Grid Resource Management based on Functional Dependency

Doan Thanh Tran, Eunmi Choi*

School of Business IT, Kookmin University
Jeongneung-dong, Seongbuk-gu, Seoul, 136-702, Korea
td_thanh@yahoo.com, emchoi@kookmin.ac.kr

Abstract. In this paper, we propose a resource management system in Grid computing in order to specify system Quality of Service (QoS) requirements for dynamic and complex emerging applications. Our approach is based on the *functional dependency* among application components to specify the probability of system QoS requirements for the emerging application. Experimental results show that our application scheduling based on *functional dependencies* can achieve scheduling and managing emerging applications to satisfy a client's quality of service in Grid computing. The results also show significant improvement of performance comparing to cluster distribution and random distribution scheduling approaches.

1 Introduction

Large-scale grids are complex systems composed of thousands of components from disjoined domains. Planning the capacity to guarantee quality of service (QoS) in such environments is a challenge because global service-level agreements (SLAs) depend on local SLAs. Thus, resource management is the major concern in Grid. The resource management system (RMS) is central to the operation of a Grid. Resources are the entities such as processors and storage that are managed by the RMS. The set of services provided by a RMS varies depending on the intended purpose of the Grid. The resource management system should predict the impact of applications' requests on the overall resource pool and quality of service guarantees.

Condor [1] is the typical one, not using the prediction method but using the policy rules for matching between requestors and providers. In the prediction and heuristic approach, PBS (Portable Batch System) [2] and LSF (Load Sharing Facility) [3] are two typical schedulers and both are supports batch jobs scheduling. Batch rescheduling allows potentially more effective utilization of the Grid resources since more requests can be considered at one time.

* Corresponding author: Eunmi Choi (emchoi@kookmin.ac.kr). This work was supported by the Korea Science and Engineering Foundation (KOSEF) under Grant No. R04-2003-000-10213-0. This work was also supported by the Brain Korea 21 project in 2006.

This paper proposes a resource management system with the online application modeling approach for Grid Resource Management. The application components are modeled with resource requirement and functional dependency. The resource requirement considers CPU utilization, network load, and storage allocation. The functional dependency is applied to composite web services. The approach to extract dependency structures among application service in this paper is pragmatic and based on a static dependency analysis that yields information on entities within a system. The analysis shows that dependency information is stored in built-in repository of all standard operating systems. Based on the application model, we propose a complete architecture for application service management. This architecture supports resource management and scheduling for composite web services or emerging complex applications, which include many functional dependency components. Through this system, we can achieve scheduling and managing applications to optimize resource utilization while satisfying client's quality of service in Grid computing.

The rest of the paper is organized as follows. We first show the shortcoming of current approaches in Grid Resource Management and propose a new method of application modeling based on functional dependencies in Section 2. In Section 3, we describe the architecture for application service management and scheduling based on the functional dependencies. Finally, we present experimental results of our scheduling approach in Section 4, and conclude this paper in Section 5.

2 Application Modeling Based on Functional Dependency

Normally, resource management applied in Grid computing is used for single application. Compared to the old method of resource management is not applicable in complicated applications, our approach focuses on solving the resource management for dynamic and complex emerging applications.

In this approach, as well as the resource requirements of application components, we consider the online characterization of dependencies. Applying these two requirements to individual application, we create an Online Model Instance for each of application components and, based on this, we estimate the system QoS requirements.

The application model for application services includes: *R*, which is the set of System QoS requirement per application service, and *D*, which is Functional dependency structure among application services. We have the application model with the two vectors be presented as the following expression: $QoSModel = F(D, R)$

For *R* vector, we use Resource Specification Language provided in Globus Toolkits 4.0 [5] as the standard structure to describe the Grid resources and job requests. For *D* vector, we adopt the functional dependency of application services that presented in [4]. In [4], the authors consider the fact that majority of application services run on UNIX and Windows NT based systems and it is worth of analyzing the degree to which information regarding application services is already contained in the operating system. The system administrators successfully deploy application services without having to access to detailed and application-specific management instrumentation because they have this information. We called this information the application service dependencies.

3 The Architecture for Application Service Management

Our objective is to create an automated solution of resource management for resource utilization increased while guaranteeing service level agreement to end users. In this section, we describe the architecture for application service management, the execution flow, and the details of admission control and resource assignment.

The architecture for application service management is shown in Figure 1. As a result of the static analysis during application installation and provisioning, each application service offering has associated with a list of resources that provide the basis of that service. This data is kept in dependency repository and maintained by the application dependency analysis process discussed in previous section and depicted in upper part of Figure 1.

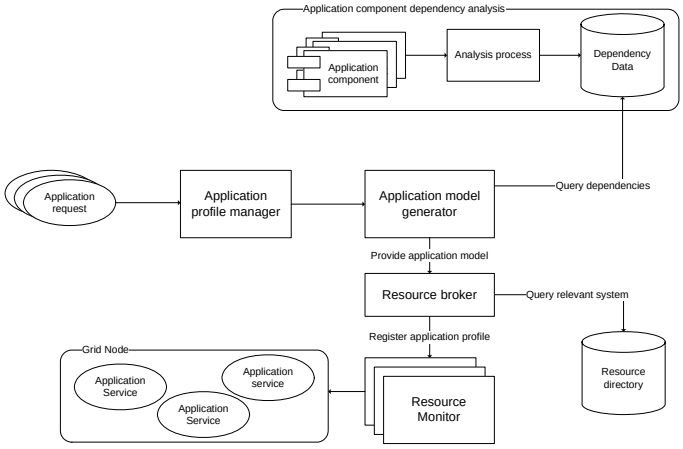


Figure 1. The Architecture for Application Service Management

In the architecture, we have four major components:

- *Application profile manager* is in charge of choosing application requests from end users and submitting it to *Application model generator*.
- *Application model generator* receives application profile from *Application profile manager* and query dependency structure from *Dependency repository* to generate the application model $F(D,R)$. Then it submits the application model to *Resource broker*.
- *Resource broker* receives an application model from *Application model generator*, extracts separate application service component a_i from *R*, and finds suitable grid nodes to process it from *Resource directory*. After choosing best-fit grid node for the application service component, *Resource broker* registers it to *Resource monitor* of that grid node. The application service components are consequently processed by *Resource broker* depending on the dependency structure *D*.
- *Resource monitor* is in charge of executing an application service components registered by *Resource broker* and retrieving resource utilization information to update to *Resource directory*.

3.1 Admission Control

The *Resource broker* performs the admission control for each application service component a_i with resource requirement $\{c, n, s, l_n, l_s\}_{a_i} \in R$ in the request where c is CPU utilization requirement, n is network utilization requirement, s is storage utilization requirement, l_n is network access latency, and l_s is storage access latency.

Supposed that at each grid resource node we have T_c, U_c, T_n, U_n, T_s, U_s which are the total capacity (T) and the utilized capacity (U) of CPU, network bandwidth, and storage. P_C, P_N, P_S are the weight parameters to reserve the resources for unexpected workload of CPU utilization, network utilization, and storage utilization respectively. $0 \le P_C, P_N, P_S \le 1$. Because we cannot assign the full resource capacity for a well-fit job request, these parameters will help ensuring the system work properly.

The admission checking for each grid resource node that has all required resources will be as follow:

- $P_c \times (T_c - U_c) \ge c_{a_i}$, the CPU requirement of a_i must be less or equal to the available CPU capacity $(T_c - U_c)$ of the grid resource node.
- $P_n \times (T_n - U_n) \ge n_{a_i}$, the network requirement of a_i must be less or equal to the available network capacity $(T_n - U_n)$ of the grid resource node.
- $P_s \times (T_s - U_s) \ge s_{a_i}$, the storage requirement of a_i must be less or equal to the available storage capacity $(T_s - U_s)$ of the grid resource node.
- $l_n \le l_{na_i}$, the network latency requirement of a_i must be greater than the network latency of the grid resource node.
- $l_s \le l_{sa_i}$, the storage latency requirement of a_i must be greater than the storage latency of the grid resource node.

3.2 Resource Assignment

Resource assignment is optimized for minimizing wait-time and maximizing utilization of servers. Considering utilization of a server, we need to measure the utilization of CPU, network capacity, and storage capacity. The differences of priority of these resources depend on the application services, the environment, and the policy of the system. Therefore, we try to give a general metric to consider all the resources' utilization as follow

- Fitting metric for each grid node that passes admission checking

Supposed that at each grid resource node we have T_c, U_c, T_n, U_n, T_s, U_s , which are the total capacity and the utilized capacity of CPU, network bandwidth, and storage. The terms c_a, n_a, s_a are the CPU, network, and storage requirement of application a. We have the available capacities of this node: T_c-U_c, T_n-U_n, T_s-U_s. Therefore the utilization rates of CPU, network, and storage of application a on the remaining capacities are $\frac{c_a}{T_c - U_c}, \frac{n_a}{T_n - U_n}$, and $\frac{s_a}{T_s - U_s}$ where $\frac{c_a}{T_c - U_c}, \frac{n_a}{T_n - U_n}, \frac{s_a}{T_s - U_s} \le 1$. The

simple metric is the sum of these utilization rates. However, in many real cases, the priorities of requirements are various. Depending on these priorities, we can set the weights for each utilization rate. For that reason, we choose the fitting metric for an application *a* as follow:

$$M_{fit} = Wc_f \times \frac{c_a}{T_c - U_c} + Wn_f \times \frac{n_a}{T_n - U_n} + Ws_f \times \frac{s_a}{T_s - U_s} \leq 1$$

where $Wc_f + Wn_f + Ws_f \leq 1$, and Wc_f, Wn_f, and Ws_f are weight of CPU, network, and storage. These weights are chosen to optimize wait-time and utilization of servers. Depending on different type of grid resources and application request, we can choose the suitable weights. For example, for the computational grid, the CPU optimization is the most significant. Therefore, we can set the value of Wc_f high for this type of Grid. For the grid requires heavy network communication, we can set the Wn_f high.

- Best fit criterion: Supposed that we have *n* nodes to pass the admission check, the chosen Grid resource node is the node that satisfy the following expression:

$$M_{fit} = \max_{i=1}^{n} M_{fit_i}$$

The idea of this metric is a simple greedy algorithm. The key point is that with weights of requirements, we can optimize the utilization of the system by choosing suitable weights depending on the system and the application types.

4 Experiment Results and Evaluation

In this section, we evaluate the performance of our application allocation approach against cluster allocation approach and random allocation approach. The GridSim simulator [5] is used in our study. For evaluating the system, we consider three criteria: the system throughput, the Grid node utilization, and the job's waiting time.

Table 1. The QoS Model of the Standard Application Requests

Request Type	CPU Utilization	Network bandwidth	Storage	Duration in wall clock time
Heavy composite web service request	15% Guarantee on a 3Ghz machine	15Mbps	15MB	6 hours
Light composite web service request	10% Guarantee on a 3Ghz machine	10Mbps	10MB	1 hour
Heavy batch job request	Minimum threshold of 35% on a 3Ghz machine	0Mbps	300MB	4 hours
Light batch job request	Minimum threshold of 5% on a 3Ghz machine	0Mbps	100MB	3 hours

The major application requests we used in the simulation are composite web service requests. In some test cases, we choose mixed workload of batch and composite web services in accordance with realistic workload cases. We consider two scenarios in the realistic workload cases: day time experiment and night time experiment.

During day time, the workload of a batch job request is light. Only small number of jobs is submitted. The workload of a composite web service request is heavy during day time, so a large number of composite web services are submitted. The arrival rate of these requests has Poisson distribution. During night time, the workload of a batch job request is heavy. They are submitted in large number. In contrast, the workload of a composite web service request is light. There are very few requests submitted.

For the composite web service request, we assume that the relations are handdrafted and auto-provided to the *Application model generator*. We have two types of Composite Web service request structures for two different light and heavy composite web service requests. Table 1 shows the QoS model for the standard application requests in experiment. For the diversity of our request, we apply normal distribution with the random variation of 0-20% of the standard requirements.

All simulation scenarios run in the system which composes 100 Grid resource nodes. Each node has the resource capabilities as follows: Intel Pentium 4 (D850EMVR, 3.06GHz, Hyperthreading Technology Enabled), Operating system Windows 2003 Sever, SPEC/MIPS rating: 1099, Network share: 100Mbps, Storage share: 700MB, Resource manager type: Time-shared. To ensure that the system runs properly, we set the limitation of utilization of each type of resources at 90% of the resource capability. The 10% resource capability is reserved for overhead.

4.1 Composite Web Service Request Simulation

In this type of experiment, we generate only composite web service requests. We compare the request processing with and without using functional dependency. The purpose of this experiment is to compare the performance of our approach of compartmentalizing the composite web service request into a set of sub application services with the performance of cluster allocation and random allocation approaches.

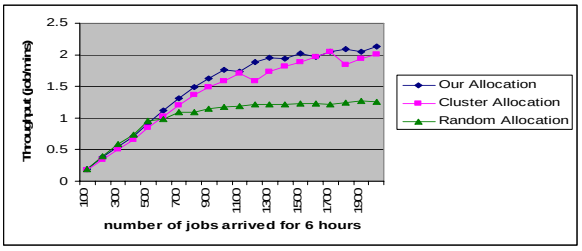


Figure 2. Throughput Results of Our Allocation, Cluster Allocation, and Random Allocation

The input data is heavy request type only and the requirements vary using normal distribution with the variation of 0-20% of those of the standard request described in Table 1. The depth of dependency request is two or three. The requests are sent for six hours from the beginning of the experiment and using Poisson distribution to distribute the arrival rate of the requests to the Grid system. The number of requests ranges from 100 to 2000. The collecting information is throughput, max waiting time, average waiting time, and resource utilization.

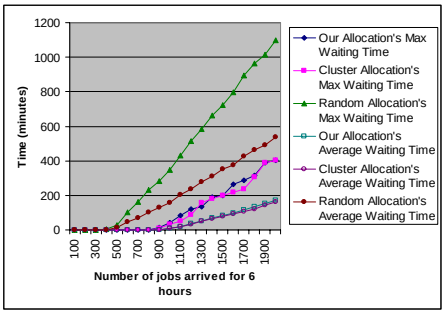
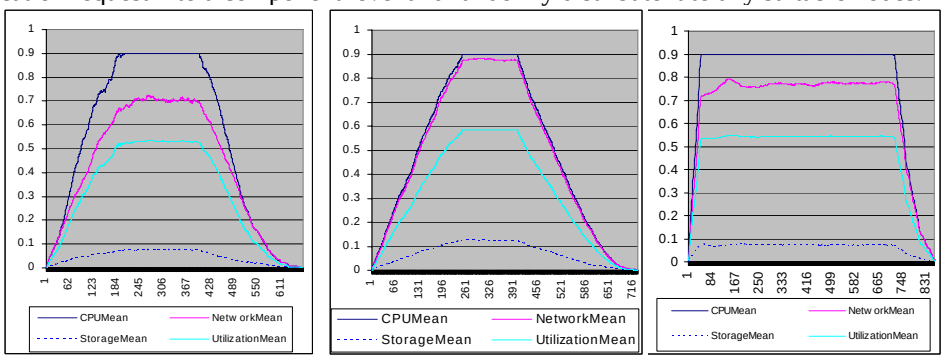


Figure 3. Waiting Time Results of Our Allocation, Cluster Allocation, and Random Allocation

Figure 4. Grid Nodes' Utilization Variance Results of Our Allocation, Cluster Allocation, and Random Allocation.

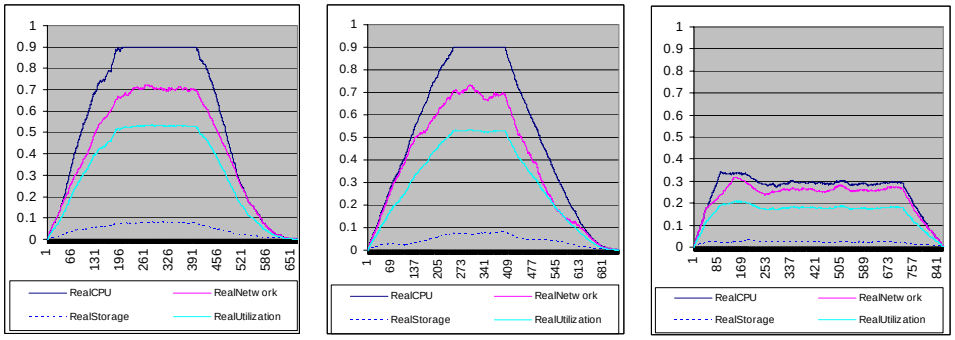
With our allocation, we use the functional dependency to distribute the application components whenever they are ready to be executed. For the cluster allocation, we do not consider the application components and their dependency. Therefore, we assign one request in only one Grid node. For the random allocation, we separate the application request into a component level and randomly distribute it to any suitable nodes.



(a) (b) (c)

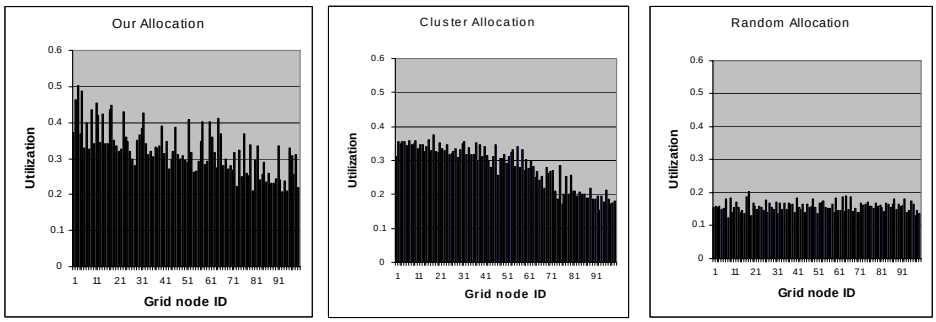
Figure 5. Mean Values of Resource Reservation of Our Allocation (a); Cluster Allocation (b); and Random Allocation (c)

The results (showed in Figures 2, 3, and 4) show that our solution always has the best throughput in most of test cases. The waiting time of our allocation is also the best result. The utilization variance of our allocation is better, compared to the cluster allocation. Due to the busy waiting, the random allocation yields the lowest utilization variance, but the throughput and waiting time are the worst. These results have proved that our system gains better performance when applying functional dependency. Therefore, we choose the case in which the number of generated requests in 6 hours is about 1000 requests to compare the utilization of the system. Figure 5 shows the utilization of the system in the three approaches.



(a) (b) (c)

Figure 6. Mean Values of Real Utilization of Our Allocation (a); Cluster Allocation (b); and Random Allocation (c)



(a) (b) (c)

Figure 7. Grid Nodes' Utilization of Our Allocation (a); Cluster Allocation (b); and Random Allocation (c)

The results in Figures 5 and 6 show that only our approach provides the resource reservation equivalent to the system utilization. Our allocation method comparing to cluster allocation method also provides better resource utilization as in Figure 6 (a) and (b). The random allocation method shows a great difference between resource reservation and resource utilization. Figure 7 also shows that our approach achieves the best utilization, compared to other approaches. Our allocation method has better distributing balance, comparing to cluster allocation method when the random approach achieves the worst utilization.

4.2 Mixing Batch Request and Composite Web Service Request Simulation

The purpose of this experiment is to demonstrate the system in the realistic workload cases in which the grid will have different type of requests submitted at different time period with different requirements. In this experiment, we collect the resource utilization of system, the maximum waiting time, the average waiting time of the composite web service requests, and the throughput of each type of requests.

The day time experiment uses the number of requests which is chosen based on the previous results. We already knew that the simulation system reaches the peak of utilization when the number of heavy web service requests is around 1000. Besides, in day time, there will be not so many batch job requests submitted to the system. Therefore, we set 1000 heavy composite web service requests and 200 light batch job requests. The heavy composite web service requests are submitted within 6 hours from the beginning of the experiment and generated with the arrival rate based on Poisson distribution.

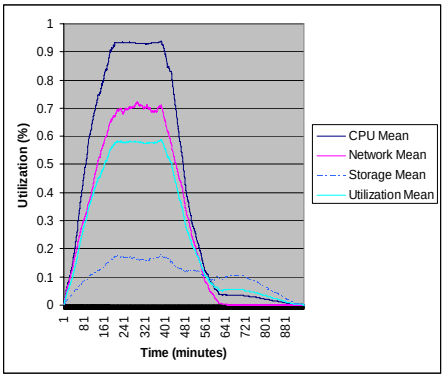


Figure 8. Mixed Request Simulation – Day Time Utilization Results

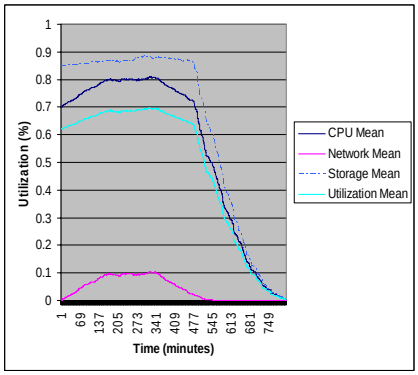


Figure 9. Mixed Request Simulation – Night Time Utilization Results

The results show that all the composite web services have average waiting time 7 minutes (max is 29 minutes). Comparing to the 4 hour-request duration, this is an acceptable results. Besides, Figure 8 shows that the utilization of the system and the variance of grid nodes' utilization are as good as the experiment results only with composite web services. In the night time experiment, we consider that we have a list of heavy batch job requests already submitted and, during experiment time, there will be a small number of light composite web services submitted to the system. The night time results in Figure 9 show that our system can achieve high performance when scheduling for a major number of Batch job requests with a small number of composite web service requests.

Using the GridSim toolkit, we developed a simulator to experiment our new approach on grid resource management. Based on the simulator, we created different test scenarios to compare processing performance of composite web service requests of our allocation algorithm with that of standard cluster allocation algorithm and that of random allocation algorithm. The results showed that our approach always provides best utilization performance, throughput, and reasonable utilization variance between grid nodes. Our allocation approach which is based on functional dependency always provides better results comparing to standard cluster allocation approach while the random allocation approach showed the worst utilization and throughput.

For the practical of our experimentation, we consider the realistic workload cases in which the composite web service requests are processed collaterally with batch job requests. In this type of experiment, we divide into two test cases: day time experi-

mentation and night time experimentation. All the results showed that our approach could provide high utilization performance even in the artificial real-world cases.

5 Conclusion

In this paper, we propose an online application modeling approach for grid resource management. The approach uses functional dependency in the application model. The proposed architecture of application service management, based on the functional dependency structure, can structuralize the application request into a set of dependency application components. Based on this functional dependency, we develop a scheduling algorithm to distribute the application requests in Grid computing. This algorithm uses online application model instance generation approach to generate the application model for each application request. The scheduling algorithm uses this model to optimize the resource assignment process. To validate our scheduling algorithm, we develop a simulator based on the GridSim toolkit. The simulating results have proved that our approach shows significant improvement of performance, comparing to cluster allocation and random allocation approaches.

References

1. Raman, R. and Livny, M.. (1998). “ Matchmaking: Distributed Resource Management for High Throughput Computing”, Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, Chicago, IL.
2. Henderson, R. and Tweten, D.. (1996). "Portable Batch System: External reference specification," Technical report, NASA Ames Research Center.
3. W. Allcock, J. Bester, J. Bresnahan, S. Meder, P. Plaszczak, S. Tuecke. (2003). “ GridFTP: Protocol Extensions to FTP for the Grid”. Global Grid ForumGFD-RP
4. G. Kar, A. Keller, and S. Calo, (2000) “Managing Application Services over Service Provider Networks: Architecture and Dependency Analysis,” in Proceedings of NOMS 2000, Honolulu.
5. R. Buyya and M. Murshed, (2002) “GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing”, CCPE, Volume 14, Issue 13-15, Wiley Press, Nov.-Dec..
6. SPEC CPU2000 Results, http://www.spec.org/cpu2000/results/cpu2000.html
7. Klaus Krauter, Rajkumar Buyya, Muthucumaru Maheswaran. (2001). “A taxonomy and survey of grid resource management systems for distributed computing”, Software: Practice and Experience Volume 32, Issue 2
8. Ian Foster, Carl Kesselman, and Steven Tuecke. (2001). “ The Anatomy of the Grid - Enabling Scalable Virtual Organizations”, Intl. Journal of Supercomputing Applications.
9. I. Foster and C. Kesselman. (1997). “ Globus: A Metacomputing Infrastructure Toolkit.”. Intl Journal of Supercomputer Applications, Volume 11, No. 2.