

Adaptive Resource Selection for Grid-enabled Network Services

Byoung-Dai Lee and Jon B. Weissman
Department of Computer Science and Engineering,
University of Minnesota, Twin Cities
{blee, jon}@cs.umn.edu

Abstract

Due to the popularity of high-speed networks and advances in packaging and interface technologies, there has been significant efforts for providing high performance applications as network services that can be accessed remotely across the network, thus promoting sharing of both software and hardware. For high-demand network services, in particular, it will often be the case that the network services are installed at multiple sites so that each participating site can handle parts of client requests. We label such services as Grid-enabled network services. In this paper, we present two adaptive site selection heuristics that do not depend on accurate predictions of completion times of service requests.

1. Introduction

As complexities and scales of domain problems in science and engineering areas increase, high performance applications have played a critical role in solving and modeling the problems. Areas including fluid dynamics, molecular dynamics, quantum chemical reaction dynamics, and global climate modeling, to name a few, are actively using high performance applications to explore and simulate interesting phenomena. Recently, due to the popularity of high-speed networks and advances in packaging and interface technologies ([1][2][8][17][18]), there has been significant efforts for providing high performance applications as network services that can be accessed remotely across the network, thus promoting sharing of both software and hardware. High performance applications such as data mining [7], theorem proving and logic [5], parallel numerical computations [4][12] are example services that are going on-line.

For high-demand network services, in particular, it will often be the case that the network services are installed at multiple sites so that each participating site can handle parts of client requests. We label such services as Grid-enabled network services. The first step of resource management for Grid-enabled network services is to select an appropriate site to handle the request. Once a site is selected, the site will assign an appropriate number of resources to the request. Therefore, Grid-enabled network services necessitate effective coordination of participating sites among multiple (and often concurrent) service requests to provide scalable performance for a wide spectrum of users. Otherwise, some of the sites would become overloaded while others remain idle. One way of selecting a site is to predict the completion time of the request on each site and select the one that is predicted to finish the request earliest. However, if each site employs dynamic resource management such as Shortest-Job-First (SJF) scheduling [14] or resource harvesting [9], it is often difficult to predict the completion time of the request because the queued requests as well as incoming requests affect the order of the request execution.

In this paper, we present two adaptive site selection heuristics that do not depend on accurate predictions of completion times of service requests. We have deployed prototype service for N-body simulation and evaluated the performances of the heuristics against the Round-Robin selection policy.

The rest of the paper is organized as follows. Section 2 presents related work. Section 3 describes the system model. Section 4 explains the proposed heuristics in detail and Section 5 presents the experimental results. Finally, Section 6 concludes and discusses future work.

2. Related Work

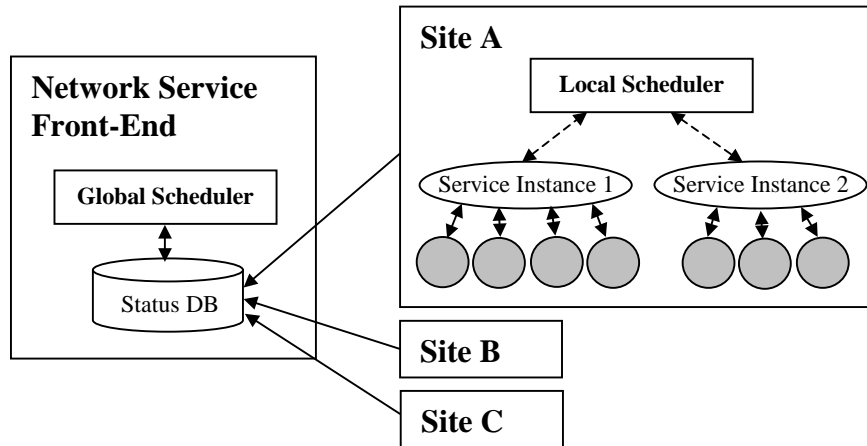


Figure 1. The architecture Grid-enabled network services: The example network service uses resources at three sites: A, B, and C. At site A, service instance 1 is using four resources and service instance 2 is using three resources.

Significant research has been conducted in the area of resource selection in the Grid. Much of this work, however, is based on the accurate predictions of the completion times of jobs when selecting resources for the jobs ([4][12][13][15][16][19][21]).

[3] proposes a scheduling algorithm that is driven by a user supplied application deadline and a resource access budget. The algorithm selects resources in such a way that user requirements are met (e.g. deadline), and yet it keeps the cost of computation at the minimum.

[6] is a de facto standard for building Grid computing environments. Recent adoption of Web service technology as part of the standardization process indicates that the service-oriented view will be dominant computing paradigm for the future. While providing mechanisms for invoking, launching, and replicating network services, it does not specify any policies for resource management in order to improve the usability of the system.

[4] and [12] are two representative network service infrastructures that bear strong similarities both in motivation and in general design. For each user request, a scheduler (or an agent in [4]) selects a set of servers that can handle the computation and ranks them based on the minimum completion time.

[11] presents a general-purpose resource selection framework that supports both single-resource and multiple-resource selection. Their work extends the Condor matchmaking framework [10]. By allowing users to specify the characteristics of resources that they want to use, it can increase flexibility and usability of the framework.

3. System Model

The high performance network services that we are considering are stateless data parallel services. Data parallel services are those that require communication between constituent processes and the communication patterns are symmetric. Many existing science and engineering applications including numerical solvers, N-body simulation, and parallel CFD, etc. can be transformed and provided as data parallel services with ease. Since processes of data parallel services are tightly coupled, using resources that span multiple sites may result in poor performance due to large network latency. Therefore, we assume that a single request will be satisfied with a single site. However, for loosely coupled distributed services (e.g. parameter studies), the assumption can be related.

Figure 1. illustrates the architecture of Grid-enabled network services. End-users send service requests to the front-end of the network service and the front-end is responsible for redistributing the service request to different sites each offering dedicated and space-shared resources. We assume that each site contains a cluster that is homogeneous, but clusters at different sites may be heterogeneous. For example, clusters may have different number of processors of different speed, different amount of memory and disk, and different interconnections.

The front-end consists of two components: a global scheduler and a status database. The global scheduler decides to which site it will forward the request while the status database maintains the up-to-date status information (e.g. current queue length) of each site. Note that the front-end does not decide consider communication times between clients and participating sites when it makes the decision since we assume that the computation time of the

service request is the dominant component of the service time. When a number of service requests arrive simultaneously, the front-end might become a performance bottleneck. However, by simply having each site report its status to multiple front-ends, it can be easily replicated to multiple sites eliminating the performance bottleneck. The infrastructure installed at each site also consists of two components: local scheduler and service instances. When a service request arrives at a site, the local scheduler of the site decides how many resources will be allocated to the request and instantiates the request if resources are available. Otherwise, the request will be queued. Note that in what follows, we assume that each site employs shortest-remaining-time resource harvesting technique for its local resource management. The idea behind shortest-remaining-time resource harvesting is that when a new service request can finish earlier than active service instances, resources of the service instances can be harvested for the new request to enable it to run. The detailed algorithm and experimental results can be found in [9].

4. Adaptive Site Selection

Grid-enabled network services can provide scalable performance by distributing service requests over multiple sites, and yet they are viewed as a single system to users. Resource management for Grid-enabled network services is complex because resources are distributed and heterogeneous. For example, since each site can have different resource capacities, without proper distributions of service requests to each site, some of the sites would become overloaded quickly resulting in a dramatic increase of the service time. The global scheduler may choose a site using the estimated completion time of the request. However, under dynamic resource scheduling, where priorities of requests can change dynamically over time, it is not always possible to acquire an accurate prediction of completion time of the request because the queued requests as well as incoming requests affect the order of the request execution. For example, under shortest-remaining-time harvesting, resources of a long-running request can be harvested for short-running requests. Therefore, the completion time of the long-running request depends on the short-running requests that will arrive after the long-running request starts, which is difficult to know in advance. In addition, in order to predict the completion time of the request, the global scheduler must simulate the dynamic resource management system of each site whenever reordering is required, which is computationally expensive. Therefore, we present two adaptive site selection heuristics: *Weighted Queue Length Heuristic* and *Multi-level*

Queue Based Heuristic, which do not depend on accurate predictions of completion time of requests.

4.1 Weighted Queue Length Heuristics (WQL)

In WQL, the global scheduler utilizes the queue lengths reported periodically by each site and the weights associated with each. The weight is in inverse proportion to the speed with which each site can complete sample requests. The basic idea of WAL is that it presumes that a site with a shorter queue will typically finish a service request earlier. In addition, since each site has a different resource capacity, the estimated queue length is normalized with the weight.

When a service request arrives, the global scheduler will select a site using the following equation:

$$\min_{1 \leq i \leq k} \left\{ (n_{\text{queued}}^i + n_{\text{forwarded}}^i) \times w^i \right\}$$

n_{queued}^i : the number of requests queued at site i
 $n_{\text{forwarded}}^i$: the number of requests forwarded to site i
during current time interval
 w^i : weight associated with site i
 k : the total number of site

Once the global scheduler selects a site and forwards the request to the site, it increments $n_{\text{forwarded}}^i$ and when each site reports its status to the front-end, then $n_{\text{forwarded}}^i$ of each site is reset to 0.

The above equation may not be complete in that it considers only the number of requests queued, not the run-times of each service request. In particular, under dynamic resource scheduling, a site with a shorter queue does not always guarantee shorter service time. For example, if each site is running SJF scheduling, for a short-running request, selecting a site (A) with many long-running requests pending will be better than a site (B) with a few short-running requests pending since it is more likely that at A, the request will have higher priority over requests pending. However, in certain cases, for example, when the run-times of incoming requests keep increasing, the equation holds. WAL bears the case in mind and regards the run-time of the request that needs scheduling as longest at each site. Since the long-running requests typically affect the overall performance, placing those requests on the site with a shortest queue could reduce wait-times of other requests as well as that of the request. Furthermore, if the request is not actually the longest one, then the local scheduler will serve the request faster than expected by the global scheduler.

4.2 Multi-level Queue Based Heuristics (MLQ)

T_1	T_2	T_3	T_n
Site A LB = 10,000 UB = 20,000 QL = 2	Site A LB = 10,000 UB = 23,333 QL = 3	Site A LB = 10,000 UB = 25,000 QL = 5	Site A LB = 10,000 UB = 20,000 QL = 3
Site B LB = 20,000 UB = 30,000 QL = 10	Site B LB = 23,333 UB = 26,666 QL = 12	Site B Unavailable QL = 10	Site B LB = 20,000 UB = 30,000 QL = 3
Site C LB = 30,000 UB = 40,000 QL = 4	Site C LB = 26,666 UB = 40,000 QL = 4	Site C LB = 25,000 UB = 40,000 QL = 6	Site C LB = 30,000 UB = 40,000 QL = 2

Figure 2. Multi-level Queue Based Heuristic: LB and UB represent the lower bound and upper bound of run-times assigned to each site, respectively. QL denotes the queue length reported at the start of each time period.

Under dynamic resource management, the priorities of requests can change dynamically over time. For example, in shortest-remaining-time harvesting, short-running requests have higher priorities over long-running requests. Therefore, it is likely that the wait-times of long-running requests exceed their maximum wait time thresholds because they must yield resources for short-running requests. However, once the requests have waited for their wait time thresholds, the priorities of those requests become higher than those of any other requests irrespective of requests lengths. Otherwise, long-running requests will suffer from starvation. The downside of the dynamic changes of priorities of requests is that if there

are many such requests pending that initially had low priorities but have upgraded to have higher priorities, then incoming requests should wait until all such requests finish. If the priorities of the incoming requests are higher than the initial priorities of those requests, the wait times of the new requests will increase dramatically. In MLQ, by grouping requests with similar characteristics together and forwarding them to the same site, the requests with higher priorities would suffer less from dynamic changes of priorities. Furthermore, by assigning faster sites to requests with lower priorities, faster service time for low priority requests can be expected.

In MLQ, the range of run-time is assigned to each site. When a new request arrives, the global scheduler

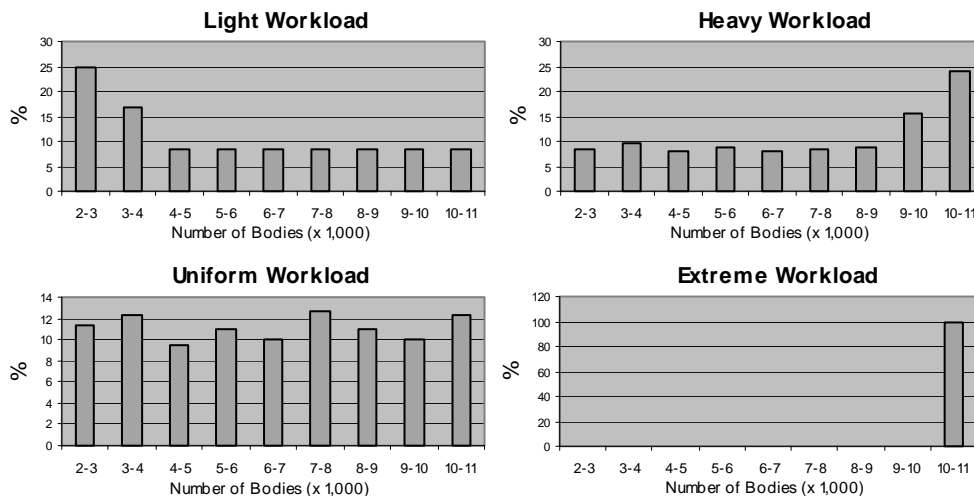


Figure 3. Synthetic workloads.

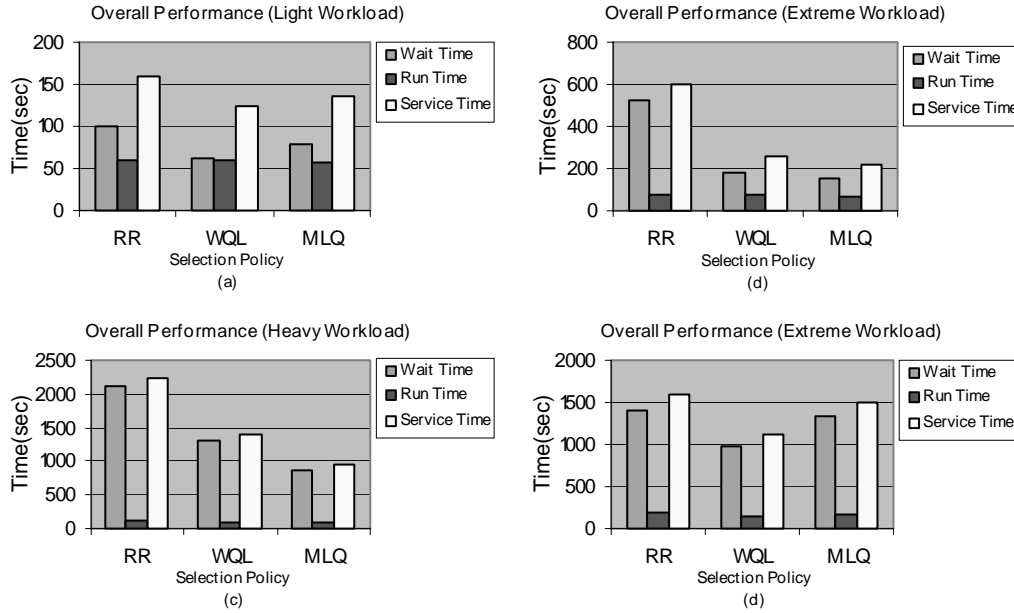


Figure 4. Comparative performance for different selection policies.

predicts the run-time of the request and forwards it to the site whose range includes the predicted run-time of the request. The range will be assigned in such a way that the site providing faster resources will handle long-running requests. Note that MLQ predicts the run-times of requests in order to characterize the requests. Thus, other information such as input parameters to the service also could be used for request characterization. When many new requests have similar run-times, for example, a certain site will become overloaded quickly. To address the problem, MLQ checks the status of each site periodically and if the ratio of maximum queue length and minimum queue length is above a threshold, it changes the range of each site adaptively so that the overloaded site will serve smaller fractions of client requests. In addition, if a site remains overloaded even after its range has been changed, MLQ will eventually mark the site as “Unavailable” until the site finishes most of the service requests queued at the site. Figure 2 illustrates how MLQ works. At T_1 , the scheduler detects that B is overloaded compared to other sites. It decides to shrink the range assigned to B. As a result, some service requests will be forwarded either to A or C, which otherwise would have been directed to B. At T_2 , B is still overloaded, which makes the scheduler black B. Now, only A and C are actively servicing the incoming service requests. Since both A and C handle requests that would have been forwarded to B, B will eventually finish most of its queued requests (T_3). At T_n , the scheduler re-assigns the range to B so that all of the sites handle incoming

requests. Note that the range of each site at T_n is not necessarily the same as those at T_1 . It depends on the client workload observed.

5. Experimental Results

We have developed a N-body simulation service to evaluate the performances of the proposed heuristics. The objective of N-body simulation is to find the positions and movements of the bodies in space that are subject to gravitational forces from other bodies using Newtonian laws of physics [20]. N-body simulation service is implemented using the Master/Slave paradigm, where the master maintains a bag of tasks and slaves repeatedly get tasks, update the bodies, and then return the result to the master. To use N-body simulation service, the users submit four parameters: start time, end time, delta time (the length of the time interval), and input bodies. The testbed where N-body simulation service has been deployed consists of three heterogeneous clusters (Table 1).

Table 1. Testbed specification.

	#(Proc.)	Machine Info.	OS	Weight
Linux	8	Intel Pentium III 750Mhz	Linux	1.2
Solaris	5	UltraSparcIII 900Mhz	Solaris	1.4
SGI	2	Octane2 600Mhz	IRIX6.5	3.1

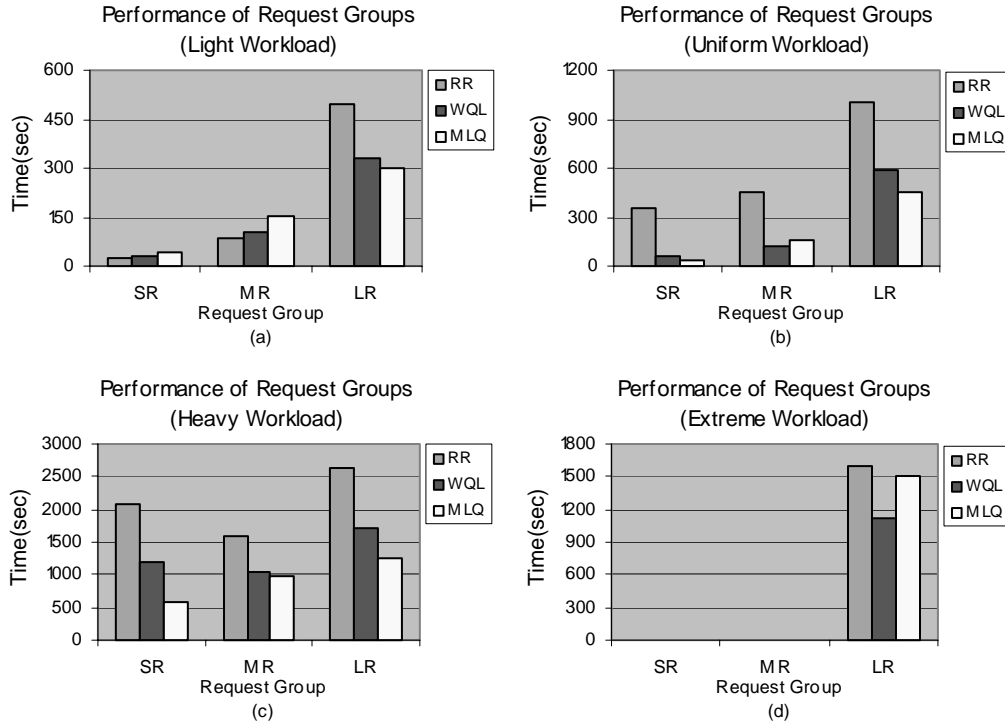


Figure 5. Performance of request groups under different workloads.

Since there is no real trace for network services, we generated several synthetic workloads: Light, Uniform, Heavy and Extreme workloads (Figure 3). The x-axis in the graph represents the number of bodies submitted to the service and the y-axis represents the percentage of the number of requests of each category in the total number of requests. Note that since we fixed three time-related parameters (start time, end time, delta time), which determine the number of iterations, the number of bodies controls the run-time of the service request. For example, the more bodies requested, the longer it takes to compute the results.

We compare the performance of the proposed heuristics against Round Robin selection policy. For each workload, we measured average wait time, average run-time, and average service time of each selection policy (Figure 4). In addition, in order to explore how each policy handles requests of different run-times, we divided service requests into three groups and measured the average service time of each group (Figure 5). Table 2 shows the request groups.

Table 2. Service request groups.

	SR	MR	LR
Num. of Bodies	2,000-5,000	5,000-8,000	8,000-11,000

For example, SR group contains service requests such that the number of bodies submitted is between 2,000 and 5,000.

The threshold of MLQ that triggers dynamic changes of range is set to 2.0 for all experiments. For each workload, both WQL and MLQ outperformed round-robin selection policy since they considered the resource capacity of each site when scheduling service requests. WQL and MLQ achieved performance improvement up to 37% and 58% respectively. As expected, each heuristic shows different performances under different workloads. WQL shows better performance under light workload and extreme workload while MLQ outperforms other policies under medium workload and heavy workload. In MLQ, sites providing faster resources are assigned ranges to service long-running requests. Under light workload, this assignment leads to less use of faster sites because the percentage of long-running requests is small. As a result, slower sites will serve more requests than faster sites. This is shown Figure 5(a). The performance of LR group in MLQ is better than that in WQL but for SR and MR groups, MLQ performs worse than WQL. As the percentage of long-running requests increases, it is more likely that many requests will wait until their maximum wait time threshold due to either yielding resources for requests with higher priorities or longer execution times of

requests in the workload. In such case, the priorities of those requests will eventually be upgraded in order to prevent the starvations of the requests. Since WQL considers only the queue lengths when scheduling requests, short-running requests could be forwarded to a site where many upgraded long-running requests are pending, thus resulting in dramatic increase of wait-times of short-running requests. In MLQ, by sending requests with similar priorities to the same site, short-running requests will be less impacted by the long-running requests whose wait-time have been exceeded the maximum threshold. In addition, it is less likely that the requests must yield their resources for requests with high priorities, which leads to decrements in wait-times of long-running requests. Figure 5(b) and 5(c) show that the performance of every request group in MLQ outperforms that in WQL. Under extreme workload, however, since all of the requests in the workload are in LR group, each site will be equally overloaded with requests with similar characteristics. Therefore, forwarding requests based only on characteristics of requests will not perform well because the selected site may have more requests pending than others.

6. Conclusions and Future Work

Due to the popularity of high-speed networks and advances in packaging and interface technologies, it is possible for software components to be shared across the network through encapsulation and offered as network services. For high-demand network services, it will often be the case that the network services are installed at multiple sites so that each site can handle parts of client requests. In such Grid-enabled network services, the first step in resource management is to select an appropriate site to handle the request. In this paper, we present two adaptive site selection heuristics, Weighted Queue Length Heuristic and Multi-level Queue Based Heuristic, which do not depend on accurate predictions of completion times of requests. We evaluated the performances of the heuristics using an N-body simulation service and achieved performance improvement up to 37% and 58% against round robin selection policy.

As experimental results show, each heuristic shows different behaviors under different client workloads. One future area of interest is to develop algorithms that choose the site selection policies dynamically based on previous information about the performance of each heuristic and client workload observed.

Acknowledgement

This work was sponsored in part by the Army High Performance Computing Research Center under the

auspices of the Department of the Army, Army Research Laboratory cooperative agreement number

References

- [1] R. Armstrong et al., "Towards a Common Components Architecture for High-Performance Scientific Computing", *Proceedings of 8th International Symposiums on High Performance Distributed Computing*, 1999.
- [2] R. Bramley et al., "A Component-Based Services Architecture for Building Distributed Applications", *Proceedings of 9th International Symposiums on High Performance Distributed Computing*, 2000.
- [3] R. Buyya et al., "A Deadline and Budget Constrained Cost-Time Optimization Algorithms for Scheduling Task Farming Applications on Global Grid", *Proceedings of Parallel and Distributed Processing Techniques and Applications*, 2002.
- [4] H. Casanova and J. Dongarra, "NetSolve: A Network Server for Solving Computational Science Problems", *International Journal of Supercomputing Applications and High Performance Computing*, 11(3), 1997.
- [5] D. Dill, SVC: The Standard Validity Checker, <http://www.sprout.stanford.edu/SVC>.
- [6] Globus, <http://www.globus.org>.
- [7] R. L. Grossman et al., "The Preliminary Design of Papyrus: A System for High Performance Distributed Data Mining over Clusters, Meta-Clusters and Super-Clusters", *Proceedings of KDD-98 Workshop on Distributed Data Mining*, 1998.
- [8] K. Keahey and D. Gannon, "PARDIS: CORBA-based Architecture for Application-Level PARAllel DIStributed Computation", *Supercomputing '97*, 1997.
- [9] Byoung-Dai Lee and Jon B. Weissman, "Adaptive Resource Management for Network Services", *Proceedings of 3rd International Workshop on Grid Computing*, 2002.
- [10] M. Lizkow et al., "Condor- A Hunter for Idle Workstations", *Proceedings of 8th International Conference on Distributed Computing Systems*, 1998.
- [11] Chuang Liu et al., "Design and Evaluation of a Resource Selection Framework for Grid Applications", *Proceedings of 11th International Symposiums on High Performance Distributed Computing*, 2002.
- [12] H. Nakada et al., "Design and Implementation of NinF: Towards a Global Computing Infrastructure", *Journal of Future Generation Systems*, Metacomputing Issue, 1999.
- [13] A. Takefusa et al., "A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid", *Proceedings of 10th International Symposiums on High Performance Distributed Computing*, 2001.
- [14] A. Silberschatz and P.B. Galvin, "Operating System Concepts", Addison-Wesley, 1998.
- [15] J. Subholk et al., "Automatic Node Selection for High Performance Application on Networks", *Proceedings of 7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 1999.

- [16] S. S. Vadhiyar and J. Dongarra, "A Metascheduler for the Grid", *Proceedings of 11th International Symposiums on High Performance Distributed Computing*, 2002.
- [17] S. Vinoski, "CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments", *IEEE Communication Magazine*, 14(2), 1997.
- [18] Web Service Description Language (WSDL) 1.1 W3C Note, <http://www.w3.org/TR/wsdl>.
- [19] Jon B. Weissman and Byoung-Dai Lee, "The Virtual Service Grid: An Architecture for Delivering High-End Network Services", *Concurrency: Practice and Experience*, 14(4), 2002.
- [20] B. Wilkinson and M. Allen, "Parallel Programming", Prentice Hall, 1999.
- [21] E. W. Zegura et al., "Application-Layer Anycasting: A Server Selection Architecture and Use in a Replicated Web Services", *IEEE/ACM Transactions on Networking*, 8(4), 2000/