International Journal of Computer Sciences and Engineering Systems
Vol. 12 No.1 (June, 2018)

Computational Power and Level of Parallelism Based Scheduling for Heterogeneous Grid Environments

G. Sumathi[1] & N. P. Gopalan[2]

[1]Department of Information Technology, Sri Venkateswara College of EngineeringSriperumbudur 602105
Tamil Nadu, India, *E-mail:sumathiganesan@yahoo.com*
[2]Department of Computer Applications, National Institute of TechnologyTiruchirappalli 620015
Tamil Nadu, India, *E-mail: gopalan@nitt.edu*

Manuscript received August 25, 2017, Manuscript revised November 30, 2017

Abstract: *Grids have emerged as paradigms for the next generation parallel and distributed computing. Computational Grid can be defined as large-scale high-performance distributed computing environments that provide access to high-end computational resources. Grid scheduling is the process of scheduling jobs over grid resources. Improving overall system performance with a lower turn around time is an important objective of Grid scheduling. In this paper a Computational Power and Level of Parallelism based scheduling algorithm is proposed. The algorithm classifies the jobs into high, medium and low categories based on their priority. The classification is done based on the overall objective of maximizing resource utilization and maximizing the throughput. Priority assignment is done by considering the new parameters computational power of job and level of parallelism. Computational power required by a job is computed by considering it's computational complexity. Value for level of parallelism is assigned by considering the job parallelism and resource parallelism. A job, which needs high computational power and exhibits low parallelism is given a high priority. The fastest free resource available in the grid is allocated to the job that has high priority. Prioritizing the jobs in this way can improve the performance of computational grids. The effectiveness of our algorithm is evaluated through simulation results and it's superiority over other known algorithms is demonstrated.*

Keywords: *Computational Grid, Scheduling, Computational Complexity, Job Parallelism, Resource Parallelism*

1. INTRODUCTION

Computational Grids are emerging as a new computing paradigm for solving challenging applications in science, engineering and economics [1]. Computational Grid can be defined as large-scale high-performance distributed computing environments that provide access to high-end computational resources [2]. Each of these resources could be a uni-processor machine, a symmetric multiprocessor cluster, a distributed memory multiprocessor system, or a massively parallel supercomputer. Each resource (node) consists of a number of heterogeneous resources. The resources on the grid are usually accessed via an executing "job".

Grid scheduling is the process of scheduling jobs over grid resources. A grid scheduler is different from local scheduler in that a local scheduler only manages a single site or cluster and usually owns the resource. A grid scheduler is in charge of resource discovery, grid scheduling (resource allocation and job scheduling) and job execution management over multiple administrative domains.

In heterogeneous grid en-vironment with its multitude of re-sources, a proper scheduling and efficient load balancing across the grid can lead to improved overall system per-formance and a lower turn-around time for individual jobs. First Come First Serve (FCFS) algorithm neither considers any of the job parameters nor the resource parameters. Shortest Job Fastest Resource (SJFR) and Longest Job Fastest Resource (LJFR) algorithms consider computational complexity of jobs for scheduling and ignore the priority of a job.

A scheduling algorithm based on computational complexity and level of parallelism of the jobs is proposed and tested. Priority assignment is done by considering the new parameters computational power of job and level of parallelism. Computational power required by a job is computed by considering it's computational complexity. Computational power of job is directly proportional to the amount of time a resource needs to be reserved by a job. Value for level of parallelism is assigned by considering the job parallelism and resource parallelism. Value for level of parallelism is indirectly proportional to the amount of time a resource needs to be reserved by a job. The fastest free resource available in the grid is allocated to the job that has high priority. Prioritizing the jobs based on their nature can improve the real time performance of computational grids.

The rest of the paper is organized as follows: The grid framework is presented in Section 2. The proposed scheduling algorithm is discussed in Section 3. The performance study is carried out and results are discussed in Section 4. Finally, some concluding remarks are made in Section 5.

2. GRID FRAMEWORK

Fig. 1 shows the framework of the grid. Global and Local Grid Resource Brokers (**GGRB** & **LGRB)** and Grid Information Server (**GIS)** are the three main components of the grid. Each of these components has it's own independent functionalities that help in grid management and job scheduling and thus serve the purpose of a grid.

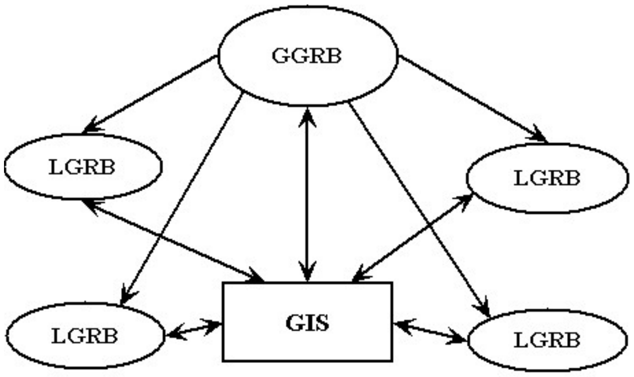

Figure 1: Framework of a Grid

2.1 Local Grid Resource Broker (LGRB)

Local grid resource broker is a synonym for a grid resource. Each grid resource has been categorized based on it's processing speed as follows:

(a) Type 1 – TFLOPS machines
(b) Type 2 – GFLOPS machines
(c) Type 3 – MFLOPS machines

This categorization adds to the heterogeneous nature of a grid. Each LGRB in the grid can be any one of the above three resources. There can be many LGRBs possible in the grid. With the addition of every LGRB, the number of resources and consequently the number of processing elements (**PE**s) are increased. A job submitted to the grid may be migrated to any of the LGRBs in the grid for execution. Once a job has been migrated to a particular LGRB, the LGRB ensures execution of the job on specified number of processors. Since computational grids have been taken into consideration, the number of processing elements in an LGRB is the actual resource of the grid.

2.2 Global Grid Resource Broker (GGRB)

Incoming jobs are submitted to the GGRB where in the jobs are scheduled. Basically, a grid scheduler (GGRB) receives applications from users, selects feasible resources for these applications according to acquired information from grid information server, and finally generates application-to-resource mappings, based on certain objective functions and predicted resource performance. The role of a scheduler is to make intelligent decisions about when and where to run jobs in order to maximize job throughput and the utilization of resources.

A single GGRB takes care of scheduling jobs in the grid based on the resources available as per the scheduling algorithm. Once a task has been scheduled to a particular LGRB, GGRB migrates the job to that LGRB for execution.

2.3 Grid Information Server (GIS)

Grid Information Server is the database bank of the grid. Information about the status of available resources is very important for the GGRB to make a proper schedule. The role of GIS is to provide such information to grid scheduler. GIS is responsible for collecting and predicting the resource state information, such as CPU capacities, memory size, etc. It can answer queries for resource information or push information to subscribers.

It keeps track of resources available in the grid. Any new LGRB should register itself with GIS. GIS provides information regarding free resources to the GGRB based on which the GGRB schedules the jobs.

Registration: Any new LGRB should register itself with the GIS by sending a request. GIS responds with an acknowledgement, which means that it is ready to accept a new resource as a grid member. Now, it's the LGRBs turn to send the details regarding itself, it's type, number of processing elements, speed of each processing element, etc.

Job Scheduling: GGRB stores the incoming jobs in a queue. When scheduling is to be done GGRB requests GIS with a query for suitable resources. As soon as GIS receives a request from the GGRB it sends the IP address of suitable resource to GGRB, if available. Jobs submitted to GGRB are migrated to LGRBs based on a scheduling algorithm for execution.

3. SCHEDULING ALGORITHM

A proper scheduling algorithm can lead to an improved overall system performance and a lower turn around time. Since a Grid has heterogeneous resources it is often complex to design an efficient scheduling algorithm.

Computational Power and Level of Parallelism based Algorithm: The jobs present in the queue maintained by GGRB are sorted according to their priority. GGRB sends a request to GIS for resource requirements having priority as search parameter for each job. GIS finds a suitable resource among the available free resources and sends the IP address of that resource to GGRB. If there is no free resource available in the grid then the job will be added to the queue. We categorize the grid resources based on their execution speed. The fastest free resource available in the grid is allocated to the job which has high priority. Accordingly, resources will be allocated to the jobs that have medium and low priority as we discussed in [4].

The Computational Power and Level of Parallelism based algorithm classifies the jobs into high, medium and low categories based on their priority. The classification is done based on the overall objective of maximizing resource utilization and maximizing the throughput (in terms of number of jobs completed). Priority assignment is done by considering the new parameters computational power of job and level of parallelism. Computational power required by a job is computed by considering it's computational complexity. Computational complexity is directly related to the amount of time a resource needs to be reserved by a job. Value for level of parallelism is assigned by considering the job parallelism and resource parallelism. Value for level of parallelism is indirectly related to the amount of time a resource needs to be reserved by a job.

Computational Complexity: Task partitioning algorithm takes care of efficiently dividing an application into tasks of appropriate grain size and an abstract model of such a partitioned application is represented by a Directed Acyclic Graph (DAG). Each task of a DAG corresponds to a sequence of operations and a directed arc represents the precedence constrains between the tasks. Each task can be executed on a processor and the directed arc shows transfer of relevant data from one processor to another.

Each node in DAG represents sequence of operations. All the operations are represented in terms of additions. Node weight represents the amount of computations (in terms of additions) involved in the particular node. This graph needs to be traversed to find out the longest path. The total sum of the amount of computations involved in each node through which the traversal has been performed leads to computational complexity of the application. In categorizing the computational complexity "high" means that the job needs scarce, powerful resources and high running time or a proper combination of them.

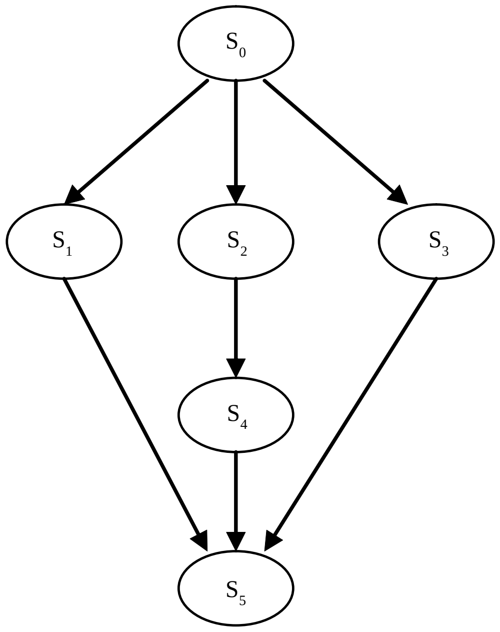

Figure 2: An Example DAG

Level of Parallelism: Amount of parallelism exhibited by a job is computed by analyzing it's layered DAG representation. Number of subtasks that can be executed in parallel is equal to width of the DAG. Fig.2 shows the DAG with subtasks S_0, S_1, S_2, S_3, S_4 and S_5. Width of the sample DAG is three. Subtasks S1, S2 and S3 can be executed in parallel. Maximum number of independent instructions getting executed in a unit time (in one clock cycle) is equal to width of the DAG that gives the amount of parallelism exhibited by the job.

Amount of parallelism exhibited by a resource is computed by considering the number of operations per cycle per processor, number of processors per node and number of nodes in a system. Amount of parallelism exhibited by each free resource available in the grid is computed. Max, min and mid ranges are fixed by considering the amount of parallelism exhibited by all available free resources in the grid. Value for level of parallelism is assigned by comparing the amount of parallelism exhibited by each job with max, min and mid ranges. Low level of parallelism means that a resource needs to be used for a long time.

Priority Assignment: A job which needs high computational power and which exhibits low parallelism is given a high priority. A job, which exhibits high parallelism and needs low computational power, is given a low priority. A job, which exhibits a medium level of parallelism and needs normal computational power, is given a medium priority. The fastest free resource available in the grid is allocated to the job that has high priority. The procedures are given below:

```
Procedure AssignLevelofParallelism(ResourceList R_List)
        While(R_List '" NULL)
        For each resource
        P1 = No. of operations per cycle per processor
        P2 = No. of processors per node
        P3 = No. of nodes in a system
/*PR_List contains the amount of parallelism exhibited by each resource*/
        PR_List[i] = P1 * P2 * P3
        End While
    Find Max, Min and Mid values in PR_List
/*PJ_List contains the amount of parallelism exhibited by each job*/
    For each job in PJ_List
    If PJ_List[i] >= Max
    LP_List[i] = High
    //LP_List contains level of parallelism value
    ElseIf PJ_List[i] >= Mid
    LP_List[i] = Medium
    Else LP_List[i] = Low
    EndIf
End AssignLevelofParallelism
Procedure AssignPriority( GridletList G_List)
  While( G_List '" NULL)
```

```
For each job
//CC_List contains the Computational Complexity of jobs
    If (CC_List[i] =High AND LP_List[i] = Low)
        Priority_List[i] = 1
    ElseIf (CC_List[i] = High AND LP_List[i] = Medium)
        Priority_List[i] = 2
    ElseIf (CC_List[i] = Medium AND LP_List[i] = Low)
        Priority_List[i] = 3
    ElseIf (CC_List[i] = High AND LP_List[i] = High)
        Priority_List[i] = 4
    ElseIf (CC_List[i] = Medium AND LP_List[i] =
        Medium)
      Priority_List[i] = 5
    ElseIf (CC_List[i] = Low AND LP_List[i] = Low)
      Priority_List[i] = 6
    ElseIf (CC_List[i] = Medium AND LP_List[i] = High)
      Priority_List[i] = 7
    ElseIf (CC_List[i] = Low AND LP_List[i] = Medium)
      Priority_List[i] = 8
    ElseIf (CC_List[i] = Low AND LP_List[i] = High)
      Priority_List[i] = 9
    EndIf
End AssignPriority
```

Let m represents number of free resources available in the grid and n represents number of jobs present in the queue. The worst case time complexity of the algorithm is O(n logn) when $m \le n$ and O(m logm) when $m > n$.

4. PERFORMANCE STUDY

We compare the performance of our algorithm with First Come First Serve, Shortest Job Fastest Resource and Longest Job Fastest Resource algorithms.

First Come First Serve (FCFS): As the name suggests this scheduling algorithm schedules the jobs on a "First come First serve" basis. As soon a job is submitted to GGRB, scheduler searches the Resource Database for an appropriate resource linearly. The job is migrated to that resource for execution. This algorithm neither considers any of the job parameters nor the resource parameters.

Shortest Job Fastest Resource (SJFR): Shortest Job Fastest Resource is a scheduling algorithm, which tries to reduce the overall turn around time of the jobs. The shortest job (based on computational complexity) is scheduled to the fastest resource (based on speed of each system available in that node). SJFR algorithm does not consider the priority of the job submitted.

Longest Job Fastest Resource(LJFR): Longest Job Fastest Resource is a scheduling algorithm, which tries to reduce the overall execution time of the jobs. The longest job (based on computational complexity) is scheduled to the fastest resource (based on speed of each system available in that node). Since the longest job is submitted to the fastest resource execution time of the longest job is drastically reduced when compared to its execution time on any other resource in the grid. LJFR does not consider the priority of a job. As far as execution time is considered it gives the best results.

Simulation Setup: We tested our algorithm by conducting many experiments. There are a total of ten free resources in the simulated grid and these are classified into Type1 (TFLOPS), Type2 (GFLOPS) and Type3 (MFLOPS) machines. Ten different jobs are considered at a time. We conducted more than 50 experiments. Different sets of jobs and resources are considered for each experiment.

Performance evaluation is done based on execution time. Execution time for each job is calculated by calculating the elapsed time between submission time and completion time of the job. The sample test case has been shown in Fig.3 to have a relative comparison of FCFS, SJFR, LJFR and Computational Power and Level of Parallelism based Scheduling algorithms. Graphs are drawn with the GridletID (JobID) on X-axis and Execution Time in seconds on Y-axis.

The simulation results show that all the three algorithms outperform FCFS. LJFR and Computational Power & Level of Parallelism based algorithms outperform SJFR. Sometimes LJFR and Computational Power & Level of Parallelism based algorithms are close and sometimes Computational Power & Level of Parallelism based algorithm outperforms LJFR. Minimal elapsed time implies high throughput and better resource utilization.

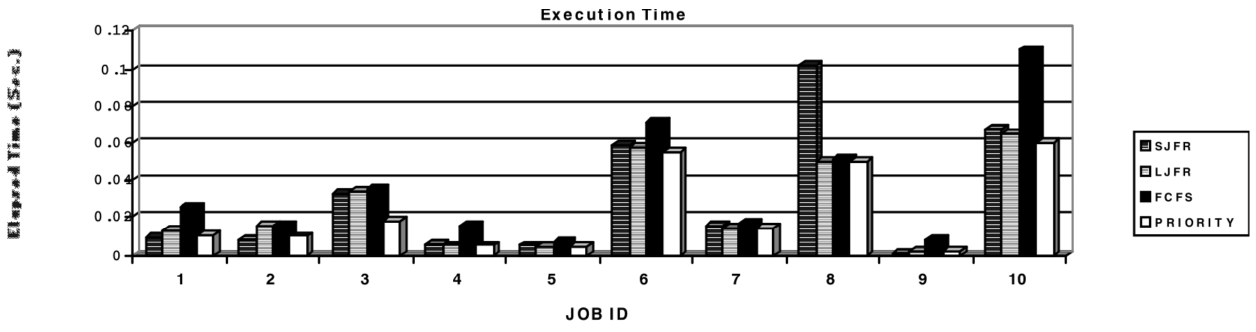

Figure 3: Sample Test Case

5. CONCLUSIONS

Design of a proper scheduling algorithm with an aim to improve the performance of a grid has indeed been a complex job with a lot of parameters to be taken into consideration. The SJFR and LJFR algorithms take computational complexity of the jobs and speed of the resources into consideration while scheduling the jobs. The Computational Power and Level of Parallelism based algorithm classifies the jobs into high, medium and low categories based on their priority. The classification is done based on the overall objective of maximizing resource utilization and maximizing the throughput in terms of number of jobs completed. Priority assignment is done by considering the new parameters computational power of job and level of parallelism. A job, which needs high computational power, exhibiting low parallelism is given a high priority. The fastest free resource available in the grid is allocated to the job that has high priority. Prioritizing the jobs in this way can improve the performance of computational grids. The effectiveness of our algorithm is evaluated through simulation results and it's superiority over other known algorithms is demonstrated.

REFERENCES

[1] Foster, I., Kesselman, C. The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1998.

[2] Foster, I., Kesselman, C.: The Globus Project: a Status Report In Proc. IPPS/SPDP'98 Workshop on Heterogeneous Computing, 1998, pp. 4-18.

[3] Abraham. A., Buyya. R., and Nath. B.: "Nature's heuristics for scheduling jobs on computational grids", Proc. 8th IEEE Int. Conf. on Advanced computing and communications, Cochin, India, 2000.

[4] Sumathi. G., Gopalan. N.P.: "Grid Scheduling Algorithms for Heterogeneous Environment", *Proc. IEEE Int. Conf. on Signal & Image Technology and Internet Based Systems* (SITIS 2005), Cameroon, West Africa.

[5] Kwok. Y. K. and Ahmad. I.: "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors, ACM Computing Surveys, 31(4), pages 406-407, 1999.

[6] Rajkumar Buyya, David Abramson and Jonathan Giddy, "Economy Driven Resource Management Architecture for Computational Power Grids", International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000), Las Vegas, USA, 2000.

[7] Francine Berman, Richard Wolski, Henri Casanova and Walfredo Cirne, "Adaptive Computing on the Grid Using AppLes", *In IEEE Transactions on Parallel and Distributed Systems*, **14**, 369-382, 2003.

[8] Tsan Sheng Hsu, Joseph C. Lee, Dian Rae Lopez and William A. Royce, "Task Allocation on a Network of Processors", *In IEEE Transactions on Computers*, **49**, 1339-1353, 2000.

[9] *www.gridbus.org/gridsim*