SURVEY ON GRID SCHEDULING

Mrs.G.Malathi
Assistant Professor,
Department of B.E. CSE
Vellalar College of Engineering and Technology,
Erode
Email: malathi_gurunathan@rediffmail.com

Mrs. S.Sarumathi
Professor,
Department of B.Tech IT
K.S.Rangasamy College of Technology ,
Tiruchengode -637215,
Email:rishi_saru20@rediffmail.com

ABSTRACT

Grid computing has become an increasingly popular solution to support the sharing and coordination of heterogeneous and geographically distributed resources. To perform application execution in the Grid, scheduling of Grid resources is necessary. To achieve this goal, an efficient Grid scheduling system is an essential part of the Grid. Grid Resource Scheduling (brokering) is defined as the process of making scheduling decisions involving resources over various domains. Grid scheduling is more complicated than local resource scheduling because it must manipulate large-scale resources across management. Here various Grid scheduling algorithms are discussed from different points of view, such as static vs. dynamic policies, objective functions, applications models, adaptation, QoS constraints, and strategies dealing with dynamic behavior of resources, and so on. Rather than covering the whole Grid scheduling area, this survey provides a review of the subject mainly from the perspective of scheduling algorithms.

Keywords- Grid Computing, Grid Scheduling, Algorithms, Performance, Reliability.

1. INTRODUCTION

Recent research on these topics has led to the emergence of a new paradigm known as *Grid computing.* Grid Computing is intended to offer seamless access to varieties of resources. A computational grid is based on the cooperation of distributed computer systems where user jobs can be executed either on local or on remote computer systems. To achieve the promising potentials of tremendous distributed resources, effective and efficient scheduling algorithms are fundamentally important. Unfortunately, scheduling algorithms in traditional parallel and distributed systems, which usually run on homogeneous and dedicated resources, e.g. computer clusters, cannot work well in the new circumstances [1]. In the next section, the basic concepts of grid computing and grid scheduling are present. In this paper, the state of current research on scheduling algorithms for the new generation of computational environments will be surveyed. Then we present the classification of existing scheduling algorithms. On the other hand there is the expectation that a larger number of resources are available. It is expected that this will result in a reduction of the average job response time. Also the utilization of the grid computers and the job-throughput is likely to improve due to load-balancing [6] effects between the participating systems.

2. OVERVIEW OF THE GRID SCHEDULING PROBLEM

A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities [2]. Grid Computing offers a model for solving massive computational problems by making use of the unused resources (CPU cycles and/or disk storage) of large numbers of disparate, often desktop, computers treated as a virtual cluster [4, 3] embedded in a distributed telecommunications infrastructure.

From the point of view of scheduling systems, a higher level abstraction for the Grid can be applied by ignoring some infrastructure components such as authentication, authorization, and resource discovery and access control. Thus, in this paper, the following definition for the term *Grid* adopted: "*A type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous and heterogeneous resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements*" [5]. To facilitate the discussion, the following frequently used terms are defined:

• A ***task*** is an atomic unit to be scheduled by the scheduler and assigned to a resource.

• The ***properties*** of a task are parameters like CPU/memory requirement, deadline, priority, etc.

• A ***job*** (or ***metatask***, or ***application***) is a set of atomic tasks that will be carried out on a set of resources. Jobs can have a recursive structure, meaning that jobs are composed of sub-jobs and/or tasks, and sub-jobs can themselves be decomposed further into atomic tasks. In this paper, the term *job*, *application* and *metatask* are interchangeable.

• A ***resource*** is something that is required to carry out an operation, for example: a processor for data

processing, a data storage device, or a network link for data transporting.

• A ***site*** (or ***node***) is an autonomous entity composed of one or multiple resources.
• A ***task scheduling*** is the mapping of tasks to a selected group of resources which may be distributed in multiple administrative domains.

3. GRID SCHEDULING ARCHITECTURE

The phases of this scheduler are illustrated in Figure 1. Grid scheduling involves three main phases: resource discovery, which generates a list of potential resources, information gathering about those resources and selection of a good set of them, and job execution, which includes file staging and cleanup.

Resource Discovery*:* The first phase, i.e. resource discovery. This phase requires a standard way to express application requirements with respect to the resource information stored in the Grid Information System. It is thus needed an agreed-upon schema to describe the attributes of the systems, in order for different systems to understand what the values mean.

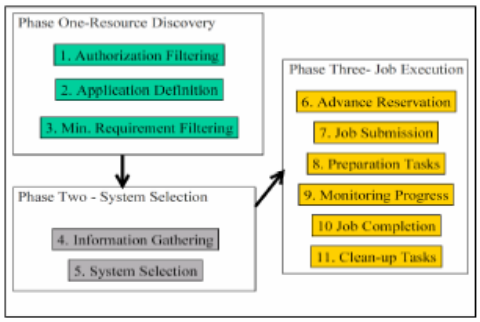

Figure 1: Architecture of a general Grid scheduler.

Resource selection*:* Resource selection usually occurs after resource discovery phase. While the first phase filters out unwanted resources, this phase should determine from this large list the best set of resource(s) chosen to map the application.

This selection requires gathering of detailed dynamic information from resources, e.g. by querying performance prediction systems such as Network Weather System (NWS). This information should be used to rank resources, and to allow the scheduler to choose the ones that should ensure high performance in the execution of the application.

While resource selection can be quite simple for sequential jobs, this selection can become particularly complex for multi-component parallel applications. This mechanism is considered inflexible and not suitable with respect to the autonomous nature of Grid resources. For example, the owner of a resource might want to allow access only to users belonging to a certain group or able to pay a fee. It was exactly to propose that resource retrieval and selection should be treated as a bilateral matching process.

Job Execution*:* The last phase of the scheduling architecture is job execution. This phase can be very complex, since the preparation of a job run can require various intermediary steps, like staging of files, advance reservation, etc.

One of the main activities shown in Figure 1 is the monitoring of the progress of application execution. A user, when a job is not making sufficient progress, may stop the job and reschedule it by returning to Step 4. Such rescheduling is significantly harder for parallel job executing on multiple sites. Dynamic scheduling is needed in that case.

4.CHALLENGES OF SCHEDULING ALGORITHMS IN GRID COMPUTING

Scheduling algorithms have been intensively studied as a basic problem in traditional parallel and distributed systems, such as symmetric multiple processor machines (SMP),massively parallel processors computers (MPP) and cluster of workstations (COW).Looking back at such efforts, we find that scheduling algorithms are evolving with the architecture of parallel and distributed systems. Table 1 captures some important features of parallel and distributed systems and typical scheduling algorithms they adopt.

Table 1: Evolution of scheduling algorithms with parallel and distributed computing systems

Typical Architecture	DSM, MPP	COW	Grid
Chronology	Late 1970s	Late 1980s	Mid 1990s
Typical System Interconnect	Bus , Switch	Commercial LAN, ATM	WAN/Internet
Cost of Interconnection	Very Low/ Negligible	Low/Usually Not Negligible	High / Not Negligible
Interconnection Heterogeneity	None	Low	High
Node Heterogeneity	None	Low	High
Single System Image	Yes	Yes	No
Resource Pool Static/Dynamicity	Predetermined and Static	Predetermined and Static	Not Predetermined and Dynamic
Resource Management Policy	Monotone	Monotone	Diverse
Typical Scheduling Algorithms	Homogeneous Scheduling Algorithms	Heterogeneous Scheduling Algorithms	Grid Scheduling Algorithms

The reason can be found by going through the assumptions underlying traditional systems [7]:

- All resources reside within a single administrative domain.
- To provide a single system image, the scheduler controls all of the resources.
- The resource pool is invariant.
- Contention caused by incoming applications can be managed by *the scheduler according to some* policies, so that its impact on the performance that the site can provide to each application can be well predicted.

- Computations and their data reside in the same site or data staging is a highly predictable process, usually from a predetermined source to a predetermined destination. Unfortunately, all these assumptions do not hold in Grid circumstances. In Grid computing, many unique characteristics make the design of scheduling algorithms more challenging [8], faced due to

    - Heterogeneity and Autonomy
    - Performance Dynamism
    - Resource Selection and Computation-Data Separation

5. GRID SCHEDULING ALGORITHMS

In this section, we provide a survey of scheduling algorithms in Grid computing,

5.1 A Taxonomy of Grid scheduling Algorithms

Since Grid is a special kind of such systems, scheduling algorithms in Grid fall into a subset of this taxonomy.

- **Local vs. Global**

At the highest level, a distinction is drawn between local and global scheduling. The local scheduling discipline determines how the processes resident on a single CPU are allocated and executed; a global scheduling policy uses information about the system to allocate processes to multiple processors to optimize a system-wide performance objective. Obviously, Grid scheduling falls into the global scheduling branch.

- **Static vs. Dynamic**

The next level in the hierarchy (under the global scheduling) is a choice between static and dynamic scheduling. This choice indicates the time at which the scheduling decisions are made. In the case of static scheduling, information regarding all resources in the Grid as well as all the tasks in an application is assumed to be available by the time the application is scheduled. By contrast, in the case of dynamic scheduling, the basic idea is to perform task allocation as the application executes. Both static and dynamic scheduling are widely adopted in Grid computing. For example, static scheduling algorithms are studied in [9] and [14] and in [12] [13] [10] and [11], dynamic scheduling algorithms are presented.

- ***Static Scheduling***

In the static mode, every task comprising the job is assigned once to a resource. Thus, the placement of an application is static, and a firm estimate of the cost of the computation can be made in advance of the actual execution. One of the major benefits of the static model is that it is easier to program from a scheduler's point of view. The assignment of tasks is fixed a priori, and estimating the cost of jobs is also simplified. The static model allows a "global view" of tasks and costs. But cost

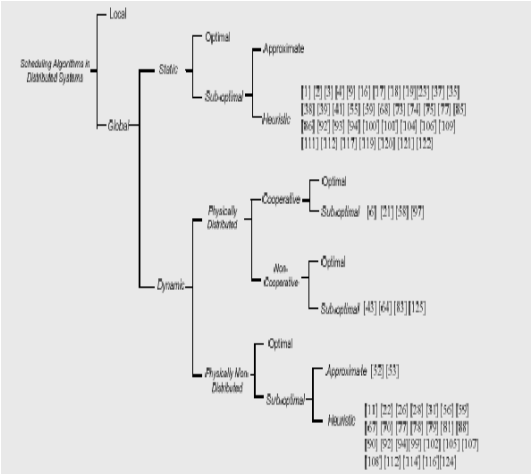

Fig. 2: A Hierarchical taxonomy for scheduling algorithms. Branches covered by Grid scheduling algorithms up to date are denoted in italics. Examples of each covered branch are shown at the leaves.

estimate based on static information is not adaptive to situations such as one of the nodes selected to perform a computation fails, becomes isolated from the system due to network failure, or is so heavily loaded with jobs that its response time becomes longer than expected. Unfortunately, these situations are quite possible and beyond the capability of a traditional scheduler running static scheduling policies. To alleviate this problem, some auxiliary mechanisms such as rescheduling mechanism [15] are introduced at the cost of overhead for task migration. .

- ***Dynamic Scheduling***

Dynamic scheduling is usually applied when it is difficult to estimate the cost of applications, or jobs are coming online dynamically (in this case, it is also called online scheduling). A good example of these scenarios is the job queue management in some meta computing systems like Condor [17] and Legion [16]. Dynamic task scheduling has two major components [18]: system state estimation (other than cost estimation in static scheduling) and decision making. According to how the dynamic load balancing is achieved, there are four basic approaches [19]:

- Unconstrained First-In-First-Out (FIFO, also known as First-Come-First- Served)
- Balance-constrained techniques
- Cost-constrained techniques
- Hybrids of static and dynamic techniques

- **Optimal vs. Suboptimal**

In the case that all information regarding the state of resources and the jobs is known, an optimal assignment could be made based on some criterion function, such as minimum make span and maximum resource utilization. But due to the NP-Complete nature of scheduling algorithms and the difficulty in Grid scenarios to make reasonable assumptions which are usually required to prove the optimality of an algorithm, current research

tries to find suboptimal solutions, which can be further divided into the following two general categories.

- **Approximate vs. Heuristic**

The approximate algorithms use formal computational models, but instead of searching the entire solution space for an optimal solution, they are satisfied when a solution that is sufficiently "good" is found. In the case where a metric is available for evaluating a solution, this technique can be used to decrease the time taken to find an acceptable schedule. The factors which determine whether this approach is worthy of pursuit include [20]: In [21] eleven heuristics studied are: OLB, MET, MCT, Min-min, Max-min, Duplex, GA, SA, GSA, Tabu, A*. Among them, the following heuristics are very interesting:

OLB: ***Opportunistic Load Balancing*** is the simplest strategy to assign each task to the next available machine without considering the expected execution time on that machine.

MET: Here ***Minimum Execution Time*** considers the expected execution time of each task on a machine and selects the machine with minimum execution time.

MCT: ***Minimum Completion Time*** assigns each task to the machine with minimum completion time (machine available time + ETC). This is the most common metric in use.

Min-min: The ***Min-min heuristic*** is a two step task scheduler. First, select a "best" (with minimum completion time) machine for each task. Second, from all tasks, send the one with minimum completion time for execution. The idea behind Min-min is to send a task to the machine which is available earliest and executes the task fastest.

Max-min: The ***Max-min heuristic*** takes the same first step as Min-min but send the task with maximum completion time for execution. This strategy is useful in a situation where completion time for tasks varies significantly. Using this heuristic, the tasks with long completion time are scheduled first on the best available machines and executed in parallel with other tasks. This leads to better load-balancing and better total execution time.

GA: A ***Genetic Algorithm*** is an evolutionary technique for large space search. The general procedure of GA search is as following: 1) ***Population generation.*** A population is a set of *chromosomes*. Each chromosome represents a possible solution, which is a mapping sequence between tasks and machines. [21] Randomly generate 200 chromosomes; 2)***Chromosome evaluation***. Each chromosome is associated with a *fitness value*, which is the total completion time of the task-machine mapping this chromosome represents. The goal of GA search is to find the chromosome with optimal fitness value; 3)***Crossover*** **and** ***mutate*** the chromosomes selected based on *selection rules*. The selection rules are analogous to evolutionary selection rules. But in many papers chromosomes are randomly selected. Crossover is the process of swapping certain subsequences in the selected chromosomes. Mutate is the process of replacing certain subsequences with some task-mapping choices new to the current population. Both crossover and mutation are done randomly [21]. After crossover and mutation, a new population is generated. Then it will be evaluated, and the process starts over until some *stopping criteria* are met. The stopping criteria can be, for example, 1) no improvement in recent evaluations; 2) all chromosomes converge to the same mapping; 3) cost bound is met.

A typical GA structure

```
Genetic Algorithm ()
[
Initialize population;/*an initial population of strings randomly generated*/
Evaluate population;/*check fitness of string against fitness function*/
While termination criterion not reached
        [       /*genetic operators*/
                Select solution for next population;
                Perform crossover and mutation;
                Evaluate population;
        ]
]
```

SA: ***Simulated Annealing*** is a search technique based on physical process of annealing, which is the thermal process of obtaining low-energy crystalline states of a solid. The *temperature* is increased to melt solid. If the temperature is slowly decreased, particles of the melted solid arrange themselves locally, in a stable "ground" state of a solid. SA theory states that if temperature is slowed sufficiently slowly, the solid will reach thermal equilibrium, which is an optimal state. By analog, the thermal equilibrium is an optimal task-machine mapping (optimization goal), the temperature is the total completion time of a mapping (cost function), and the change of temperature is the process of mapping change. If the next temperature is higher, which means a worse mapping, the next state is accepted with certain exponential probability. The acceptance of "worse" state provides a way to escape local optimality which occurs often in local search. The simulation results prove that GA heuristic has the overall best performance but with most expensive search time cost. SA is not as efficient as its application in other domain science problems. More research on choosing efficient fitness values and selection rules are needed.

The conventional ***Sufferage*** heuristic uses MCT as metric for a mapping. "The idea behind

Sufferage is that a host should be assigned to the task that would 'suffer' the most if not assigned to that host." The sufferage value of each task is the difference between the first MCT and the second MCT. The Sufferage heuristic doesn't work well for cluster resources because the MCT on the machines belonging to the same cluster are quite close, which makes the sufferage value approaches zero and eliminates cluster machines from selection. So Xsufferage heuristic developed for an application-level scheduler system– APPLeS [22] computes a cluster-MCT to enable the Sufferage heuristic to work in a cluster environment

- **Distributed vs. Centralized**

In dynamic scheduling scenarios, the responsibility for making global scheduling decisions may lie with one centralized scheduler, or be shared by multiple distributed schedulers. In a computational Grid, there might be many applications submitted or required to be rescheduled simultaneously. The centralized strategy has the advantage of ease of implementation, but suffers from the lack of scalability, fault tolerance and the possibility of becoming a performance bottleneck. Arora et al present a completely decentralized, dynamic and sender-initiated scheduling and load balancing algorithm for the Grid environment. A property of this algorithm is that it uses a smart search strategy to find partner nodes to which tasks can migrate. It also overlaps this decision making process with the actual execution of ready jobs, thereby saving precious processor cycles.

- **Cooperative vs. Non-cooperative**

If a distributed scheduling algorithm is adopted, the next issue that should be considered is whether the nodes involved in job scheduling are working cooperatively or independently (non-cooperatively). In the non-cooperative case, individual schedulers act alone as autonomous entities and arrive at decisions regarding their own optimum objects independent of the effects of the decision on the rest of system. Good examples of such schedulers in the Grid are application-level schedulers which are tightly coupled with particular applications and optimize their private individual objectives. In the cooperative case, each Grid scheduler has the responsibility to carry out its own portion of the scheduling task, but all schedulers are working toward a common system-wide goal. Each Grid scheduler's local policy is concerned with making decisions in concert with the other Grid schedulers in order to achieve some global goal, instead of making decisions which will only affect local performance or the performance of a particular job. An example of cooperative Grid scheduling is presented in [23], where the efficiency of sender-initiated and receiver-initiated algorithms adopted by distributed Grid schedulers is compared with that of centralized scheduling and local scheduling. The hierarchy taxonomy classifies scheduling algorithms mainly from the system's point view, such as dynamic or static, distributed or centralized. There are still many other important aspects forming a scheduling algorithm.

5.2 OBJECTIVE FUNCTIONS

The two major parties in Grid computing, namely, resource consumers who submit various applications, and resources providers who share their resources, usually have different motivations when they join the Grid. These incentives are presented by objective functions in scheduling.

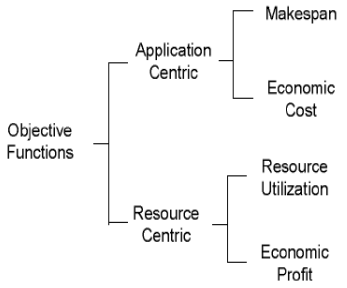

Fig. 3: Objective functions covered in this survey.

The objective functions can be classified into two categories: application-centric and resource-centric [24]. Fig. 3 shows the objective functions.

- **Application-Centric**

Scheduling algorithms adopting an application-centric scheduling objective function aim to optimize the performance of each individual application, as application-level schedulers do. Most of current Grid applications' concerns are about time, for example the 13*makespan*, which is the time spent from the beginning of the first task in a job to the end of the last task of the job. Makespan is the one of the most popular measurements of scheduling algorithms. The primary difficulty facing the adoption of this kind of objective functions lies in the normalization of two different measurements: time and money. Such situations make scheduling in the Grid much more complicated. It is required that Grid schedulers be adaptive enough to deal with such compound missions. At the same time, the development of the Grid infrastructure has shown a service-oriented tendency so the *quality of services* (QoS) becomes a big concern of many Grid applications in such a non-dedicated dynamic environment. The meaning of QoS is highly dependent on particular applications, from hardware capacity.

- **Resource-Centric**

Scheduling algorithms adopting resource-centric scheduling objective functions aim to optimize the performance of the resources.

Resource-centric objectives are usually related to resource utilization, for example, *throughput* which is the ability of a resource to process a certain number of jobs in a given period; *utilization*, which is the percentage of time a resource, is busy. Low utilization means a resource is idle and wasted. For a multiprocessor resource, utilization differences among processors also describe the load balance of the system and decrease the throughput.

5.3 ADAPTIVE SCHEDULING

An adaptive solution to the scheduling problem is one in which the algorithms and parameters used to make scheduling decisions change dynamically according to the previous, current and/or future resource status.

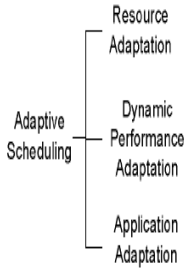

Fig. 5: Taxonomy for adaptive scheduling algorithms in Grid computing.

Resource Adaptation:

Because of resource heterogeneity and application diversity, discovering available resources and selecting an application-appropriate subset of those resources are very important to achieve high performance or reduce the cost.

Dynamic Performance Adaptation:

The adaptation to the dynamic performance of resources is mainly exhibited as: (i) changing scheduling policies or rescheduling [25] [26] (for example, the switching between static scheduling algorithms which use predicted resource information and dynamic ones which balance the static scheduling results), (ii) workload distributing according to application-specific performance models, and (iii) finding a proper number of resources to be used.

Application Adaptation:

To achieve high performance, application-level schedulers in the Grid (e.g. AppLeS [27]) are usually tightly integrated with the application itself and are not easily applied to other applications. As a result, each scheduler is application-specific.

6. CONCLUSION

This literature review, the task scheduling problem in Grid computing is discussed, mainly from the aspect of scheduling algorithms. Although task scheduling in parallel and distributed systems has been intensively studied, new challenges in Grid environments still make it an interesting topic, and many research projects are underway. Through our survey on current scheduling algorithms working in the Grid computing scenario, we can find that heterogeneity, dynamism, computation and data separation are the primary challenges concerned by current research on this topic. We also find that the evolution of Grid infrastructures, e.g., supports for complex application models such as DAG, resource information services and job migration frameworks, provides an opportunity to implement sophisticated scheduling algorithms. In addition to enhancements to classic scheduling algorithms, new methodologies are applied, such as the adaptive application-level scheduling, Grid economic models and Nature's heuristics.

REFERENCES

1. F. Berman, *High-Performance Schedulers*, chapter in The Grid: Blueprint for a Future Computing Infrastructure, edited by I. foster and C. Kesselman, Morganaufmann Publishers, 1998.
2. I. Foster, C. Kesselman and S. Tuecke, *The Anatomy of the Grid*: *Enabling Scalable Virtual Organizations*, in the International J. Supercomputer applications, 15(3), pp.200-220, fall 2001.
3. I. Foster and C. Kesselman (editors), *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1999.
4. F. Chudak and D. Shmoys, *Approximation Algorithms for Precedence Constrained Scheduling Problems on Parallel Machines that Run at Different Speeds*, in J. of Algorithms, vol.30, pp.323--343, 1999.
5. M. Baker, R. Buyya and D. Laforenza, *Grids and Grid Technologies for Wide-area Distributed Computing*, in J. of Software-Practice & Experience, Vol. 32, No.15, pp:1437-1466, December 2002.
6. K. Cooper, A. Dasgupta, K. Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Mazina,J. Mellor-Crummey, F. Berman, H. Casanova, A. Chien, H. Dail, X. Liu, A. Olugbile,O. Sievert, H. Xia, L. Johnsson, B. Liu, M. Patel, D. Reed, W. Deng, C. Mendes, Z.Shi, A. YarKhan and J. Dongarra, *New Grid Scheduling and Rescheduling Methods inthe GrADS Project*, in Proc. of the 18th International Parallel and DistributedProcessing Symposium (IPDPS'04), pp.199--206, Santa Fe, New Mexico USA, April2004.
7. F. Berman, *High-Performance Schedulers*, chapter in The Grid: Blueprint for a Future Computing Infrastructure, Morgan Kaufmann Publishers, 1998.

8. Y. Zhu, *A Survey on Grid Scheduling Systems*, Department of Computer Science, Hong Kong University of science and Technology, 2003.
9. H. Casanova, A. Legrand, D. Zagorodnov and F. Berman, *Heuristics for Scheduling Parameter Sweep Applications in Grid Environments*, in Proc. of the 9th heterogeneous Computing Workshop (HCW'00), pp. 349-363, Cancun, Mexico, May 2000.
10. H. Chen and M. Maheswaran, *Distributed Dynamic Scheduling of Composite Tasks on Grid Computing Systems*, in Proc. of the 16th International Parallel and Distributed Processing Symposium (IPDPS 2002), pp. 88--97, Fort Lauderdale, Florida USA,April 2002.
11. A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C.Kesselman, P. Kunszt and M. Ripeanu, *Giggle: A Framework for Constructing Scalable Replica Location Services*, in Proc. of the ACM/IEEE Conference on Supercomputing, pp.1-17, Baltimore, Maryland USA, November 2002.
12. K. Kurowski, B. Ludwiczak, J. Nabrzyski, A. Oleksiak and J. Pukacki, *Improving Grid Level Throughput Using Job Migration And Rescheduling*, Scientific Programming vol.12, No.4, pp. 263-273, 2004.
13. A. Takefusa, S. Matsuoka, H. Casanova and F. Berman, *A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid*, in Proc. of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01), pp. 406--415, San Francisco, California USA, August 2001.
14. S.Y. You, H.Y. Kim, D. H. Hwang, S. C. Kim, *Task Scheduling Algorithm in GRID Considering Heterogeneous Environment*, in Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA '04, pp.240-245, Nevada, USA, June 2004.
15. K. Cooper, A. Dasgupta, K. Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Mazina,J. Mellor-Crummey, F. Berman, H. Casanova, A. Chien, H. Dail, X. Liu, A. Olugbile,O. Sievert, H. Xia, L. Johnsson, B. Liu, M. Patel, D. Reed, W. Deng, C. Mendes, Z.Shi, A. YarKhan and J. Dongarra, *New Grid Scheduling and Rescheduling Methods in the GrADS Project*, in Proc. of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04), pp.199--206, Santa Fe, New Mexico USA, April2004.
16. S. J. Chapin, D. Katramatos, J. Karpovich and Andrew S. Grimshaw, *The Legion Resource Management System*, in Proc. of the 5th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '99), in conjunction with the International Parallel and Distributed Processing Symposium (IPDPS '99), Lecture Notes in Computer Science, Vol. 1659, pp.162-178**,** San Juan, Puerto Rico, April 1999.
17. D. Wright, *Cheap Cycles from the Desktop to the Dedicated Cluster: combining Opportunistic and Dedicated Scheduling with Condor*, in Proc. of Conference on Linux Clusters: the HPC Revolution, Champaign Urbana, IL USA, June 2001.
18. H. G. Rotithor, *Taxonomy of Dynamic Task Scheduling Schemes in Distributed Computing Systems*, in IEE Proc. on Computer and Digital Techniques, Vol.141, No.1,pp.1-10, January 1994.
19. H. El-Rewini, T. Lewis, and H..Ali, *Task Scheduling in Parallel and Distributed Systems*, ISBN: 0130992356, PTR Prentice Hall, 1994.
20. T. Casavant, and J. Kuhl, *A Taxonomy of Scheduling in General-purpose Distributed Computing Systems*, in IEEE Trans. on Software Engineering Vol. 14, No.2, pp.141--154, February 1988.
21. L He, S.A. Jarvis, D.P. Spooner, D. Bacigalupo, G. Tan and G.R. Nudd, *Mapping DAG-based Applications to Multiclusters with Background Workload*, in Proc. Of IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05), pp.855-862, May 2005.
22. E. Heymann, M. A. Senar, E. Luque and M. Livny, *Adaptive Scheduling forMaster-Worker Applications on the Computational Grid*, in Proc. of the 1st IEEE/ACM International Workshop on Grid Computing, Lecture Notes In Computer Science, Vol. 1971, pp.214-227, Bangalore, India, December 2000.
23. H. Shan, L. Oliker, R. Biswas, and W. Smith, *Scheduling in Heterogeneous Grid Environments: The Effects of Data Migration*, in Proc. of ADCOM2004: International Conference on Advanced Computing and Communication, hmedabad Gujarat, India,December 2004.
24. Y. Zhu, *A Survey on Grid Scheduling Systems*, Department of Computer Science,Hong Kong University of science and Technology, 2003.
25. N. Spring and R. Wolski, *pplication Level Scheduling of Gene Sequence Comparison on Metacomputers*, in the Proc. Of 1998 International Conference on Supercomputing (ICS'98), pp. 141-148, Melbourne, Australia, July 1998.
26. Rob V. van Nieuwpoort, Jason Maassen, Gosia Wrzesi_nska, Thilo Kielmann and Henri E. Bal, *Adaptive Load Balancing for Divide-and-Conquer Grid Applications*, http://www.cs.vu.nl/~kielmann/papers/satin-crs.pdf, submitted to the J. of Super-computing, 2004.

27. F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J.Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su and D.Zagorodnov, *Adaptive Computing on the Grid Using AppLeS*, in IEEE Trans. On Parallel and Distributed Systems (TPDS), Vol.14, No.4, pp.369--382, 2003.

BIOGRAPHY

Malathi.G received the B.E degree in Computer Science and Engineering from Bharathiyar University, in 1999 and the M.E degree in Computer Science and Engineering from Anna University, Chennai in 2007. She is working as an Assistant Professor in the Department of B.E. CSE at Vellalar College of Engineering and Technology, Erode. Her area of interests includes database systems, Data Warehousing and Mining, Computer Networks, Compiler Design and Grid Computing.

Sarumathi.S received the B.E degree in Electronics and Communication Engineering from Madras University, in 1994 and the M.E degree in Computer Science and Engineering from Anna University, Chennai in 2007. She is working as a Professor in the Department of B.Tech IT at K.S.Rangasamy College of Technology, Tiruchengode. Her area of interests includes database systems, Data Warehousing and Mining, Computer Networks, Parallel databases and Multimedia systems. She is a member of ISTE.