Adaptive Workflow Scheduling Strategy in Service-Based Grids*

JongHyuk Lee[1], SungHo Chin[1], HwaMin Lee[2], TaeMyoung Yoon[1], KwangSik Chung[3], and HeonChang Yu[1,**]

[1] Dept. of Computer Science Education, Korea University
{spurt, wingtop, tmyoon, yuhc}@comedu.korea.ac.kr
[2] The Korean Intellectual Property Office
hwamin@kipo.go.kr
[3] Dept. of Computer Science, Korea National Open University
kchung0825@knou.ac.kr

Abstract. During the past several years, the grid application executed same jobs on one or more hosts in parallel, but the recent grid application is requested to execute different jobs linearly. That is, the grid application takes the form of workflow application. In general, efficient scheduling of workflow applications is based on heuristic scheduling method. The heuristic considering relation between hosts would improve execution time in workflow applications. But because of the heterogeneity and dynamic nature of grid resources, it is hard to predict the performance of grid application. In addition, it is necessary to deal with user's QoS as like performance guarantee. In this paper, we propose a service model for predicting performance and an adaptive workflow scheduling strategy, which uses maximum flow algorithms for the relation of services and user's QoS. Experimental results show that the performance of our proposed scheduling strategy is better than common-used greedy strategies.

Keywords: adaptive grid scheduling, workflow, maximum flow.

1 Introduction

In the mid 1990s, Grid computing has emerged as an important new field, distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and high-performance orientation [1]. Grid computing system [2] consists of large sets of diverse, geographically distributed resources that are grouped into virtual computers for executing specific applications. In common Grid computing, resource components could be processes, processors within a computer, network interfaces, network connections, entire sites, database, file system and specific computers. Today, Grid computing offers the strongest low cost and high throughput solutions [1, 2] and is spotlighted as the key technology of the next generation Internet. Grid computing is used in fields as diverse as astronomy, biology, drug discovery, engineering, weather forecasting, and high-energy physics.

* This work was supported by the Korea Research Foundation Grant funded by the Korean Government(MOEHRD) (KRF-2006-D00173).
** Corresponding author.

C. Cérin and K.-C. Li (Eds.): GPC 2007, LNCS 4459, pp. 298 – 309, 2007.

Recently, the Grid and Web Service are converging as WSRF (Web Service-Resource Framework)[3] that defines a system for creating stateful resources between Web services in terms of an implied resource pattern. The current methodology in Grid computing is service oriented architecture.

In service-based Grids, Grid resources are virtualized as services(e.g., database, data transfer). So the Grid not only provides computational resource and data resource, but also supports logic application that cooperates with services integration with the composition of the Grid service. Instead of application executing a single job, Grid application consists of a collection of several dependency services. Therefore, many grid applications belong to the category of workflow application. Most of science and business grid applications take the form of linear workflow structure. That is, the science grid application is a parameter sweep application processed using same code for different data, and the business grid application is a transaction application that queries at databases, processes data, and stores in database. Because of processing data in parallel with extensive parameter bounds, workflow application is of benefit to performance. In service-based Grids, it is necessary to consider a relation of services for execution performance because a linear workflow application executed parallel jobs via several services on one or more hosts.

It is easy for workflow structure not only to compose services but also to visualize, verify, schedule, execute, and monitor services. Many kinds of workflow management systems are developed for grid workflow applications. There are two steps for producing workflow. The first step is a service composition to use workflow language and the second step is a scheduling to map sub-task to service. In general, an efficient scheduling of workflow applications is based on heuristic scheduling method. The heuristic considering relation between hosts would improve execution time in workflow applications. But due to the heterogeneity and dynamic nature of grid resources, it is hard to predict the performance of grid application. In addition, it is necessary to deal with user's QoS like performance guarantee.

In this paper, we propose service model for predicting performance and adaptive workflow scheduling strategy, which uses maximum flow algorithms for considering the relation of services and user's QoS.

The rest of the paper is as follows. In section 2, we state a scheduling problem and propose a service model for predicting performance. Section 3 describes the novel strategy to execute the workflows adaptively. In section 4, we present an experimental evaluation of our scheduling by comparing it with existing scheduling strategies. Section 5 presents related works. In section 6, we conclude the paper and discuss some future works.

2 Problem Statement

Workflow scheduling system is to translate application task graph into service graph in computing environment.

2.1 Task Graph

A task graph is an abstract workflow that represents an application as a general model of directed acyclic graph. It is represented as follows;

$G^T = (V^T, E^T)$
V^T : the set of tasks
E^T : the set of edges between tasks that represent a partial order among them

The fact that an edge $e_{i,j}$ is a partial order between task v_i and v_j means that a task v_j is executed after completing a task v_i. In case a task v_i and v_j are a same parent, two task can be executed parallelly. Representing G^T as matrix M of size $v \times v$, $d_{i,i}$ is a computation cost of v_i, and $d_{i,j}$ is a communication cost between v_i and v_j. In this paper, we assume that a task graph implies a start task and a end task.

2.2 Service Graph

A service graph is an directed weighted graph of services in grid computing environments. It is represented as follows.

$G^S = (V^S, E^S)$
V^S : $\{s_1, s_2, \ldots, s_n\}$ the set of services that can be executed at available node
E^S : the set of edges between services

A service graph is a complete connected graph. V^S denotes a computation performance and E^S denotes a communication performance between services. A k-th service node that executes service s_j is $s_{j,k}$. The computation cost of task v_i at service $s_{i,k}$ is $w_{i,j,k}$. If service $s_{i,k}$ can't execute task v_i, then $w_{i,j,k} = \infty$. The communication cost between service node $s_{m,k}$ for task v_i and $s_{n,k}$ for v_j is $c_{i,m,k|j,n,k}$.

2.3 Performance Criteria

Application completion time is consist of computation time and communication time. We assume that grid application executes task t_1 and t_2 sequentially. A task graph is composed with two nodes and one edge between them. That is, $G^T = (\{t_1, t_2\}, E^T)$. For mapping this task graph to service graph, we have to search service s_1 and s_2 that can process task t_1 and t_2. That is $G^S = (\{s_1, s_2\}, E^S)$. If service s_1 completes before communicating with s_2, completion time of this application is defined as follows.

$$\textit{completion time} = \textit{communication time}(A, s_1) + \textit{computation time}(s_1) + \textit{communication time}(s_1, s_2) + \textit{computation time}(s_2) + \textit{communication time}(s_2, A) + \textit{computation time}(A) \quad (1)$$

Grid application A invokes service s_1 and the result of service s_1 is sent to service s_2. Service s_2 processes a task and the result of service s_2 is sent to grid application A. In practice, completion time is determined according to a node that a service is executed in. Therefore, completion time of a node about some service should be predicted and be applied for mapping task graph to service graph.

For predicting completion time of grid service, it is necessary to select optimized service according to performance model described the characteristics of service and to compose workflow. In addition, we need to consider not only scheduling using information of physical resource, but also supporting user's QoS. Hence, in this paper, the performance model is considered as follows.

- service static model : considering a static information of resources like CPU, memory, disk space, and network bandwidth.
- service dynamic model : Owing to influencing service performance by resource capability directly, considering a dynamic information of resources like available CPU, available memory, available disk space, available network bandwidth, and network latency. We also consider the predicted resource status using service patterns like service reservation, frequency of service use, and service throughput.

Since Grid is free of participation and withdrawal of a node, it is necessary that grid service scheduler predicts the performance of a service and applies it dynamically. In this paper, we use a statistical method to predict the performance of a service. Regression is a statistical method that supports relationships between variables and is an appropriate method for predicting an effect about a cause. In regression, the dependent variable(y) that is an effect and the independent variable(x) that is a cause denote as $x \rightarrow y$. That is, the relation between x and y is represented as follows;

$$y = \beta_0 + \beta_1 x + \varepsilon \tag{2}$$

where β_0 is a constant; β_1 is a coefficient of regression; ε is an error rate.

After regression analysis, we can determine a relationship between a dependent variable and an independent variable. If we applied this regression technique with performance as a dependent variable and each resource status as an independent variable, we can predict the performance of a service that participates newly in Grids using existing regression coefficient. In our work, we use a multiple linear regression that allows the modeling of multiple independent variables, which are information of resources defined by service model in Grids.

We consider static and dynamic physical elements x_i such as CPU, memory, disk space, network bandwidth, service reservation, frequency of service use in a service model. The service throughput (y_s), the equation applied these elements to multiple regression, is as follows.

$$y_s = \beta_0 + \sum_{i=1}^{n} \beta_i x_i + \varepsilon \tag{3}$$

where β_0 is a constant; β_i is a coefficient of regression; n is a count of elements; ε is an error rate.

Table 1 is an example data for performance model using multiple linear regression that is executed in same service. The Independent variables are CPU, CPU available, memory available, disk available, and network bandwidth. The dependent variable is throughput. Table 2 is a model summary that multiple linear regression is done. As shown in Table 2, this model can be explained well because coefficient of determination(R Square) is 0.971. That is, the strength of the linear association between independent variables and dependent variable of this model is high. As shown in Table 3, F-test is 93.634 and significant probability is 0.000. Therefore, the

one of regression coefficients in the population is not 0 at least. Table 4 is regression coefficients about each independent variable. We can predict a throughput of new entrance node using these coefficients.

Table 1. Example data for performance model

CPU	CPU available	Memory available	Disk available	Network bandwidth	Throughput
1600	.80	234	3320	25	40
1800	.40	346	4592	35	28
2000	.60	78	9295	29	33
2400	.40	321	2934	90	34
1600	.50	398	2039	34	45
…	…	…	…	…	…
3000	.30	455	3945	10	36

Table 2. Model summary

R	R Square	Adjusted R Square	Std. Error of the Estimate
.985(a)	.971	.961	3.678

Table 3. ANOVA(Analysis Of Variance between groups)

	Sum of Squares	df	Mean Square	F	Sig.
Regression	6333.602	5	1266.720	93.634	.000
Residual	189.398	14	13.528		
Total	6523.000	19			

Table 4. Coefficients

	Unstandardized Coefficients		Standardized Coefficients	t	Sig.
	B	Std. Error	Beta		
(Constant)	-41.266	5.793		-7.124	.000
CPU	.016	.002	.510	8.863	.000
CPU_available	64.981	5.261	.724	12.350	.000
memory_available	.026	.004	.339	5.826	.000
disk_available	.000	.000	.037	.704	.493
network_bandwidth	.015	.037	.022	.417	.683

3 Adaptive Scheduling Using Dynamic Maximum Flow Algorithm

It is important to select a computation node and a data node for minimizing overall job completion time. It is necessary to minimize completion time for processing data and communication time between computation node and data node. Moreover, it is essential to optimize use of resource through scheduling algorithm. Our objective is to minimize overall job completion time and to optimize use of resource. For our objective, we present an adaptive scheduling using dynamic maximum flow algorithm that finds a flow of maximum value in flow network G with source *s* and sink *t*.

The adaptive workflow scheduling algorithm presented in Algorithm 1 works as follows. The input of WorkflowScheduling in Algorithms 1 is task graph G^T and service level agreement SLA which involve user's QoS. G^T is mapped to service graph G^S by SLA and resource performance criteria. Then Algorithm 2 is invoked with G^S. MaximumFlow in Algorithm 2 is based on Ford-Fulkerson method[9] which finds some augmenting path p and increases the flow f on each edge of p by the residual capacity $c_f(p)$. Algorithm 3 based on breadth-first search is to find augmenting path in residual network of G^S. FindAugmentingPath in Algorithm 3 assumes that the input graph G^S is represented by adjacency lists in descending order by sufferage heuristic value. Migration in Algorithm 1 is a function that migrates the tasks through comparison of flow before rescheduling with flow after rescheduling if a performance guarantee is violated. After all tasks executed, scheduler updates service's makespan(e.g. throughput) for performance criteria.

```
WorkflowScheduling(G^T, SLA)
    G^s ← Find available services satisfied SLA about G^T
     MaximumFlow(G^S)
     while all tasks not executed
         do Fetch task
            if a performance guarantee is violated
                then do update V^s[G^s]
                     MaximumFlow(G^s)        // rescheduling
                     Migration(G^s, G^s_prev)
     update service's makespan
```

Algorithm 1. Workflow Scheduling

```
MaximumFlow(G^S) // find maximum flow about workflow G^S
    for each edge (s_i, s_j) j E^S[G^S]
         do f[s_i, s_j]   0
            f[s_j, s_i]   0
    while (there exists a path p from start service to end
           service in the residual network G^S)
         // min{c_f(s_i, s_j) : (s_i, s_j) is in p}
         do c_f(p)   FindAugmentingPath(G^S, source, sink)
              for each edge (s_i, s_j) in p
                  do f[s_i, s_j]   f[s_i, s_j] + c_f(p)
                     f[s_j, s_i]   -f[s_i, s_j]
```

Algorithm 2. Maximum Flow

```
FindAugmentingPath(G^S, source, sink)
    for each vertex u j V[G^S] - {source}
         do color[u]   WHITE
    color[source]   GRAY
    Enqueue(Q, source)
    c_f[source]   -1
    while Q ≠ 0
         u = Dequeue(Q)
        // Adj[u] is sorted by sufferage value
         for each v j Adj[u]
        do if (color[v] == WHITE &&
                capacity[u][v] - flow[u][v] > 0)
          then color[v]   GRAY
                Enqueue(Q, v)
                c_f[v] = u
          color[u]   BLACK
return c_f;
```

Algorithm 3. Find Augmenting Path

For example, assume that Grid application A is composed of task T_B, T_C, and T_D. The number of service nodes for tasks T_B, T_C, and T_D is 2, 3, and 1 respectively. The linear workflow and the workflow mapped service are represented in Fig. 1. The edge capacity of workflow is calculated by performance criteria.

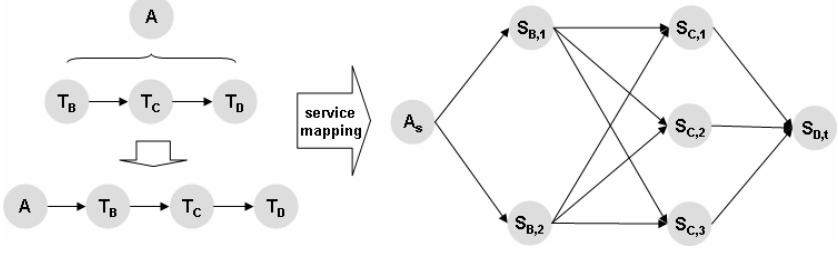

Fig. 1. Linear workflow and workflow mapped service

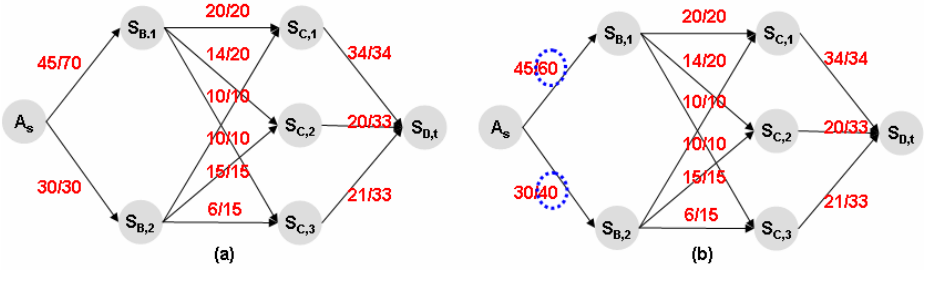

Fig. 2. Result through performance modeling and maximum flow

Fig. 2(a) is a result through performance modeling and MaximumFlow in Algorithm 2. The edge of workflow denotes 'flow/capacity'. The capacity of 70 between A_s and $S_{B,1}$ means that $S_{B,1}$ can process requested job of A_s at the throughput rate of 70. If a performance guarantee is violated, the workflow scheduler reschedules after updating current capacity of workflow. Fig. 2(b) is the result of rescheduling. As shown in Table 5, the maximum flow increases. If the maximum flow decreases, it means that a new service node should be added.

Table 5. Service order and comparison of flow before rescheduling and flow after rescheduling

Service order	Flow before rescheduling	Flow after rescheduling
$A_SB_1C_2D_t$	10	10
$A_SB_1C_1D_t$	20	20
$A_SB_1C_3D_t$	15	15
$A_SB_2C_1D_t$	14	14
$A_SB_2C_2D_t$	10	10
$A_SB_2C_3D_t$	6	15

4 Experiment

Although experiments and performance evaluations need to be performed in a practical large-scale grid platform, it is difficult to build a large-scale grid platform and to experiment repeatedly. Therefore, we simulate our scheduling algorithm using SimGrid toolkit and experiment performance of real grid application implemented a service based virtual screening system in practical small-scale grid environments.

Simulation scenario is classified into two categories: adding service and adding task. In this paper, we compare our scheduling with greedy heuristic scheduling that allocates more tasks to node with better performance. Performance prediction scheduling is greedy heuristic scheduling with performance model described in this paper. Experiment workflow is a generic science workflow that searches, downloads, processes data, and stores result in Fig. 1.

4.1 Performance Evaluation According to the Number of Nodes for Services

In Grid workflow, the number of nodes for service A requesting workflow is 1, the number of nodes for service D collecting results is 1, the number of nodes for service B is 3, and the number of nodes for service C is 5, 10, 15 in each experiments. The number of tasks is 5,000. Fig. 3 shows the result of evaluation. As shown in Fig. 3, our scheduling is better than other algorithms by 15% ~ 20%. The difference of execution time between case that the number of nodes for service C is 10 and case that the number of nodes for service C is 15 is small. It is because the collection of service C could process mostly data from the collection of service B in the former. Therefore, although the number of nodes for service increases in some collection of service, the efficiency of performance doesn't increase. Through our scheduling, we predict a sudden change of efficiency in that the number of nodes for service C is 10.

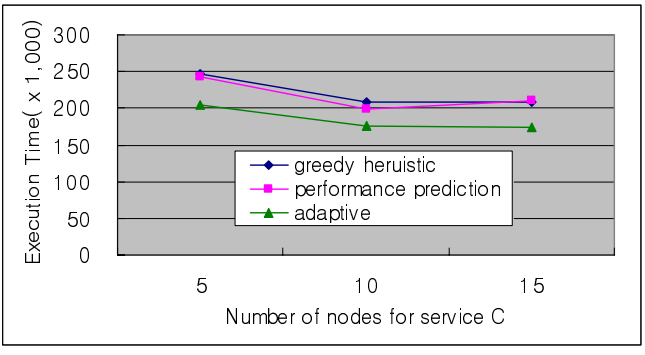

Fig. 3. Result of performance evaluation according to the number of nodes for service C

4.2 Performance Evaluation According to the Number of Tasks

In Grid workflow, the number of nodes for service A requesting workflow is 1, the number of nodes for service D collecting results is 1, the number of nodes for service B is 3, and the number of nodes for service C is 10. The number of tasks is from 1,000 to 11,000 at intervals of 2,000. Fig. 4 shows the result of evaluation. As shown in Fig. 4, our scheduling is better than other algorithms by 10% ~ 15%.

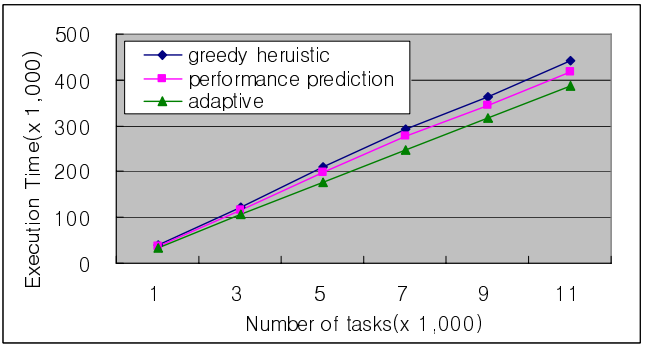

Fig. 4. Result of performance evaluation according to the count of tasks

4.3 Performance Evaluation in Real Grid Application

We implemented a service-based virtual screening system which is one of large-scale scientific applications that require large computing power and data storage capability. A virtual screening is the process of reducing an unmanageable number of compounds to a limited number of compounds for the target of interest by means of computational techniques such as docking [10, 11]. Thus this application suits with Grid computing technology which supports a large data intensive operation.

We experimented our virtual screening system in a testbed that consists of 15 computation nodes and 5 data nodes. We performed docking jobs with 30,000 ligand molecules on a target receptor. Fig. 5 shows the comparison of execution times as the number of docking jobs increases. We compared three different approaches to execute docking jobs. The first approach is to execute docking jobs on only single node which has the best computing performance. The second approach is to execute docking jobs on selected 5 computation nodes. We selected 5 computation nodes according to high computing performance. The third approach is to execute docking jobs using our Grid service-based virtual screening system applied our scheduling. Fig. 5 shows that the performance of our virtual screening system is better than other approaches. When 30,000 docking jobs were executed, the execution time of first approach was 587,541 seconds, the execution time of second approach was 221,516 seconds, and third approach was 162,964 seconds.

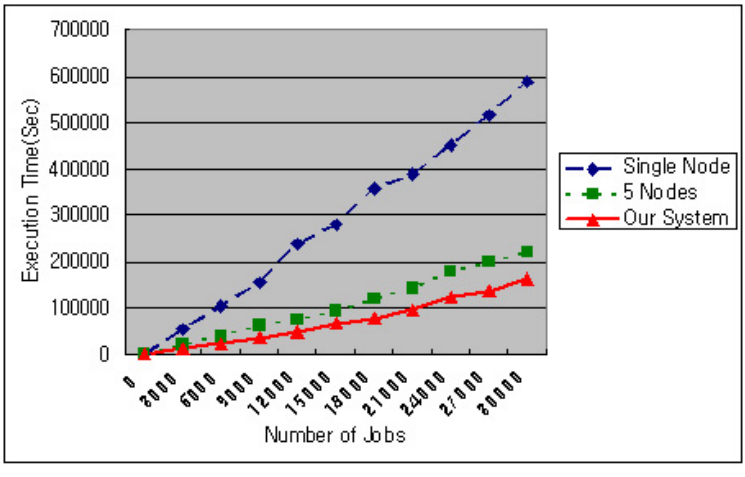

Fig. 5. Comparison of execution time for three cases

5 Related Works

Grid Scheduling is a superscheduling[4] or metascheduling that is the process of scheduling resource where that decision involves using multiple administrative domains. Scheduling is classified into a static scheduling and a dynamic scheduling according to a point of scheduling time. The static scheduling resolves the order of all jobs before executing jobs. The dynamic scheduling can modify the order of jobs in runtime.

In [5], Muthucumaru et al gives an overview of two types of mapping heuristics: on-line and batch mode heuristic. These heuristics are dynamic mapping heuristics for a class of independent tasks in heterogeneous distributed computing. In online mode, mapper allocates tasks to resources as soon as it arrives at the mapper. In batch mode, mapper collects tasks until calling mapping events and allocates tasks to resources after calling mapping events. In particular, sufferage heuristic is newly proposed,

which is different with min-min, max-min heuristic[6]. Sufferage value is defined as difference between minimum earliest completion time and second earliest completion time. In [7], Casanova et al extends sufferage heuristic as Xsufferage. In XSufferage, the sufferage vaule is computed not with minimum earliest completion time, but with cluster-level minimum earliest completion time, which is important in Grid environment. In [8], Eduardo et al proposed the GridWay framework which executes and schedules efficiently parameter sweep application in Grid environment. This framework applied adaptive scheduling to reflect the dynamic Grid characteristic, adaptive execution to migrate running jobs to better resource, and reuse of common file to reduce file transfer overhead. [5] and [7] are a static scheduling and [8] is a dynamic scheduling. But [5], [7], and [8] can't support the form of workflow. In this paper, we support the dynamic scheduling of dependent task using sufferage value.

6 Conclusion

In this paper, we proposed adaptive scheduling strategy for parallel execution of a linear workflow considering dynamic resource in service-based Grids. We presented a performance model using regression technique and an adaptive scheduling strategy using maximum flow algorithm. Our experiments showed that our scheduling is better than other algorithms.

In the future, we plan to investigate our scheduling strategy at commercial point of view as shown in performance evaluation according to the number of nodes for services. We also plan to work on applying not only linear workflow but also complex workflow.

References

1. I. Foster, C. Kesselman and S. Tuecke, The Anatomy of the Grid : Enabling Scalable Virtual Organizations, International Supercomputer Applications, Vol. 15, No. 3 (2001)
2. Ian Foster, and Carl Kesselman, The Grid : Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers (1998)
3. K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, S. Tuecke, From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution,
4. http://www.ibm.com/developerworks/library/ws-resource/ogsi_to_wsrf_1.0.pdf, (2004)
5. J.M. Schopf, Ten Actions when SuperScheduling, Global Grid Forum Document GFD.04, July (2001)
6. Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hensgen, and Richard F. Freund, Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems, Proceedings of the 8th Workshop on Heterogeneous Computing Systems (HCW '99), San Juan, Puerto Rico, Apr. (1999)
7. O. Ibarra and C. Kim, Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors. Journal of the ACM, 24(2):280-289, (1977)
8. Casanova, H., Legrand, A., Zagorodnov, D., and Berman, F., Heuristics for Scheduling Parameter Sweep Applications in Grid Environments, Proceedings of the 9th Heterogeneous Computing Workshop (HCW'00), pp. 349-363, (2000)

9. Eduardo Heudo, Ruben S. Montero, Ignacio M. Lorente, Experiences on Adaptive Grid Scheduling of Parameter Sweep Applications, Proceedings of the 12th Euromicro Conference on Parallel Distributed and Network-Based Processing(EUROMICRO-PDP'04), (2004)
10. Lestor R. Ford, Jr. and D. R. Fulkerson, Flows in Networks, Princeton University Press, (1962)
11. Jordi Mestres and Ronald Knegtel, Similarity versus docking in 3D virtual screening, Journal of Perspectives in Drug Discovery and Design, Vol. 20, (2000)
12. Shoichet, Bodian, and Kuntz, Molecular docking using shape descriptors, Journal of Computational Chemistry, Vol. 13, No. 3, pp. 380-397, (1992)