Research Journal of Applied Sciences, Engineering and Technology 7(4): 650-655, 2014
DOI:10.19026/rjaset.7.302
ISSN: 2040-7459; e-ISSN: 2040-7467

Submitted: January 12, 2013 Accepted: March 21, 2013 Published: January 27, 2014

Research Article

Intelligent Reliable Schedule with Budget Constraints in Grid Computing

[1]Joshua Samuel Raj and [2]V. Vasudevan
[1]Department of Computer Science and Engineering, Karunya University, India
[2]Department of Information Technology, Kalasalingam Univeristy, Srivilliputhur, India

Abstract: This study proposes an intelligent reliable schedule with budget constraints in grid computing to reduce the makespan in grid environment. A good schedule improves the usability of the grid environment, but often faces the challenge of reliability and budget. In our paper we have addressed the reliability and budget constraints in grid environment by augmenting intelligence to the schedule by means of intelligent exploitation of random pushing and random job stealing. The objective of this paper is to schedule the tasks onto the reliable processor based on the user requirement and also provide the opportunity to schedule the tasks onto the reliable processor which don't have high budgets. The experimental outcomes have supported the theoretical analysis by classifying the tasks and scheduling those to the various resources which are again divided as levels of criteria based on rank values enabling us to get a good and efficient schedule.

Keywords: Budget, grid scheduling, random job stealing, random pushing, reliability

INTRODUCTION

Grid Computing a pioneer technique in harnessing the geographically dislocated computer power, has changed the perception on the utility and availability of the computer power, which has carved a new technology that openly ventures and amalgamates an infinite number of computing devices into any grid environment, augmenting to the computing capability and providing resolutions to the various tasks within the operational grid environment basically by enabling, sharing, selection and aggregation of geographically distributed autonomous resources dynamically at runtime, depending on their availability, capability, performance and cost, thereby shifting the focus to collaborative environments, federating services and exchanging transactions in a mutual manner to share resources and thereby achieve common goals to enhance productivity and speed up progress in much the same way that the Internet did in yesterdays economy, paving the way for numerous research efforts in grid scheduling mechanisms. Grid Computing is our greatest hope for delivering computing as utility to homes and offices.

Grid Computing is a type of distributed computing to solve complex applications involving huge computational requirements where the resources in the remote places are accessed by connecting all the resources together in a network. The resources of computers owned by individuals or by organizations from several countries are connected to form a single, vast super computer. A Grid gathers together resources and makes them accessible in a secure manner to users and applications (Grimshaw, 2002). Grid computing is needed when there is a necessity for huge computing of data and the data is stored in different institutions. Based on the needs, grid is classified into Private vs. Public, Regional vs. Global, All-purpose vs. Particular scientific problem. The large applications from the users are divided into a set of subtasks and sent to the several resources connected to the main server which is freely available. There are four different classes of Grid users such as end-users of applications, application developers, system administrators and managers of organizations. After the computations are over at the particular resources, the results are sent back to the main global server. As soon as all the other computations are received by the global server, the result is then provided to the user.

Grid scheduling plays the major role to schedule the tasks onto the processor efficiently. There are three phases in grid scheduling such as Resource discovery, System selection and job execution. When executing the job, the resources should be reliable so that the job can execute successfully. The reliability of the grid system is affected by several factors such as hardware failure, software failure. Reliability of computational hardware, software and data resources that comprise the grid and provide the means to execute user applications and reliability of grid networks for messaging and data transport are important and should be met (Christopher, 2009). Ignoring Grid reliability characteristics can lead

Corresponding Author: Joshua Samuel Raj, Department of Computer Science and Engineering, Karunya University, India

to reduced application performance, such as schedule length and speedup, due to wasted operations (Dogan and Ozguner, 2005). Xiao *et al*. (2000) says, reliability of the grid system depends on the failure rate of the processors and the links between them. These failure rates can be derived from Grid resource's profiling, system log and statistical prediction techniques. Srinivasan and Jha (1999) defined reliability of a system with expect to a task set as the probability that the system can run the task set without any failure.

The objective of the Tang *et al*. (2010) is to design reliability-driven scheduling architecture to measure system reliability, based on an optimal reliability communication path search algorithm to find the shortest path and then they introduce reliability priority rank (Rank) to estimate the task's priority by considering reliability overheads RASD. Two heuristic algorithms are proposed in He *et al*. (2003) such as Minimum cost Match scheduling and Progressive Reliability Maximization Schedule.

Repars and Refine algorithm are proposed in Wei *et al*. (2007) for Periodic tasks in a heterogeneous system to improve reliability while meeting the time constraints and to enhance the system reliability while being able to tolerate the failures by introducing primary backup scheme respectively. The Dynamic and Reliability-Driven Scheduling Algorithm proposed in Xiao and Hong (2005) study is to increase the reliability by minimizing the system reliability cost. The objective of RDGS algorithm proposed in Kovvur *et al*. (2011) study is to maximize the total number of tasks completing execution based on Communication to Computing Ratio (CCR) which decides the appropriate grid site for scheduling tasks.

Srikumar and Rajkumar (2005) aims to minimizes either the cost or time depending upon the choices of the user based on deadline in his study. Bag-of-Tasks applications are used for scheduling which consists of costs with requests. Stephie *et al*. (2013) considered both deadline and budget are used as a main factor to schedule the tasks on reliable processors.

Ponnambalam *et al*. (2000) aims to solve the job shop problem and to minimize the makespan using Tabu search technique.

In our study, we consider all rank level tasks. It also gives opportunity to the tasks to schedule onto the processor which has low rank level. This is accomplished by random stealing and random job pushing.

MATERIALS AND METHODS

Based on the Divide and Conquer concept, every application is divided into small tasks so that tasks can effiently utilize the processor. Finally the output of the executed tasks are combined together and send back to the user. The divided applications are send as a DAG. The computation and commuications costs of every task is calculated by the Global scheduler. If the user gives a high budget, the tasks are scheduled on to the reliable processors which have a low failure rate. The failure rate is stored in a System log by the Grid Information System (GIS).

The Reliabilty probablity of the tasks are calculated to guarentee that each task is successfully executed on the processor and the output succefully reaches its child node. Reliability of the task calculation helps to schedule the tasks on the reliable processor based on the value. The global scheduler compares the criteria given in the look up table as shown in Table 1 with the value given by the user and it will choose the algorithm. It will select the best reliable resources from list of possible reliable resource for each task, exclusive for very high level ranked tasks but never ignorant of the low level ranked tasks. Intelligence is added by means of random pushing and random job stealing.

After the execution of the algorithms, each task will get a set of reliable resource list. For example Task1 say T1, can execute on R2, R3, R6. Because after execute the algorithms the task1 may get the same reliable rank level value on the following resources:

T1 = {R2, R3, R6}

Resource manager will add additional tailer to the each task which gives the information that what are all the possible resources for a task to execute. So that the task will look like:

T1 R2, R3, R6

Initially the task is allowed to execute on the first resource from the set of reliable resource by pushing the task to the respective queue. So that RM puts the T1 to R2 queue.

Each resource queue will maintain the threshold value, after the arrival of more tasks to the same queue. For Example:

T10 = {R2, R4, R5} and T20 = {R2, R7, R8}
T10 R2, R4, R5
T20 R2, R7, R8

At that time, the tasks are pushed randomly to the any of the resource from the set of reliable resource list of each task. This concept is called Random pushing. So the T10 can push randomly to R4 or R5 instead of R2.

While one resource is idle, it will start to search for the tasks which have its resource name at its tailer and on finding such task in the resource queue it will steal the tasks from the resource queue and execute the task. This is called work stealing.

Table 1: Global scheduler lookup table

Budget	Deadline	Parameter concern	Rank level	Approach
High	Near and hard	Budget-based and deadline based	Very high 1<i<2	EEFT and reliability of the resources
Low	Near and soft	Deadline based	High 2<i<3	EEFT and reliability of the resourecs, random pushing
High	Far and hard	Budget based	Medium 3<i<4	Reliablity of the resources
Low	Far and soft	-	Low	Random job stealing, random pushing

If the resource R2 execute the tasks T1 and T20 is scheduled onto the R2, but waits for the T1 to complete. At that point of time R7 is idle, so it will grab the T20 from R2 queue.

At the start the Resource manager for its convenience will sort the resource queue from highly reliable to low reliable resources. Therefore the tasks which have low reliable rank may occupy the last few resource queues from the set of sorted resource queue and theses tasks will execute in the low reliable resource. Whenever a reliable resource is free it will check the tasks with its resource name and if unable to find such one it will steal the tasks from the lower order queues. Each resource should be aware of their order. So that low rank level tasks can also get the opportunity to execute the tasks on good reliable resource.

The Random pushing and Random stealing is defined as follows:

Random pushing: In Rob *et al*. (2001) study, random pushing is used for Load balancing. This is accomplished by pushing jobs from queue to resources, when queue length exceeds threshold value. The job allocation from a resource queue will be to random resources, thus saving processor idle time.

Random job stealing: In Robert and Charles (1994) study, random pushing is used for Load balancing. The idle processor or resources attempt to steal the tasks from other processor.

The HEFT algorithm selects the task with the highest rank value at each step and assigns the selected task to the processor, which minimizes its earliest finish time by calculating EEFT. Here the Rank is based on the deadline & budget. If the rank level is higher most and Highest then EEFT algorithm is performed first and then followed by calculating reliability of the tasks on the processor. Work done is calculated by Execution time of load of the task on the Resource as:

$$Work\ done = \frac{Weight\ of\ the\ task\ W(t)}{Capacity\ of\ the\ \text{Resource}\ C(R_i)} \quad (1)$$

Earliest Execution Finish Time (EEFT) of the task on the resource is:

$$EEFT(n,R_i) = EEST(n,R_i) + \frac{Workdone}{Rs} \quad (2)$$

where, EEST is the earliest execution start time of a task on the resources and Rs is the processor clock speed. The EEFT of the parent node should be greater than than the EEST of the child node, So that the output data can be flown from parent to child node. Assuming that the resourecs are in different locations there will be a delay in the communications among the procesors. For each task, reliabilty of the task is calculated in percentage by reliabilty of the task execution on the resources successfully without resources failure rate and delay between resources:

$$\text{Re}\,l(n) = Workdone \times (1\text{-}fR_i) \times (1\text{-}delay) \times 100 \quad (3)$$

where, fR_i denotes failure rate of the i^{th} resource R. Both failure rate of the resources and delay among the virtually connected resources is tracked from system log.

Grid information system:

```
for each fixed clock rate
do
    if GIS receive signal
    Updates GIS with resource information
    {R1, R2…RN} details
    Else
    Update Delay rate or failure rate fpi
    /disconnected
    end if
end
```

Grid scheduler:

```
for each fixed clock rate
do
  Updates RM with freely available resource
  {Rf1, Rf2…RfN} details
end
```

- Divide applications into subset of tasks:

  DAG G = <V, E>

- Compare the user B-value with the available resource B-value
- Sort the deadline
- Rank the tasks and choose the algorithm
- based on the criteria in the Lookup table:

  if Rank Level i<2 where i = 1

```
        Get the sorted Deadline array D [i]
        Initialize Rel[] = 0;
        for each task in set do
            for each available resource in set do
                    Calculate workdone using Eq 1
            Calculate EEFT (n, Ri) using the Eq 2
            Compute Rel[] = Rel (n) using Eq 3
            End
        end
      end if
if Rank level i<3 where i = 2
            for each task in the set do
               quelen = call (Ts (Rel[]), B, V)
               if quelen>limit
                exec randompushing ()
            end
end if
 if Rank Level i<4 where i = 3
            for each available Resource R
             do
            Compute Rel[] = Rel (n) using Eq 3
            end
end if
    Assign the Label to the tasks
    Select the suitable resource and put the tasks in
    Corresponding Resource Queue {Rq1, Rq2…RqN}
    else
          Assign the tasks on the any resource available
end if
     Assign the Label to the tasks
     Select the suitable resource and put the tasks in
Corresponding Resource Queue {Rq1, Rq2…RqN}
     else
Assign the tasks on the any resource available
  end if
```

Dispatcher: Dispatch the task from the Resource Queue:

$\{Rq_1, Rq_2 \ldots Rq_N\}$ to the respective resources

Algorithm 1: Intelligence augmented Reliable scheduling algorithm.

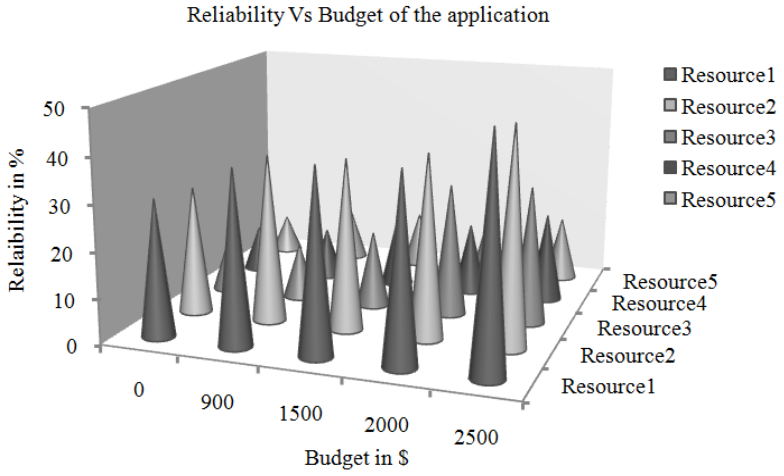

Fig. 1: The result of varying reliability based on budget

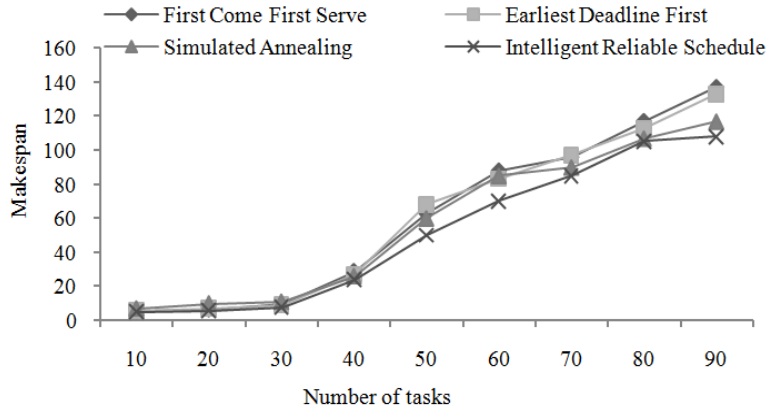

Fig. 2: Average makespan

RESULTS AND DISCUSSION

In Grid simulator Jobs are represented as Gridlets. Each Gridlet consists of job length, the size of input and output files and job owner id, along with that user have to give deadline (D-factor) and budget (B-factor). If the deadline is relaxed then the priority of the task is low, the gridlets are placed in the final positions in the resource queue. If the deadline is tight the priority is higher and executed earlier so that the task doesnt miss the deadline.

The model considered for grid environemnt in our work is a simple model which ignores the possibility of dependency between tasks, price dynamicity and communication overheads. In our experiment the resources are 2000, 2500, 6000, 9000 and 12000 GB with the assumption that the storage resource and computing resource is same. As the capacity of the resources increases, the speed and budget of the processor will be low. If the resource R1 has low capacity it should have high speed, so it is capable to execute faster and finish the execution before the deadline. The Execution start time of the task depends on the execution finish time of the previous task executed on the resources scheduled by the Grid scheduler. In dependency tasks, the execution start time of the child tasks should be after the execution finish time of the parent tasks. The execution start time of the tasks is assumed based on these criteria.

After the scheduler executes the EEFT algorithm, it will execute the Reliability of the task on the resources. If the user gives high budget value, it is possible to select the R1 based on the execution time, but the scheduler also check the Reliability of the resources and schedule the tasks to the freely available processors. In a scenario the budgets are given to the applications by the users as 0, 900, 1400, 1500 and 2500 in dollars as shown in the Fig. 1. If the user wants to execute the application freely, the scheduler will execute the applications on the resources which have low budget value. The reliability of the tasks is calculated in percentage.

The Reliability Vs Budget of the application is illustrated in the Fig. 1. The improvised makespan with the help of reliability parameter and budget parameter is compared with FCFS, EDF and SA and show in Fig. 2. From the comparative graph we infer that the Intelligent Reliable scheduling algorithm outperforms the other algorithms because of reduced failure rate.

CONCLUSION

In this study, both deadline and budget are used as main factors to schedule the tasks on reliable processors. Based on the user requirement, the algorithm is executed to achieve the reliability with a quench to satisfy the user requirement. The concepts of random pushing and job stealing are augmented as sources of augmented intelligence to get a reliable schedule within the constraints of budgets. This approach saves the running time of the algorithm. In the future, further user requirement is planned to be taken into account for reliability and scalability improvisation. Smart bacterial foraging optimization is to be explored in future to exploit the depth in research for more complicated experiments with additional objectives. Thus the reliability based scheduling is done using augmented intelligence.

REFERENCES

Christopher, D., 2009. Reliability in grid computing system. Concurr. Comp-Pract. E., 21(8): 927-959.

Dogan, A. and F. Ozguner, 2005. Biobjective scheduling algorithms for execution time reliability trade-off in heterogeneous computing systems. Comput. J., 48(3): 300-314.

He, Y., Z. Shao, B. Xiao, Q. Zhuge and S. Edwin, 2003. Reliability driven task scheduling for heterogeneous systems. Proceeding of the International Conference on Parallel and Distributed Computing and Systems, pp: 465-470.

Kovvur, R.M.R., S. Ramachandram, K.K. Vijaya and A. Govardhana, 2011. A reliable distributed grid scheduler for independent tasks. Int. J. Comput. Sci. Issues, 8(2).

Ponnambalam, S.G., P. Aravindan and S.V. Rajesh, 2000. A Tabu search algorithm for job shop scheduling. Int. J. Adv. Manuf. Technol., 16: 765-771.

Rob, V.N., K. Thilo and E.B. Henri, 2001. Efficient Load Balancing for Wide-Area Divide-and-Conquer Applications. Proceeding of the 8th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP, 01), pp: 34-43

Robert, D.B. and E.L. Charles, 1994. Scheduling Multithreaded Computations by work stealing. Proceeding of the 35th Annual Symposium on Foundations of Computer Science (FOCS). Santa Fe, New Mexico, pp: 356-368.

Srikumar, V. and B. Rajkumar, 2005. A Deadline and Budget Constrained Scheduling Algorithm for Science Applications on Data Grids. Springer-Verlag, Berlin Heidelberg, pp: 60-72.

Srinivasan, S. and N.K. Jha, 1999. Safety and reliability driven tasks allocation in distributed systems. IEEE T. Parall. Distr., 10(3): 238-251.

Stephie, I.R., S.R. Joshua and V. Vasudevan, 2013. A reliable schedule with budget constraints in grid computing. Int. J. Comp. sci., 64(3).

Tang, X., K. Li, R. Li and B. Veeravalli, 2010. Reliability-aware scheduling strategy for heterogeneous distributed computing systems. J. Parallel Distr. Com., 70(9): 941-952.

Wei, L., Q. Xiao and B. Kiranmai, 2007. Reliability-driven scheduling of periodic tasks in heterogeneous real-time systems. Proceeding of the 21st International Conference on Advanced Information Networking and Applications Workshops, 1: 778-783.

Xiao, Q. and J. Hong, 2005. A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs on heterogeneous clusters. J. Parallel Distr. Com., 65(8): 885-900.

Xiao, Q., J. Hong., X. Changsheng and H. Zongfen, 2000. Reliability-driven scheduling for real-time tasks with precedence constraints in heterogeneous systems. Proceeding of the 12th International Conference Parallel and Distributed Computing and Systems.