Scheduling Tasks on Most Suitable Fault tolerant Resource for Execution in Computational Grid

Jairam Naik K.[1], K. Vijaya Kumar[2] and N. Satyanarayana[3]

[1]*Research Scholar, Faculty of Computer Science & Engineering, JNT University, Hyderabad, India*

[2]*Professor, BMS College of Engineering, Bangalore, India*

[3]*Professor, Nagole institute of Technology & Science, Hyderabad, India*

jairam.phd643@gmail.com, kadappakumar@gmail.com, nsn1208@gmail.com

Abstract

Grid computing allows, controlled and coordinated resource sharing and problem solving in dynamic, multi- institutional virtual organizations. Grids are categorized in to two, as Computational grid and Data grid. In computing grid, allocating resources to the tasks based on its requirements and task scheduling is an important issue. Generally, the task scheduler assigns tasks to a suitable resource. The nodes with better performance would be assigned first for execution. When the load on a grid was high, and all resource with a better performance is assigned, other tasks have to be assigned to the resource with unsuitable performance. To improve the performance and throughput of the system, it is required to assign the tasks to a suitable and lower fault rate processor. Therefore, if a task is allocated to a resource without considering the performance factor, resource fault rate, the overall execution time will increases and throughput will decreases obviously. To solve this problem, a task scheduling algorithm that finds suitable and low fault rate resource for task execution is introduced in this paper. Experiments made on GridSim, shows that the proposed algorithm works well in dynamic environments.

Keywords: *Grid Computing, Task Scheduling, Processor Utilization, Fault Rate, Resources*

1. Introduction

A computational grid is a Collection of software and hardware resources that provide dependable, seamless, and pervasive access to high-end computational capabilities. It has been widely adopted in science, technology, engineering computing and business processing. The resources on the grid may be geographically dispersed in multiple administrative domains and can be installed in any operating systems, likewise Windows, Linux or UNIX. Among these applications, users may have different computing and other resource requirements in the system. Therefore, how can the resources are to be assigned to a task efficiently is an important issue in the grid environment.

The role of the grid system can be categorized into three parts. First, a computing broker is used to partition the jobs into smaller tasks. This will make jobs complete faster, by executing on multiple available resources. Second, the grid resource information service (GRIS), its main functions are, resource discovery and resource table maintenance. Third, the task scheduler is responsible for dispatching tasks to resources depending on the scheduling algorithms.

Many algorithms [3-5] have been proposed for scheduling task on grid resources. Min-min [6], max-min [6], and Fastest Processor to Largest Task First (FPLTF) [7] algorithms perform well in computing environment. The First Come First Serve (FCFS) scheduling algorithm [8] does not assure fairness and throughput among tasks, but the improved Round Robin (RR) scheduling algorithm solves this. With RR task switching is more and, if an important task requires execution immediately when the task workload is high, the requirement might not be fulfilled. The Work Queue with Replication algorithm (WQR) [9], does not need any grid information and is easy to implement. WQR provides good performance irrespective of tasks arrival, statically or dynamically, but it performs well in a homogeneous resource environment. The MFTF (Most Fit Task First) algorithm [9] assigns suitable resources for task execution, but does not consider the resource utilization problem.

The most Fitting resource for task execution algorithm [1] assigns suitable resource for task execution based on closeness factor. But the closeness factor was not considering the fault rate of the computing resource. Hence, the response time and the throughput of the grid will increases. In some cases, the response time is unpredictable, and the real-time task may miss the deadline.

Ideally, in a grid system, the resource assigned to a task should be a match to the resource requested and have to complete the execution in expected time. That is, the grid scheduling system should not assign too many or too less resources to tasks. In this paper, we proposed a scheduling mechanism that selects the most suitable, low fault rate resource for a task. The algorithm searches for the most suitable resource for a task, and balances the workload in each resource node.

The rest of this paper is organized as like this. Section 2 gives an overview of previous work on grid task scheduling in computational grid. Section 3 describes the proposed algorithm. Section 4 shows the experimental results, and Section 5 concludes the paper.

2. Related Work

In the literature, many task scheduling algorithms, such as Round Robin (RR), Fastest Processor to Largest Task First (FPLTF), Work Queue with Replication (WQR) and Most Fit Task First (MFTF) have been proposed. The common features among these algorithms are the following.

(1) All tasks are to be queued before the processors are allocated.

(2) When the required resources are available, tasks are broken down and processed according to the algorithms specifications.

(3) These steps are repeated until all tasks are completed their execution.

The only difference among them is the priority setting up mechanism for task processing. In the following section, we will introduce some scheduling algorithms performed in to groups separately by whether it is a static and a dynamic scheduling algorithm.

2.1. Static Scheduling Algorithm

The Static scheduling algorithms make all decisions before Scheduling begins. The simplest is the Round Robin (RR) algorithm. The ready queue in the RR algorithm is a logical ring, and each task is assigned for execution in turn. If a task cannot be completed in its time quantum, it will return to stay in the queue and wait for the next turn. The major advantage of the RR algorithm is that tasks are executed in turn so that they do not need to wait for the previous largest one to complete. However, it has the problem that if the queue is fully loaded and workload is heavy, tasks might have to wait in the queue for a very long time. In the extreme case, starvation might occur, and more context switching, hence consumes more processing node time.

The Fastest Processor to Largest Task First (FPLTF) [7] algorithm schedules tasks according to the workload in the grid system. The algorithm needs the information of CPU speed and task workload. FPLTF works in two steps: first, the task scheduler sorts tasks and resources to reduce task and CPU searching time. Second, the scheduler allocates the largest task in the queue to the fastest available resource in the grid environment. Dynamic Fastest Processor to Largest Task First (DFPLTF) [7, 10] is an extension of FPLTF. Work Queue with Replication (WQR) [9] is an extension of the Work Queue (WQ) algorithm [11]. It is a knowledge free scheduling algorithm. It does not need any information for scheduling. WQR has the characteristic that a faster processor will be assigned more tasks than a slower processor. Moreover, it does not use any prediction algorithm for CPU speeds and task lengths prediction. WQR uses FCFS and random transfer task to allocate resource node.

In the min-min algorithm [6], the minimum completion time for each task is computed with respect to all the resource node in a grid. The task with overall minimum completion time is selected first and assigned to the corresponding resource node in a grid. The newly mapped task is removed, and the process repeats until all tasks are mapped. The max-min algorithm [6] is very similar to the min-min algorithm. The collection of minimum completion time is calculated for every task. The task with overall maximum completion time from the group is selected and assigned to the corresponding machine. Once the machine that provides the earliest completion time is found for every task, the task that has minimum completion time is determined and then assigned to the corresponding machine. The max-min algorithm has the same complexity as the min-min algorithm.

2.2. Dynamic Scheduling Algorithm

The dynamic scheduling algorithms, such as DFPLTF [10] and Suffrage-C [7], make some or all decisions during scheduling a task on grid. In a grid environment, the load of each processor varies quickly, and dynamic scheduling algorithms are more appropriate than static ones. The DFPLTF algorithm is based on FPLTF with a good ability to adapt to the grid environment. DFPLTF needs three types of information to schedule: Task_size, Recourse_speed and Resourse_load. Recourse_speed is the speed of the Recourse and the value of it is relative. Resourse_load is the amount of the resource time which is unavailable to the application. Task_size is the time required for the resource to complete the task execution. Initially, the Resource Time to Become Available (RTBA) of each Recourse is initialized to 0. The scheduler will assign tasks to resources according to the best Task completion time (TCT).

$$TCT = RTBA + \{Task\_size / Resource\_speed * (1- Resource\_load)\}$$

The DFPLTF algorithm may have problems because it needs a lot of information (RTBA, Task_size, Resource_load and Resource _speed). This kind of information is difficult to obtain, and therefore the prediction of TCT may not be accurate. The MFTF (Most Fit Task First) algorithm [9] mainly aims at discovering fitness between tasks and resources for the user. It is an algorithm seeing the grid environment from the user's viewpoint. However, the fitness definition seems arbitrary.

The QoS guided min-min algorithm [12] is based on the min-min principle. At the beginning, the QoS guided min-min computes the task completion time of all the tasks on all the resources. Then, tasks with high QoS request are assigned first. Second, the algorithm finds the task with the minimum earliest task completion time and assigns the task to the host that achieves this completion time. The segmented min-min algorithm [13] sorts the tasks according to the average ETC (Expected Time to Compute). The task list then is divided into segments with equal size. The segment of larger tasks is scheduled first. For each segment, min-min is applied for assigning tasks to resource node. The selecting the most fitting resource for task execution algorithm [1] selects a best resource for task execution, but not considering the fault rate (node failure history) of the processing node. Hence in some cases, the important task which is to be completed in predicted time may be allocated to the resource node with more fault rate. This can cause missing dead line and reduced throughput.

3. Selecting the Most suitable Resource for a Task

Different kinds of tasks are submitted by the user to resource nodes for execution in a grid environment. In the traditional scheduling algorithm, such as FCFS, the task is assigned to a resource that can handle it, in the order in which they are arrived, without considering its appropriateness. For a larger and more difficult task that appears later, the system would be running out of capable resources to execute it. So the task has to wait and the system performance is degrades. Therefore, when allocating resources to tasks, it required to make good judgments by selecting the most suitable resources.

For a resource type, according to the capability (or cost), group the resource into L discrete levels, denoted by $r_1, r_2, \ldots r_L$ in non-monotonic increasing order. Let $|r_i|$ denote the maximum possible value of resource group r_i . The task which requires this resource has a minimum requirement that belongs to level r_I , $1 \leq I \leq L$. Although all nodes that have this resource can execute this task, the performance will suffer, if the node is from a level r_x , where $x < I$. For example, assume that the resource is the CPU speed of a node. Let each level consist of a range of 500 MHz. Then r_1 is from 0 to 500 MHz, r_2 is from 501 MHz to 1 GHz, and so on. If a task requests a node with a CPU speed of 1.1 GHz, it should be assigned to nodes in r_3 .

In the above example, if there is only one node in level r_3 and the next task requests a CPU speed of 1.5 GHz, then it has to wait. If we can predict the future and allocate the first task to a node in level r2 , then the new task can be assigned without delay. Taking this scenario into consideration, define a closeness factor C where $0 < C < 1$. A resource requirement x is close to level r_i, if $r_i - 1 < x \leq (1 + C)r_i$ .Based on this definition, we now state our resource scheduling algorithm.

Algorithm: The Most Suitable Fault tolerant Resource Scheduling Algorithm

Input: A number of ready tasks T_1 , T_2 , . . . , T_n , with a set of requested resource nodes
Output: For each task, a node with the resource to execute it
Variables: Each task T_i has a resource requirement R_i, level L_j and a closeness factor C

Step 1: Categorize the resource into L discrete levels, denoted by r_1 , r_2 , . . . , r_L from the smallest to the largest

```
Step 2:  FOR i = 1 TO n DO
           {
             FOR j = 1 to L DO
               {
                 IF $R_i$ is close to level $r_j$ ,
                   {
                     IF a node is available in level $r_j$
                         THEN assign Ti to a node in it
                     ELSE search for a free node upward from level $r_{j+1}$ , $r_{j+2}$ , to $r_L$ and assign $T_i$ to the first free node found
                   }
                 ELSE search downward from $r_{j-1}$ , $r_{j-2}$ , to $r_1$ and assign $T_i$ to the first free node found;
               }
             IF Ti is still not assigned, THEN Ti has to wait in the queue.
           }
```

Our algorithm tries to find a node with the most suitable resource for a task. If there is no free node available, we try to use a more powerful node first. As the last resort, the less powerful node is also used. Furthermore, when calculating the closeness value, sometimes the requirement given by the user is not directly comparable to the resource or the environment where the resource is used may change. We have to perform a suitable mapping before deciding which resource is the most suitable for a given task.

In the following, we demonstrate the idea of this algorithm with an example. Assume the resource is the processor. We categorize the processors speed into levels where 500 MHz is a level. Before submitting a task, the user sets a processor speed requirement. Table 1 shows the resource nodes available in a grid environment. The processor utilization means the percentage of time that the CPU is busy. The total number of resource nodes is 15 and they are divided into seven levels according to the CPU speed with 500 MHz each. The number of resource nodes in each level is shown in Figure 1.

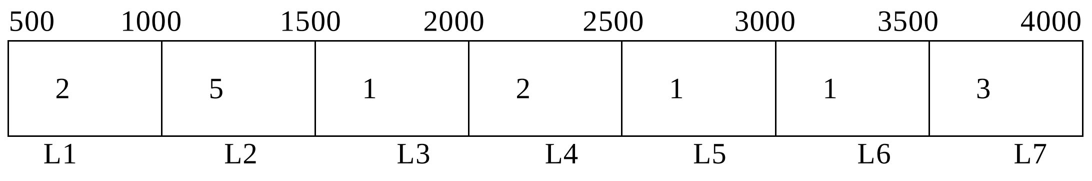


Figure 1. Seven Levels Divided Based on Processor Speed

Table 1. The Resource Nodes in a Grid Environment

Processor Speed	Processor Utilization
550 MHz	0.3
1 GHz	0.1
1.1 GHz	0.1
1.2 GHz	0.2
1.3 GHz	0.4
1.4 GHz	0.3
1.5 GHz	0.8
2 GHz	0.8
2.2 GHz	0.5
2.4 GHz	0.6
2.6 GHz	0.5
3.2 GHz	0.3
3.6 GHz	0.2
3.8 GHz	0.8
4 GHz	0.3

Table 2. PET, Cost for Each Processor

Processor Speed	PET (i , j) (s)	Fault Rate(FR)	Processor Cost (PC)
550 MHz	386.9	0.0196	394.5
1 GHz	166.7	0.0196	170
1.1 GHz	151.5	0.0392	157.4
1.2 GHz	156.3	0.0294	160.9
1.3 GHz	192.3	0	192.3
1.4 GHz	153.1	0.0686	163.6
1.5 GHz	500	0.0980	549
2 GHz	375	0.0294	386
2.2 GHz	136.4	0.0196	139
2.4 GHz	156.3	0.0686	167
2.6 GHz	115.4	0.0098	116.5
3.2 GHz	67.0	0.0294	73.1
3.6 GHz	52.1	0.0784	56.1
3.8 GHz	197.4	0.0588	209
4 GHz	53.8	0.0392	55.9

Assume a task's resource requirement is specified as 2 GHz. Directly assigning this task to a node with 2 GHz Processor is not good as can be seen from the Processor utilization of that Processor. Therefore, we have to estimate the actual performance of an assignment. For a given task j, define W_j as the estimated workload, Rj as the CPU speed requirement, and U_i and Si as the Processor utilization and Processor speed of node i, respectively.

Assume W_j = 210,000 million instructions and R_j = 2 GHz. The user desired task execution time is

$T_j = W_j / R_j = 105$ s.

Define

PET $(i, j) = (W_j / S_i)/(1 - U_i)$,

Where PET stands for Predicted Execution Time. Table 2 lists the PET value for each CPU in Table 1.

To assign the task, to a most suitable resource, it is also required to consider the Fault Rate (FR) of the processors in the grid. The Fault Rate (FR) of the processor is defined as the number of time the processor was failed in completing the execution of the given task in the expected time. The reasons for not completing the task execution in time may be its low speed or any other hardware faults. If the task is assigned to the processor based on Predicted Execution Time (PET), it may not complete the priority task in time, hence missing the deadline. This can also reduce the throughput of the system and increases the response time. To solve like this problem, considering the Fault Rate (FR) of the processor is mandatory. The FR of each processor [2] is collected from its previous performance and history. Now, the Fault Rate of each processor (FR) is calculated as

Fault Rate (FR) $= N_f / (N_f + N_s)$

Where, N_f, N_s are the number of time the processor is failed; number of time the processor is succeeded in completing the task execution in time respectively. Now we can use the PET, FR to determine the Processor Cost (PC). PC is the value of the processor, which will gives the value or rank of the processor.

Processor Cost $(PC_i) = PET_i * (1 + FR_i)$

Table 3. Leveling According to Processor Cost Values

Level	Processor Cost(PC) Range	Processor
1	600–550	None
2	550–500	1.5 GHz
3	500–450	None
4	450–400	None
5	400–350	550MHz, 2GHz
6	350–300	None
7	300–250	None
8	250–200	3.8GHz
9	200–150	1GHz, 1.1 GHz 1.2 GHz,1.3 GHz 1.4 GHz,2.4 GHz
10	150–100	2.2 GHz,2.6 GHz
11	100–50	3.2 GHz,3.6 GHz 4 GHz
12	50–0	None

Now we can use the PC to re-categorize the node. Assume the gap is 50 s per level. The first level is from 600 to 550 s, the second is from 550 to 500 s, and so on. The categorization is shown in Table 3.

Assume C = 0.02. Since the resource requirement (the desired task execution time) is 102 s and 102> (1 + 0.02) × 100, we will select a node from level 11.

4. Experimental Results

We have implemented our experiments on a java-based grid simulation tool, called GridSim [14]. By the simulator, we did two experiments. The first experiment analyzes closeness value, and the second one compares the proposed method with Most Fitting Resource (MFR). For simulation, we considered 1000 tasks and 200 processor resources. The resource requirements for tasks are between 1000 and 4000 MIPS (Million Instructions per Second). That is, a task requires to be executed by a processor resource that is at least as powerful as the resource requirement. The workloads for each task are randomly generated between 300,000 and 500,000 million instructions. The speed of processor resource is randomly generated between 1000 and 4000 MIPS. There are six levels for resource nodes with every 500 MIPS.

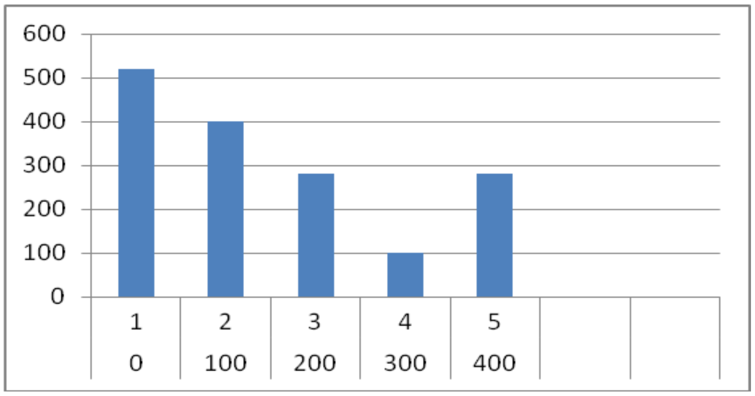
The users can submit 100 tasks in every 100 s. The detailed simulation parameters are shown in Table 4. 300,000–500,000 million instructions correspond to the operations needed for two N × N matrices, where N is between 1000 and 2000, and the current common clock speed of CPU is between 1 and 2 GHz. Allowing two instructions in one cycle, the assumption of 1000–4000 MIPS for resource speed is reasonable.

Table 4. Simulation Parameters

Parameters	Value
Number of tasks	1000
Number of processor resources	200
Resource requirement (MIPS)	1000–4000
Task Workload (MI)	300,000–500,000
Processor Resource Speed (MIPS)	1000–4000

Deviation time is defined as the difference between the fastest completed task execution time and the slowest completed task execution time. We first test how the different closeness factors affect the deviation time. Figure 2 shows the simulation results. When closeness factor is 50%, the deviation time is large. It means the difference between the fastest completed task execution time and the slowest completed task execution time is very large.

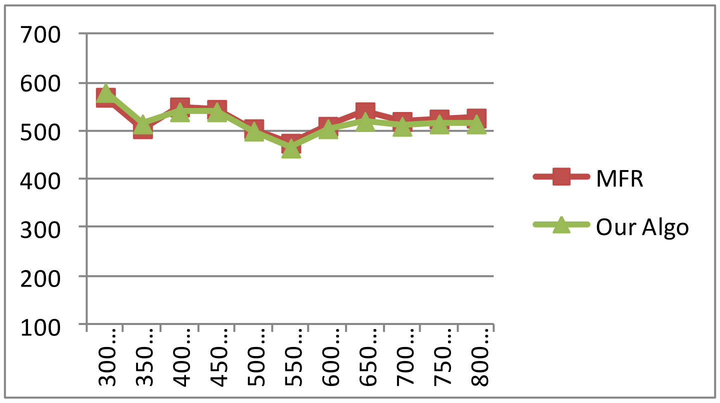
Thus, the task execution time is not very stable. It is because the large closeness factor diminishes the benefits of appropriateness. The smallest deviation time appears in a closeness factor of20%. It means that all tasks are properly and fairly treated. But when we further reduce the closeness factor to 10%, the deviation time increases again. Because the closeness factor is too small it restricts the choice of resources. Some tasks are allocated to a higher level with faster CPU speed. It results in inefficient resource allocation with some waste. Therefore, we use a closeness factor of 20% in the next experiment.



X - Closeness F
Y - Deviation Time (Sec)

Figure 2. The Closeness Factor Compared with Deviation Time

In the second experiment, we use a dynamic environment. Assume there are 2000 tasks and 400 processor resources. The tasks will arrive randomly. They are assigned to resource nodes in every 100 s. The resource requirement is between 1000 and 4000 MIPS. The workload of a task is generated randomly between 2000,000 and 3000,000 million instructions. The speed of a resource node is generated randomly between 1000 and 4000 MIPS. Between 100 and 300 tasks are generated and submitted randomly in every 100 s. We compare the proposed algorithm with MFR. The simulation results are illustrated in Figure 3.



X - Average execution time (Sec)
Y - Elapsed Time (Sec)

Figure 3. Average Execution Time for Different Scheduling Algorithms

In Figure 3, the MFR algorithm will takes a larger average execution time, because it was not considering the FR. Among the simulations, the proposed algorithm has stable average execution time. The waiting time for each task is smaller, and the average execution time is faster, throughput was more, because lower fault rate processor is assigned to the task.

One trick in the simulation execution is the calculation of PET, PC. To avoid recalculating it for each task, we can separate the tasks into several discrete categories (such as that used for resource speed) according to their workload requirements. Then in each category, we have a separate PET table, which can then be calculated beforehand. The PET, PC tables need to be updated only for those resources whose load has been changed. Then in real time task assignment, PET can be obtained by a simple table look-up instead of recalculating regarding all resources.

Finally, we must note that before the elapsed time 600, the MFR have shorter average execution time. That is, if the system load is light, it is possible that our method may let the most powerful resource lay idle. Therefore, a hybrid two-stage scheduling strategy is recommended. When in low Fault rate (FR), use MFR. When the system FR grows beyond a threshold, switch to our method.

5. Conclusion

Few task scheduling algorithms are efficient in static environments such as FPLTF and WQR. In dynamic environments, the tasks are created dynamically and arrive randomly, in different quantities. The static task scheduling algorithms are not suitable for dynamic environments. The MFR is suitable under low FR of resource node. In this paper, we propose an algorithm to assign resources to the tasks and scheduling in dynamic environments. The idea is to assign prioritized task to lower fault rate resource that can execute it in time. We first categorized the resource into different levels. Then a resource node is selected from the most suitable level. This approach can improves the throughput, by allocating the task to a most suitable lower fault rate resource node.

References

[1] R. -S. Chang, C. -F. Lin and J. -J. Chen, "Selecting the most fitting resource for task execution", Future Generation Computer Systems, vol. 27, **(2011)**, pp. 227-231.

[2] M. Amoon, "A Fault-tolerant scheduling system for computational grids", Computers and electrical Engineering, vol. 38, **(2012)**, pp. 399-412.

[3] R. -S. Chang, J. -S. Chang and P. -S. Lin, "An ant algorithm for balanced job scheduling in grids", Future Generation Computer Systems, vol. 25, **(2009)**, pp. 20– 27.

[4] K. Leal, E. Huedo and I. M. Lorene, "A decentralized model for scheduling independent tasks in Federated Grids", Future Generation Computer Systems, vol. 25, **(2009)**, pp. 840–852.

[5] R. -S. Chang, J. -S. Chang and S. -Y. Lin, Job scheduling and data replication on data grids, Future Generation Computer Systems, vol. 23, **(2007)**, pp. 846–860.

[6] M. Maheswaran, S. Ali, H. Siegel, D. Hensgen and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems, Journal of Parallel and Distributed Computing, vol. 59, **(1999)**, pp. 107–131.

[7] D. Saha, D. Menasce and S. Porto, et. al., "Static and dynamic processor scheduling disciplines in heterogeneous Parallel architectures, Journal of Parallel and Distributed Computing, vol. 28, **(1995)**, pp. 1–18.

[8] H. Casanova, A. Legrand, D. Zagorodnov and F. Berman, "Heuristics for scheduling parameter sweep applications in grid environments", in: Heterogeneous Computing Workshop, **(2000)** May 1, pp. 349–363.

[9] S. Wang, I. Hsu and Z. Huang, "Dynamic scheduling method for computational grid environments", in: Proceedings of the International Conference on Parallel and Distributed Systems, **(2005)** July, pp. 22–28.

[10] D. Silva, W. Cirne and F. Brasileiro, "Trading cycle for information: using replication to scheduling bag of tasks application on computational grids", in: Proceeding in Ruro-Par, **(2003)** August.

[11] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network", in: Proceedings First International Conference on Peer-to-Peer Computing, **(2001)** August, pp. 99–100.

[12] M. Wu, W. Shu and H. Zhang, "Segmented min-min: a static mapping algorithm for meta-tasks on heterogeneous computing systems", in: Heterogeneous Computing Workshop Proceedings, **(2000)**, pp. 375–385.
[13] H. Xiaoshan, X. Sun and G. Laszewski, "QoS guided min-min heuristic for grid task scheduling", Journal of Computer Science and Technology, vol. 18, **(2003)**, pp. 442–451.
[14] A. Sulistio, C. Yeo and R. Buyya, "Visual modeler and simulation toolkit", in: International Conference on Computational Science, **(2003)**, pp. 1123–1132.
[15] L. O'Gorman, M. J. Sammon and M. Seul, "Practical Algorithms for Image Analysis", 2nd ed., Cambridge University Press, **(2008)**.
[16] D. Coppersmith and S. Winograd, "on the asymptotic complexity of matrix multiplication", in: Proc. 22nd Annual Symposium of Computer Science, **(1981)** October 28–30, pp. 82–90.

Authors

Jairam Naik K Received his B.Tech, M.Tech Degres of Engineering in the Specialization of Computer Science & Engineering from JNT University, Hyderabad in 2003, 2009 Respectively. Now, he is Pursuing Ph.D with Faculty of Computer Science & Engineering, JNT University, Hyderabad. His current research interests include distributed parallel computing, Grid computing, Computer Communication & Networking.

Dr. K. Vijaya Kumar Received his Ph.D. Degree from Hyderabad Central University, Andhrapradesh in 2010. Now, he is a Professor of Computer Application Dept, BMS College of Engineering, Bangalore. His current research includes Feature Partitioning Approaches to Principal Component Analysis, Pattern Recognition, Distributed and parallel computing, Grid computing, Computer Communication & Networking.

Dr. N. Satyanarayana Received his Ph.D. Degree from Acharya Nagarjuna University in 2009. Now, he is a Professor of Computer Science & Engineering Dept, Nagole Institute of Technology & Science, Hyderabad. His current research interests Data Encoding Techniques for Delay and Energy Efficiency, Grid computing, Computer Communication & Networking, Mobile computing.