APPLICATION LEVEL SCHEDULING (APPLES) IN GRID WITH QUALITY OF SERVICE (QOS)

CH V T E V Laxmi[1], Dr. K.Somasundaram[2]

1,Research scholar, Karpagam University, Department of Computer Science Engineering, Visakhapatnam, Andhra Pradesh,India.,
2,Research Supervisor, Karpagam University, Professor, Department of Computer Science and Engineering, Jaya Engineering College, CTH Road, Prakash Nagar, Thiruninravur,Thiruvallur - Dist, Tamilnadu,

Abstract: *Grid computing is a form of distributed computing that involves coordinating and sharing computational power, data storage and network across dynamic and geographically dispersed organizations [6]. In the computational grid the fundamental key management is resource and workload management services such as discovering of resources, monitoring and scheduling the resources. In this research paper, we approach the problem of grid scheduling through grid scheduling with Quality of Service (QoS). An Application-Level scheduling system (AppLeS) is applied in the grid computing to measure the performance of the application on a specific site resource and utilizes this information to make resource selection and scheduling decisions. In this paper we proposed architecture for Application Level Scheduling system with a resource manager and we also proposed Page fault frequency replacement algorithm (PFFR) for the Application Level Scheduling System as it might exhibit "thrashing".*

Keywords: *Application Level Scheduling, Grid Computing, Grid scheduling, Resource discovery.*

1. INTRODUCTION

Grid computing is a paradigm which is used to provide solutions for engineering sciences, industry and commerce. Grid computing is the collection of computer resources from multiple locations to reach a common goal. A grid Computing can be considered as distributed systems with non-interactive workloads which involves a large number of files. Grids are generally constructed by using general-purpose grid middleware software libraries.

Due to increasing in the number of applications, the utilization of Grid Infrastructure has drastically improved to meet the need of computational, storage and other needs. As a single site cannot simply meet all the resource needs of today's demanding applications, therefore using distributed resources can bring many advantages to the users of applications. It can be an efficient management of heterogeneous, geographically distributed and dynamically available resource by deploying in Grid systems.[6] However the Grid environment can be effectively used through its schedulers, which acts as localized resource brokers.

The Grid computing job can be split into many small tasks. The responsibility of the scheduler is selecting the resources and scheduling the jobs in such way that the user and applications requirements are met, in terms of overall execution time and cost of the resources which are utilized in the scheduling process. [1]

DOI: 10.5121/ijgca.2014.5201

In grid computing, researchers implemented and evaluated six different policies, to demonstrate how the simulator is capable of doing, as well as helped them to understand the dynamics of grid computing system.[4]

2.RELATED WORK

2.1 Grid Scheduling With QoS

Condor, SGE, PBS and LSF are the four systems which are widely used for grid-based resource mange and job scheduling. One major problem with the four systems is their lack of Quality of

Service (QoS) support in scheduling jobs; such a system should take the following issues into account when scheduling jobs:

- Job characteristics
- Market-based scheduling model
- Planning in scheduling
- Rescheduling
- Scheduling optimization
- Performance prediction

2.2 Application Level Scheduling (AppLeS)

AppLeS [2] are an adaptive application-level scheduling system that can be applied to the Grid. Each application submitted to the Grid can have its own AppLeS. The design philosophy of AppLeS is

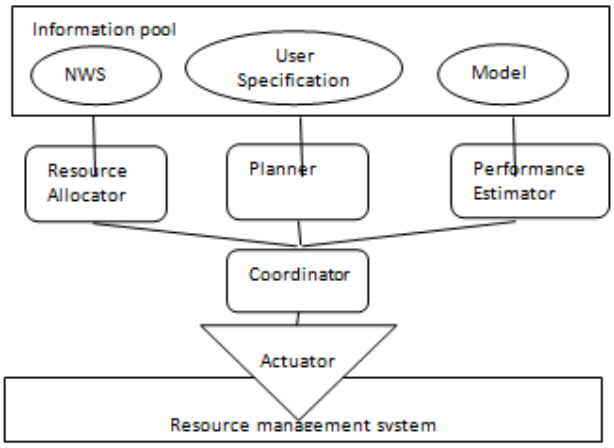

Figure1 Architecture of AppLeS

that all aspects of system performance and utilization are experienced from the perspective of an application using the system. To achieve application performance, AppLeS measures the performance of the application on a specific site resource and utilizes this information to make resource selection and scheduling decisions. Figure 1 shows the architecture of AppLeS.

The AppLeS components are:

- Network Weather Service (NWS) [3]: Dynamic gathering of information of system state and forecasting of resource loads.
- User specifications: This is the information about user criterion for aspects such as performance, execution constraints and specific request for implementation.
- Model: This is a repository of default models, populated by similar classes of applications and specific applications that can be used for performance estimation, planning and resource selection.
- Resource selector: Choose and filter different resource combinations.
- Planner: Planner is used to generate the description of the resource dependent schedule from a given resource combination.
- Performance estimator: performance estimator is used to estimate candidate schedules according to the user's performance metric.
- Coordinator: This component chooses the “best” schedule.
- Actuator: To schedule on the target resource management system this component implements the "best" schedule. When Application Level scheduling system (AppLeS) is used, the following steps are performed.
- The user provides information to AppLeS via a Heterogeneous Application Template (HAT) and user specifications. The HAT provides information for the structure, characteristics and implementation of an application and its tasks.
- The coordinator uses this information to filter out infeasible/ possibly bad schedules.
- The resource selector identifies promising sets of resources, and prioritizes them based on the logical “distance” between resources.
- The planner computes a potential schedule for each viable resource configuration.
- The performance estimator evaluates each schedule in terms of the user’s performance objective.
- The coordinator chooses the best schedule and then implements it with the actuator.

AppLeS differs from other scheduling systems in that the resource selection and scheduling decisions are based on the specific needs and exhibited performance characteristics of an application. AppLeS targets parallel master–slave applications. Condor, SGE, PBS and LSF do not take the application-level attributes into account when scheduling a job. Note that AppLeS is not a resource management system, it is a Grid application-level scheduling system.

2.3Scheduling in Grid Application Development Software (GrADS)

AppLeS focus on per-job scheduling. Each application has its own AppLeS. When scheduling a job, AppLeS assumes that there is only one job to use the resources. AppLes does not have resource managers which can negotiate with applications to balance their interests, which is generating the problem in AppLeS. In the absence of these negotiating mechanisms in the Grid computing leading to various problems, which focuses on improvement of the performance of individual AppLeS. However, there will be many AppLeS agents in a system simultaneously, each working on behalf of its own application. All of the AppLeS agents may identify the same resources as “best” for their applications and seek to use them simultaneously which is a worst-case scenario. When targeted resources are no longer available, then they might seek to reschedule their application on another resource. In this way, multiple unconstrained AppLeS might exhibit “thrashing” behavior and achieve good performance neither for their own applications nor from the system’s perspective. This is called the Bushel of AppLeS Problem.

Grid Application Development Software (GrADS) project [4] provides comprehensive programming environment in order to incorporates application characteristics and their requirements in designing of the application.

3. PROPOSED APPROACH

Application Level Scheduling System (AppLeS) focuses on per-job scheduling and each application has its own AppLeS. When scheduling a job, AppLeS assumes that there is only one job to use the resources. One problem with AppLeS is that it does not have resource managers to manage the resources. The absence of these negotiating mechanisms in the grid can lead to fall in the performance.

However, there will be many AppLeS agents in a system simultaneously, each working on behalf of its own application. All the AppLeS agents may identify the same resources as "best" for their applications and seek to use them simultaneously, which leads to worst case scenario. When the targeted resources are not available, they will reschedule their application to another resource. In this way, multiple unconstrained AppLeS might exhibit "thrashing" behavior and achieve good decisions.

The main aim of this project is to provide integrated grid application development solution which incorporates certain activities like compilation, scheduling, staging of binaries and data, application launching and monitoring during execution time. In GrADS, the Meta scheduler receives candidate schedules of various application level schedulers and implements scheduling policies for balancing different applications as show in Figure 2.

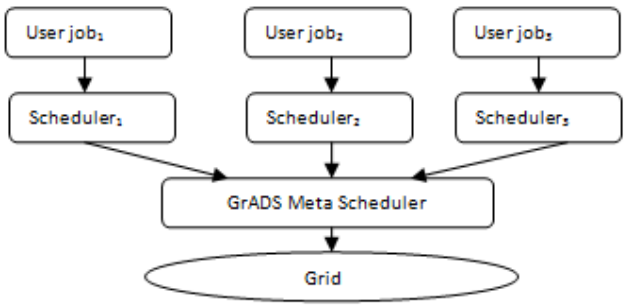

Figure 2 Job scheduling in GrADS

We proposed architecture of Application Level Scheduling (AppLeS) with resource manager in Figure 5, as the AppLeS does not have resource manger which is leading to the fall of performance. In a system there can be many AppLes agents which are simultaneously working on behalf of its own application. All AppLeS agents may identify the same resources as best for their applications and seek to use them simultaneously, which leads to worse-case scenario, and if the targeted resources are no longer available, all has to reschedule their application on another resource. In this way AppLeS might exhibit "thrashing" behaviour and will not achieve good performance neither for their own nor from the system's perspective.

Therefore we proposed Page Fault Frequency Replacement Algorithm (PFFR) to overcome the thrashing behaviour of the Application Level Scheduling in Gird (AppLeS). We can implement

Page Fault Frequency algorithm in the proposed architecture of Application Level Scheduling (AppLeS) with resource manager and we proposed Page Fault Frequency Replacement (PFFR) algorithm in the proposed architecture of Application Level Scheduling (AppLeS) with resource manager, to overcome the thrashing behaviour of the Grid AppLeS and to improve performance of the systems and to improve the proper usage of the available resources.

3.1Thrashing

If the number of process submitted to the CPU for execution, the CPU utilization will also increases. But increasing the process continuously at certain time the CPU utilization falls sharply, the CPU treated this overload and sometimes it reaches to zero. This situation is said to be "Thrashing", Figure 3 demonstrating the thrashing behaviour. The term thrashing denotes that excessive overhead and severe performance degradation or collapse caused by too much paging. Thrashing inevitably turns a shortage of memory space into a surplus of processor time.

For example the main memory consisting of 5 jobs initially at that time the page fault rate is 0.6, after few seconds add the 5 jobs to memory, then the rate will increase to 0.8, after some time add a another 5 jobs, then the page fault rate drops suddenly to 0.1 or 0.2. Sometimes it may be reach to zero. This unexpected situation is said to be "thrashing" which is leaded in the Application Level scheduling (AppLeS).

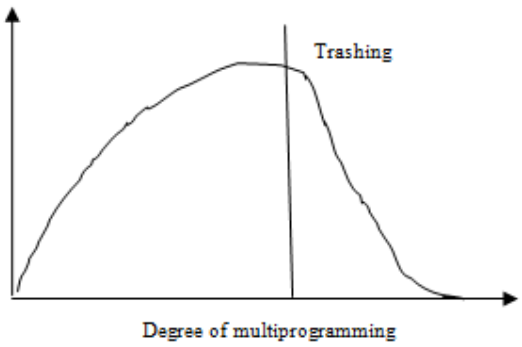

Figure 3 Thrashing

3.2Resource Manager

The resource manager's idea is generic, in that there is a basic behavioural pattern expected from each one of them. The resource managers can be implemented as object-oriented type hierarchy. A base class can be defined that would characterize all the behaviour except for some details.
For example, an audio speaker, which we defined as a resource manager, we inherit behaviour of the base class for the generic resource manager and the details of audio speaker resource we define in the subclass. Therefore we can say that, for allocating resources, the generic part of the resource manager is the key mechanism and resource specific behaviours are determined by their policies.

The resource manager accepts requests from process and allocates units of its resources.

To determine the criteria to allocate the resources to processes, request() function will execute resource specific policy algorithm. For example the resource manager policy of an audio speaker

might forbid sharing of the resources, as multiple processes may play sound to the speaker at the same time. It may also restrict the process that can be allocated to control the audio speaker. For example a processes which is owned by a particular user, may restrict the use of the audio speaker policy. Another example is that the resource manager may pre-empt the speaker from other processes, when a process executing its function and might it has high priority over a process that is playing music from the CD-ROM device. All resource managers have the general form shown in the Figure 4. A process requests units of resources in each case. If the resource manger allocates the resource, then the process continues to run. Otherwise, blocked processes are placed in a pool, to await allocation. After the allocation, the process is removed from the pool and made ready to run.

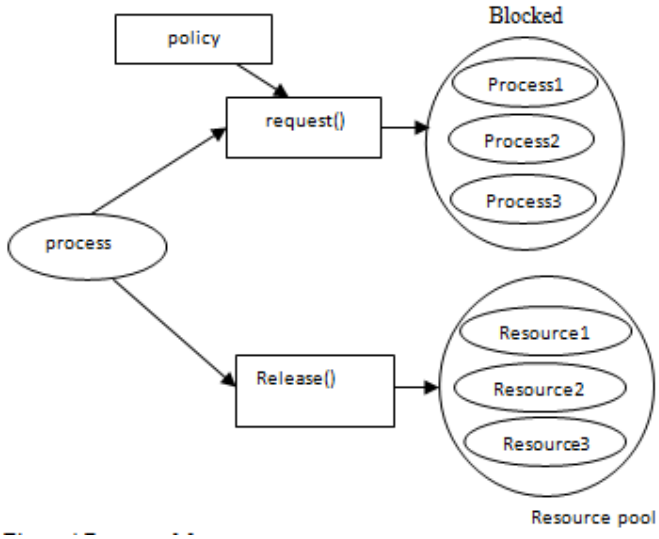

Figure 4 Resource Manager

In this paper we proposed architecture of Application Level Scheduling (AppLeS) with resource manager, as the AppLeS does not have resource manger which is leading to the fall of performance. To get the information about the resources, the resource manager gives permits to the applications to create, delete, open, modify and write the resources. A resource can be treated as data of any kind which are stored in a defined format in the resource file. The Resource Manager provides functions for the proper resource management and it also keeps track of resources present in the memory.

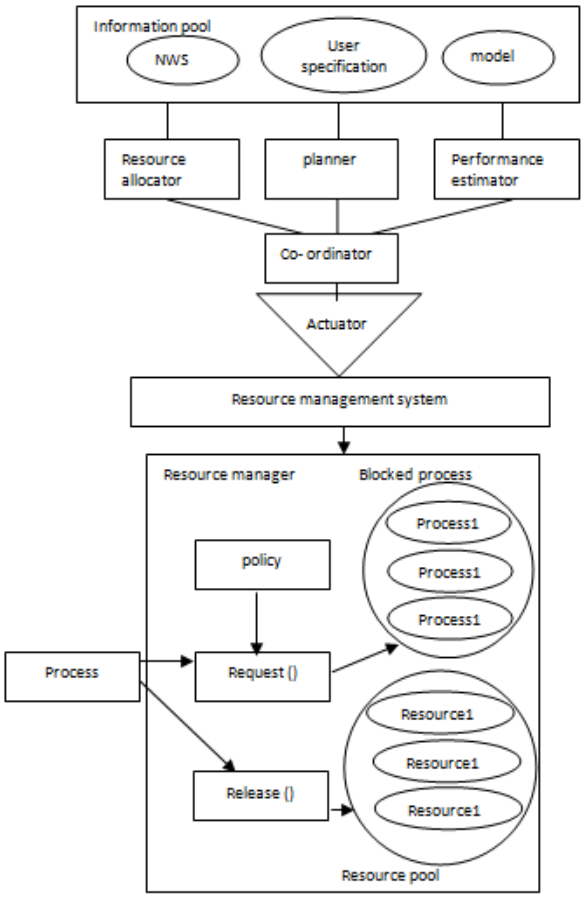

Figure 5 Architecture of Application Level Scheduling (AppLeS) with resource manager

Figure 5 implements Resource manager in the Application Level Scheduling (AppLeS) architecture. The AppLeS architecture does not have resource management system, which leads to the fall in the performance level. However, there can be many AppLeS agents in a system simultaneously, each working on behalf of its own application. A worst-case is that all the AppLeS agents may identify the same resources as "best" for their applications and seek to use them simultaneously, as there is absence of Resource manager in the AppLeS architecture. Therefore the problem can solve by implementing Resource Manager in the Application Level Scheduling (AppLeS) architecture

Each resource manager maintains a resource descriptor, which is a data structure for the resources, it is managing the details of the resource descriptor depend on the resources and the grid scheduler. Table 1 represents the kind of information we can see in a resource descriptor.

Field	Description
Internal resource name	An internal name for the resource used by the grid scheduler
Total units	The number of units of this resource type configured into the system
Available units	The number of units currently available
List of available units	The set of available units of this resource type that are available for use by processes.
List of blocked processes	The list of processes that have a pending request for units of this resource type.

Table 1 Representing information of resource descriptor

3.3The Page Fault frequency replacement algorithm (PFFR)

The Page Fault Frequency Replacement (PFFR) Algorithm uses the measured page fault frequency as the basic parameter for the memory allocation decision process. The main aim of PFFR is to prevent thrashing by allocating or deallocating frames as required, we assume that a high page fault frequency indicates that a process is running inefficiently because it is short of page frames. A low page fault frequency, on the other hand, indicates that a further increase in the number of allocated page frames will not considerably improve the efficiency and, in fact, might result in waste of memory space. Therefore, to improve system performance (e.g., space-time product) one or more page frames could be freed.

The basic policy of the PFFR Algorithm is:

Whenever the page fault frequency rises above a given critical page fault frequency level P, all referenced pages which were not in the main memory, therefore causing page faults, and are brought into the main memory without replacing any pages. This results in an increase in the number of allocated page frames which usually reduces the page fault frequency. On the other hand, once the page fault frequency falls below P, page frames may be freed. The same operation will be repeated whenever the page fault frequency rises above P again. Once a process is removed from the main memory this information can be used to schedule this process for the next, time quantum. In general, a process will be put on the processor queue only if there are enough available page frames in the pool. The information about the memory space of each process can also be used to decide which process has to be removed from the main memory if the page frame pool becomes empty. There are many ways for the supervisor to make use of the information about program behavior provided by the PFF Algorithm.

3.2.1 Implementation of the Page Fault Frequency Replacement Algorithm (PFFR)

PFFR algorithm is very simple to implement. We need only a clock in the CPU to measure the time between page faults of every process. This clock measures the process (or virtual) time of

each process. The current process time is recorded in the process' state word. To determine which pages are residing in the main memory page table entry can be used. For those paging systems that have a USE-BIT feature this feature can be used to determine those pages which have been referenced during the time interval since the last page fault occurred. Whenever a page fault occurs the USE-BITs are reset and the supervisor determine whether the process is operating below the critical page fault frequency level P. For this purpose the time of the last page fault has to be stored. If the last page fault occurred more than T=I/P msec ago, the USE-BITs are used to determine which pages have to be removed from the main memory.

3.2.2 Case Study and Experimental Analysis

Memory requirements may differ from one process to another process, by allocating too few frames to a process may lead to process thrashing, to prevent thrashing, the main aim of the PFFR algorithm is to allocate or deallocate frames as required. The Table 2 represents the PFFR Algorithm Behavior

Reference #	1	2	3	4	5	6	7	8	9	10	11	12
Page referenced	1	2	3	4	1	2	5	1	2	3	4	5
Frames	1	1	1	1	*1	*1	1	*1	*1	1	1	1
*** = reference bit set**		2	2	2	2	*2	2	2	*2	2	2	2
			3	3	3	3						
				4	4	4						
_ = faulting page							5	5	5			
										3	3	3
											4	4
												5

Table 2 PFFR Algorithm Behavior

Algorithm of Page Fault Frequency Replacement (PFFR)

Step1: Initially set every frame reference bit as 0.

step2: set its frame's reference bit, whenever a page is referenced.

step3: compare the IFT(inter-fault time) with a certain threshold, when a page fault occurs.

step4: reset all reference bits, if IFT<threshold, and then allocate a new frame to the process

Step5: If IFT ≥ threshold, reallocate all frames whose reference bit is not set and allocate a new frame for the faulting page, and reset all reference bits.

An experimental example with threshold value with 3.

As the number of frames allocated to a process is dynamic in this algorithm, while performing the analysis, use the mean number of frames in use,per reference.

In the above described example, there are 39 frames in user over 12 page references, therefore the mean number of frames is 39/12=3.25

Analysis for the above example:

12 page references
8 page faults
Page faults per number of frames = 8/3.25 ≈ 2.4615

4. CONCLUSION

This research paper carries out survey on Grid scheduling from various studies of different researchers in grid computing. It includes the problem of grid scheduling through grid scheduling with Quality of Service (QoS). In this research paper, we approached the problem of grid scheduling through grid scheduling with QoS. There are many grid scheduling system which are lack of Quality of Service (QoS) in scheduling jobs. Application Level Scheduling (AppLeS) is an adaptive application-level scheduling system which can be applied to the Grid. AppLeS measures the performance of the application on a specific site resource and utilizes this information to make resource selection and scheduling decisions. This paper focuses on designing of new architecture for AppLeS with a Resource Manager. In this research of grid scheduling with QoS, we proposed new approach for AppLeS which might exhibit “thrashing” through the Page Fault Frequency Replacement (PFFR) Algorithm, which uses the measured page fault frequency as the basic parameter for the memory allocation decision process. This PFF Replacement Algorithm allocates memory, according to the dynamically changing memory requirements of each process. It does not require prior knowledge of program behavior and can be applied to programs of different types and sizes.

REFERENCES

[1] Condor, http://www.cs.wisc.edu/condor/.
[2] Dail H.,Berman F., and Casanova, H. A Decoupled Scheduling Approach for Grid Application Development Environments. Journal of Parallel Distributed Computing, 63(5): 505 524 (2003).
[3] Figueira M., Hayes J., Obertelli G., Schopf J., Shao G., Smallen S.,Spring N., Su A. and Zagorodnov D. Adaptive Computing on the Grid Using AppLeS. IEEE Transactions on Parallel and Distributed Systems, 14(4):369–382 (2003).
[4] Feras Hanandeh,Mutaz Khazaaleh,Hamidah Ibrahim,and Rohaya Latip "CFS: A New Dynamic Replication Strategy for Data Grids" The International Arab Journal of Information Technology, Vol. 9, No. 1, January 2012
[5] Hamscher V., Schwiegelshohn U., Streit A. and Yahyapour R. Evaluation of Job-Scheduling Strategies for Grid Computing. GRID 2000, 191–202, 17–20 December 2000, Bangalore, India. Lecture Notes in Computer Science, Springer-Verlag.
[6] Ian F., and Carl K., “The Grid: Blueprint for a New Computing Infrastructure,” Elsevier Inc., Singapore, Second Edition, 2004.
[7] NWS, http://nws.cs.ucsb.edu/.
[8] Raksha S., Vishnu K.S., Manoj K M., Prachet B., A Survey of Job Scheduling and Resource Management in Grid Computing World Academy of Science, Engineering and Technology 40 2010