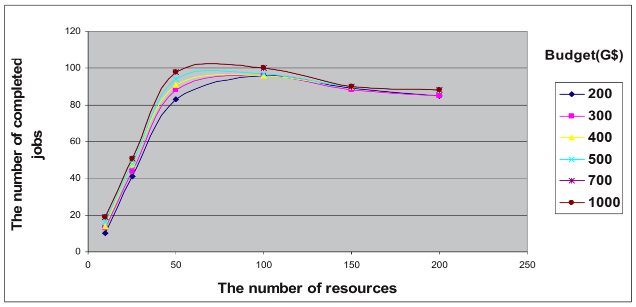

Figure 5(d). The Number of Completed Jobs by Deadline of 600 Seconds for Different Number of Resources.

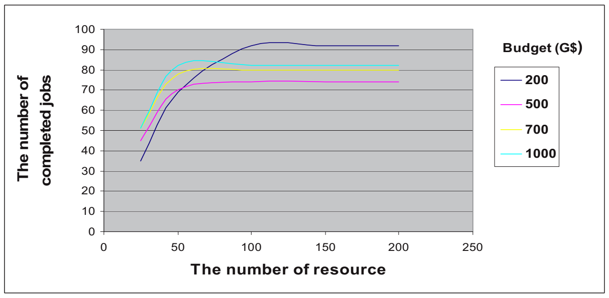

Figure 6 .The numbers of completed jobs by deadline of 300 seconds for different number of resources without any time limitation for solving FES-BIP algorithm.

To take account the total cost spent by users for jobs, we conducted some experiments which are indicated in the following Figures (7(a)-7(f)). As revealed in Figure 7(a), Figure 7(b) and Figure 7(c) concern 200,150 and 100 respectively, the total cost sharply rises from around 100 G$ to around 4500 G$ as the deadline increases to 400 seconds. From this point (4500 G$) the total cost remains constant while the deadline continues increasing this similar trend in all these graphs is independent of the number of resources. Figure 7(d), Figure 7(e) and Figure 7(f) reveal that there is a similar trend in which the total cost increases up to a certain point and from there is staying constant. Our experiment generally show that as the length of deadline and the amount of budget increase, the number of completed jobs increases accordingly. However, paying a closer attention to Figure 7(c) and Figure 7(d) shows that the total cost with 50 resources (Figure 7(d)) takes twice as much cost as with 100 resources (Figure 7(c)) although the number of completed jobs is the same. The underlying reason is that cost of resources varies when the scheduler is selecting some resources out of 100 resources it may choose those with lowest cost. These low-cost resources may not necessarily be those chosen out of 50.

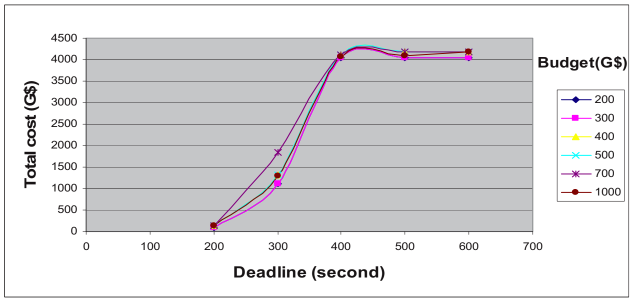

Figure 7(a). Total Spent Cost by Users for Jobs in 200 Resources and Different Budgets and Deadlines.

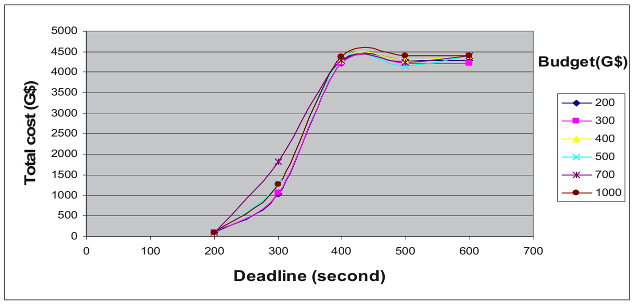

Figure 7(b). Total Spent Cost by Users for Jobs in 150 Resources and Different Budgets and Deadlines.

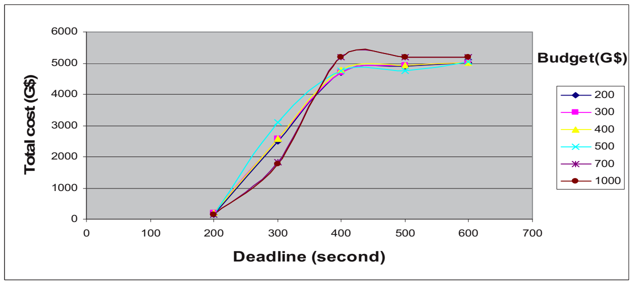

Figure 7(c). Total Spent Cost by Users for Jobs in 100 Resources and Different Budgets and Deadlines.

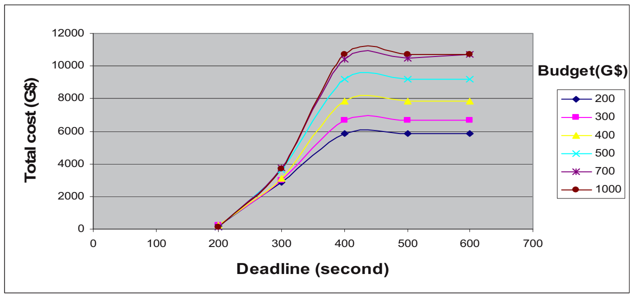

Figure 7(d). Total Spent Cost by Users for Jobs in 50 Resources and Different Budgets and Deadlines.

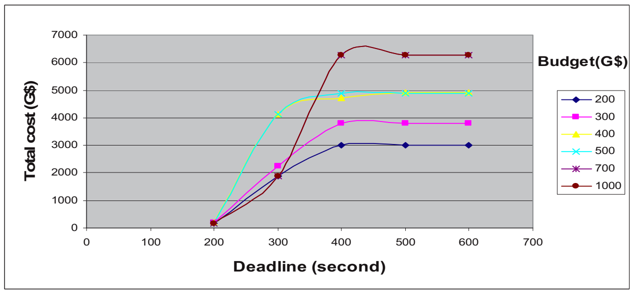

Figure 7(e). Total Spent Cost by Users for Jobs in 25 Resources and Different Budgets and Deadlines.

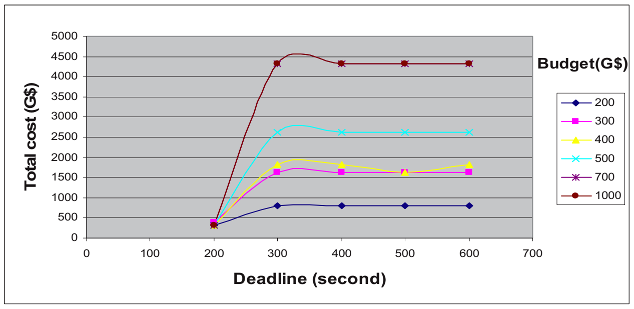

Figure 7(f). Total Spent Cost by Users for Jobs in 10 Resources and Different Budgets and Deadlines.

6. CONCLUSION AND FUTURE WORK

Our major aim in this paper was to construct and propose a model of scheduling which gets round the recurrent problems in our fault-aware economy scheduling model (FES) builds on binary integer linear programming (BIP) hence FES-BIP model.

This model can meet user's Quality of Service (QoS) requirements and can in particular find an optimal result of job scheduling in grid environment. As a follow up this study, we intend to design time and cost-time optimization algorithms. We also aim to solve the linear programming model by other optimization algorithms. As a further future aim, we will attempt to apply diverse MTTR factors for different kinds of resources using FES-BIP model.

ACKNOWLEDGEMENTS

We thank Amin Kargarian Marvasti, Faraz Faghri and Hassan Zamani for their help in solving the LP model and Sayed Hadi Hashemi and Reza Rezvani for his overall reviews.

BIOGRAPHIES

MEHDI SARIKHANI is a student of Azad University of Qazvin in Master of Computer Engineering. He received his B.A. at Sheikh Bahai University of Isfahan and he graduated in Computer Engineering in September 2007. His research interests include grid and Datagrid computing.

BAHMAN JAVADI is an INRIA post-doctoral fellow in the MESCAL team at INRIA Rhone-Alpes Grenoble. He received his PhD and Master's in Computer Engineering from Amirkabir University of Technology. His research interests include cluster and grid computing, parallel and high performance computing and networks and interconnection networks.

REFERENCES

[1] A. K. Amoura, et al., "Scheduling independent multiprocessor tasks," *Algorithmica*, vol. 32, No.2, pp.247-261, Springer New York, 2008.

[2] C. Anglano and M. Canonico, "Fault-tolerant scheduling for bag-of-tasks grid applications," LECTURE NOTES IN COMPUTER SCIENCE, vol. 3470, pp.630-639, Springer-Verlag Berlin Heidelberg ,2005.

[3] R. Buyya, et al., "Economic models for resource management and scheduling in grid computing," *Concurrency and computation: practice and experience*, vol. 14, pp. 1507-1542, John Wiley & Sons, Ltd, 6 January 2002.

[4] R. Buyya, et al., "A deadline and budget constrained cost-time optimization algorithm for scheduling task farming applications on global Grids," Monash University, Australia, Tech Rep, March 2002.

[5] R. Buyya, et al., "Scheduling parameter sweep applications on global Grids: a deadline and budget constrained cost-time optimization algorithm," *Software Practice and Experience*, vol. 35, pp. 491-512, John Wiley & Sons, Ltd, 2005.

[6] M. Canonico, "Scheduling Algorithms for Bag-of-Tasks Applications on Fault-Prone Desktop Grids," pp.55-88, Ph. D. dissertation, university of Turin, 2006

[7] G. Cheliotis, et al., "Grid Economics: 10 lessons from finance," GRIDS Lab and IBM Research Zurich and Grid Computing and Distributed Systems Laboratory and University of Melbourne, Tech. Rep, 2003.

[8] A. Dogan and F. Ozgiiner, "Scheduling independent tasks with QoS requirements in grid computing with time-varying resource prices," in Grid computing--GRID 2002: third international workshop, Baltimore, MD, USA, pp.58-69, November 18, 2002.

[9] H. Feng, et al., "A deadline and budget constrained cost-time optimization algorithm for scheduling dependent tasks in grid computing," LECTURE NOTES IN COMPUTER SCIENCE, vol. 3033, pp.113-120, Springer Berlin Heidelberg, 2004.

[10] S. Garg, et al., "A Linear Programming Driven Genetic Algorithm for Meta-Scheduling on Utility Grids," in 16th International Conference on Advanced Computing and Communications, Chennai, India, pp. 19-26, 2008.

[11] D. Kondo, et al., "The Failure Trace Archive: Enabling Comparative Analysis of Failures in Diverse Distributed Systems," to be appeared in 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2010.

[12] Z. Lan and Y. Li, "Failure-Aware Resource Selection for Grid Computing," International Conference on Dependable Systems and Networks (DSN), Philadelphia, pp.186-187, 25 June 2006.

[13] H. M. Lee, et al., "A resource manager for optimal resource selection and fault tolerance service in grids," in 10th IEEE International Symposium on Cluster Computing and the Grid, Chicago, Illinois, USA, pp.572-579, 2004.

[14] M. Maheswaran, et al., "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 59, pp. 107 - 131, 1999.

[15] M. A. Moges, et al., "Grid scheduling divisible loads from multiple sources via linear programming," 16th IASTED International Conference Parallel and distributed Computing and Systems ,Cambridge, MA, USA, pp.423-428, 9 November 2004.

[16] B. Nazir and T. Khan, "Fault Tolerant Job Scheduling in Computational Grid," 2th International Conference on Emerging Technologies, Peshawar, Pakistan, pp.708-713, 13 November 2006.

[17] P. Townend and J. Xu, "Fault tolerance within a grid environment,", In Proceedings of AHM2003, http://www.nesc.ac.uk/events/ahm2003/AHMCD/pdf/063.pdf, pp.273,2003.

[18] S. Venugopal and R. Buyya, "A deadline and budget constrained scheduling algorithm for eScience applications on data grids," LECTURE NOTES IN COMPUTER SCIENCE, vol. 3719, pp.60-72, Springer Berlin Heidelberg, 2005.

[19] S. Verboven, et al., "Dynamic grid scheduling using job runtime requirements and variable resource availability," LECTURE NOTES IN COMPUTER SCIENCE, vol. 5168, pp.223-232, Springer Spain, 2008.

[20] G. Wrzesinska, et al., "Fault-tolerant scheduling of fine-grained tasks in grid environments," *International Journal of High Performance Computing Applications*, vol. 20, , No. 1, pp.103-114, 2006.