

Distributed Resource Discovery in Large Scale Computing Systems*

Abhishek Gupta[†] Divyakant Agrawal Amr El Abbadi
Department of Computer Science
University of California at Santa Barbara

Abstract

There has been significant effort to build high throughput delivering computing systems out of distributed workstations. These systems are growing to accommodate larger number of workstations with growing demand. Discovery of available resources in such environments is a challenging problem. We present a completely distributed resource discovery solution which utilizes P2P design to provide a scalable service. Our design allows jobs to search for desired workstations, as well as, workstations to search for jobs that may run on them.

1. Introduction

Desktops and workstations are becoming more powerful and more ubiquitous. These powerful machines are usually not utilized to their full capabilities and have enough idle cycles available for use. There has been significant effort to build high throughput delivering systems out of such workstations. These high throughput systems are gaining popularity among scientists who need to run large simulations. The systems are growing to accommodate larger numbers of workstations with growing demand. Discovery of available resources in such environments is a challenging problem. The Grid [2] is a large scale distributed computing environment supporting scientific applications that require high throughput computation. The computing resources in the grid are workstations and supercomputers distributed across the Internet. The resources are contributed to the system by a number of different autonomous entities. Applications have different requirements for the number and capabilities of the resources they need in order to complete their execution. File sharing applications in the wide-area Internet settings have successfully employed P2P design of applications (eg. Gnutella [5], KaZaA [7], Chord [14],

Tapestry [15], CAN [12]). P2P systems have proven to be highly scalable and highly available which is indicated by the popularity of such file sharing applications among the users. In this paper, we apply the P2P design principles in the context of computation sharing environments to provide scalable resource discovery service in large scale high throughput systems.

Condor [8] and the computation grid [2] (such as Globus [4]) are excellent examples of systems utilizing idle or dedicated cycles from distributed resources for providing high throughput computing. Building such distributed systems that provide high throughput computing involve challenges like *resource discovery*, *checkpointing*, and *process migration*. Checkpointing has been studied in [1] in the context of Condor. A survey of process migration techniques appears in [9]. In this paper we will be concentrating on *resource discovery* which is a fundamental problem in large scale distributed high throughput computing systems.

The problem of resource discovery is that given an application with a requirement specification of the types of computing resources, locate the machines out of the available machines that meet the specified criteria. The problem of resource discovery has been studied in [10, 11] in the context of Condor [8]. Raman et al. [10] present a solution based on classified ads for jobs and machines in the system. The jobs and machines submit ads describing their characteristics and requirements to the matchmaking server which periodically runs the matching algorithm to find matches between jobs and machines. This is extended in [11] to match multiple entities. The solution is very good for small scale systems like across a LAN or within a single administrative domain. The solution however will not scale to large systems with thousands of machines and jobs because of the centralized matching process. Condor-G [3] employs a user supplied list of resource management servers in the Grid for resource discovery. In this paper we use the ClasAds [10] in a completely distributed resource discovery service. We extend the P2P approach for range queries as described in [13] to provide distributed resource discovery in Grid scenario.

Iamnitchi et al. [6] discuss decentralized resource dis-

* This research was funded in parts by NSF Grants IIS 02-20152, EIA 00-80134, and INT 00-95527.

[†] Now at Google Inc.

covery for grids. They use a Gnutella [5] like approach and search for resources by guided flooding in an unstructured overlay. Our solution for resource discovery is based on a P2P Distributed Hash Table implementation as described in Content Addressable Network(CAN) [12]. CAN builds an overlay network with a logical structure of a d dimensional cartesian space. Each peer is responsible for a portion of the space and needs to maintain information about its neighboring peers (on average $O(2d)$) in the logical space. The searches are guided by the logical structure in the overlay and the average number of hops in searches are $O(dn^{1/d})$.

The rest of the paper is organized as follows. Section 2 presents the basic design of the resource discovery service. Section 3 discusses some issues involved in the implementation of the service. Section 4 presents experimental results and finally Section 5 presents the conclusions.

2. Distributed Resource Discovery

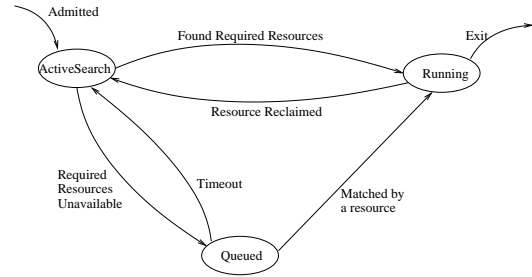
We present the design of a P2P resource discovery mechanism that allows:

- the jobs to actively search for compatible workstations to run.
- the workstations to actively search for jobs that can be run on them.
- the jobs and workstations to get queued in the absence of a match, and be discovered by an active search.

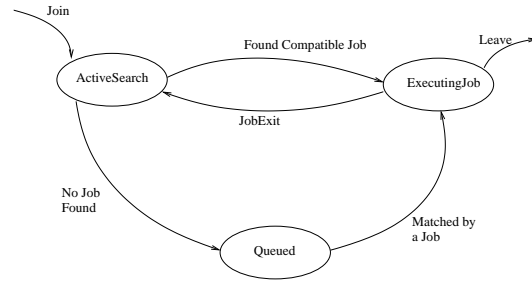
2.1. Overview of the Service

Workstations can dynamically join or leave the system and their availability may change based on the local workload at the machines. Application processes submitted to the system for execution are referred to as *jobs*. Our distributed resource discovery solution allows jobs to search for workstations matching their requirements, as well as the machines to look for jobs that can be executed on them. Figure 1 describes the states that the jobs and machines go through, from the perspective of the resource discovery mechanism. Figure 1(a) shows the states of the jobs in the system. When a job is submitted, it actively searches (ActiveSearch) for machines that match its requirements. If the job finds desired machines, it can start execution (Running). If the job could not find needed machines, then it gets queued (Queued) in the system for the amount of specified time interval. During this interval a job can be discovered by an available machine. Otherwise, after the specified time interval, the job again starts an active search for the machines.

The workstations participating in the computing system can also actively search (ActiveSearch) for compatible jobs that can run on the workstation (See Figure 1(b)). If a compatible job is found during the search, the workstation starts



(a) Job states



(b) Machine states

Figure 1. States from the perspective of Resource Discovery

executing the job and its state is Executing. When the job is done, the machine can again search for another job. If no compatible job is found during ActiveSearch, the workstation is Queued in the system, and it can be discovered by an active job searching for resources. When the workstation leaves the system, the job running on it will need to be checkpointed [1]. This job needs to search for an available machine that can continue the execution.

When a match between a job and a workstation is found, both the job and the workstation are notified of the match. The job and the workstation can negotiate further details at this point and ensure that all criteria are met and when the match is final, the machine and the job can remove their entries or queries from the system.

2.2. Advertisements

The workstations available in the systems can be classified according to their computational and storage capabilities and their network connectivity. Raman et al. [10] introduced the notion of classified ads for resource discovery. We use a simplified form of classified ads for our resource discovery service. The workstations are advertised

with the following attributes: CPU_Name, CPU_MHz, OS, Memory_MB, Storage_GB, Network_Kbps, LoadAvg, TTL. The attribute OS describes the type of operating system running on the machine as well as its version. Network_Kbps is the available network bandwidth. LoadAvg is the estimated load average on the workstation. TTL in seconds represents the time interval after which this advertisement expires. Therefore, the workstation periodically sends out ads with a TTL and updated estimates of load and network bandwidth. An example of a workstation ad is given below:

Workstation Ad

CPU_Name = Intel
Pentium 4
CPU_MHz = 2650
OS = Linux 2.4.20
Memory_MB = 1024
Storage_GB = 76
Network_Kbps = 700
LoadAvg = 0.01
TTL = 120

Job Ad

CPU_Name = Intel
CPU_MHz >= 1000
OS = Linux
Memory_MB >= 512

Jobs may have specific requirements of the computational capabilities of the available machines that they can use for execution purposes. A job searching for machines can specify an ad with acceptable ranges of values for various attributes characterizing the desired machines. An example of such an ad specification by a job is shown above, where the job is looking for machines that have Intel processors with CPU clock speeds higher than 1GHz running Linux operating system and have at least 512MB of available memory.

2.3. Advertisement Installation

The machines participating in the resource discovery service form a P2P based Distributed Hash Table as described in CAN [12]. CAN was designed for exact match queries. As the above example illustrates, job advertisements are specified both in terms of exact match predicates as well as range predicates, e.g., CPU_MHz ≥ 1000. Hence current P2P systems are not expressively powerful enough for such advertisements. In [13], Sahin et al. extended the exact-match query CAN system to support range queries. In this paper, we extend this approach to support diverse workstation and job advertisements and to efficiently match them with each other.

The resource discovery service has a logical cartesian space with 2n dimensions where n is the number of attributes describing a workstation. Each attribute corresponds to 2 dimensions. The start value of a range over an attribute is mapped to the odd dimension for the attribute and the end value of the range is mapped to the

even dimension of the attribute. Figure 2 shows an example of the 2d-space corresponding to a single attribute Memory_MB. Initially the system starts with single peer that owns the complete logical space. As new peers join the system, they contact an existing peer which divides its partition and hands over one portion to the new peer. Therefore, the space is partitioned among the peers participating in the service.

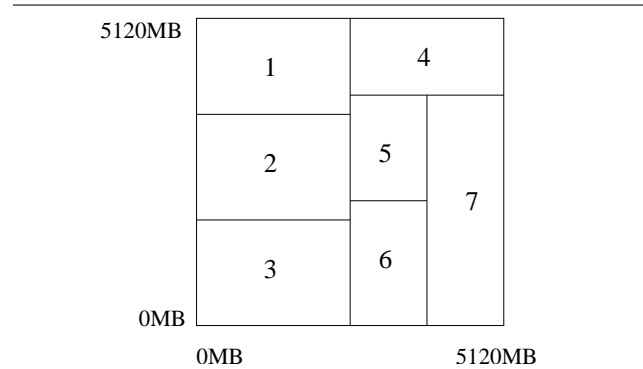


Figure 2. Example logical space corresponding to a single attribute, Memory_MB

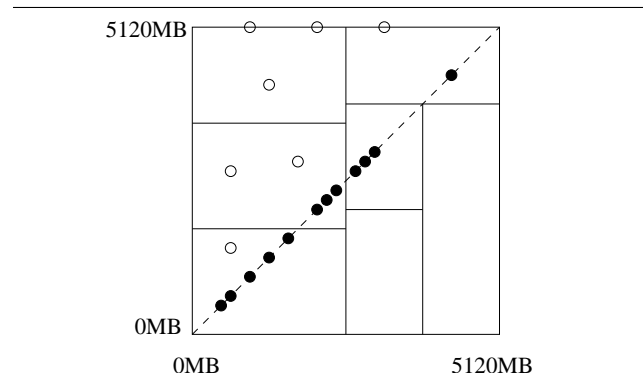


Figure 3. Installation of Ads of workstations and jobs

Job ads and workstation ads are mapped to points in this space based on the attribute values specified in their ads. A workstation ad $W = \langle v_1, v_2, \dots, v_n \rangle$ is mapped to the 2n-dimensional point $\langle v_1, v_1, v_2, v_2, \dots, v_n, v_n \rangle$. Notice that all workstation ads get mapped to the points on the diagonal hyperplane of the 2n dimensional cartesian space since they do not contain ranges. For example in Figure 3 the workstation ads are shown as filled dots on the diagonal. The workstation ads are installed at the peer whose zone contains the

point corresponding to the ads. The Job ads specify ranges of acceptable values for various attributes and they are represented as points in the same $2n$ dimensional space. A job ad $v_1 \leq A_1 \leq v_2, v_3 \leq A_2 \leq v_4, \dots, v_{2n-1} \leq A_n \leq v_{2n}$ is mapped to the point $\langle v_1, v_2, v_3, v_4, \dots, v_{2n-1}, v_{2n} \rangle$ in the $2n$ -dimensional space. Since the coordinates of the points corresponding to the job ads are determined by the ranges over the attributes, the odd dimensional values in the point are always less than or equal to the corresponding even dimensional values. Therefore, the job ads are mapped to the points in the upper left region of the cartesian space as shown in Figure 3 by circles. If the ad specifies an exact value for an attribute then the range over that attribute is degenerate and the start and end of the range are equal to the specified value. If the ad specifies an open ended range, i.e., only a lower limit on an attribute, then the range is bounded by the upper boundary of the domain of the attribute. For example the predicate `Memory_MB \geq 1024MB` will be converted to the point $(1024, 5120)$ in the logical space represented in Figure 3. An ad is installed at the peer which maintains the partition of the coordinate space that contains the point corresponding to the ad.

2.4. Matching Ads

Given a job advertisement, we need to identify all peers with qualifying workstation ads. Figure 4 shows the region of the logical cartesian space that can contain any workstation ads that can match a Job J's requirements. Job J is requesting workstations with available memory greater than 1024MB and less than 4096MB. All workstations in this range are represented by points on the diagonal between the points $(1024, 1024)$ and $(4096, 4096)$. Therefore the workstations matching a job's requirement are in the intersection of the diagonal hyperplane with the shaded region. This is the case because for any point in the intersection region, the attribute values are contained in the acceptable ranges of the job's requirements. When a job needs to actively search for machines, it can start the search at the peer that contains the job's ad point and forward the search message towards the peers intersecting with the lower right region of the point. The search message is forwarded via neighboring peers until it reaches the peers containing the diagonal region which can search their local store for workstation ads that match the search criteria. At this point, both the matching workstations and the job can be informed of the match, after which they can communicate with each other for claiming.

Alternatively, given an available workstation, we need to identify all peers with qualifying job ads. When a workstation is actively looking for jobs that can run on it, it routes the search message towards the point corresponding to its ad using the routing procedure described in CAN [12]. After reaching the ad point, the search needs to proceed in the up-

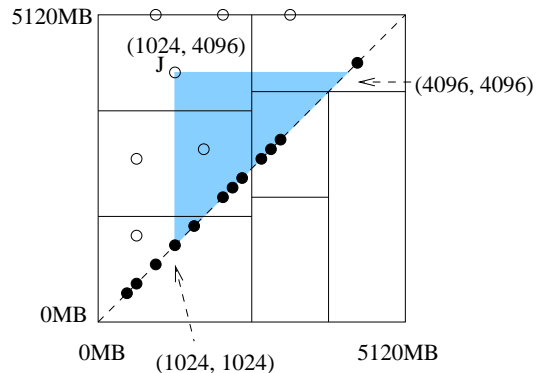


Figure 4. Job J can find needed machines in the shaded region.

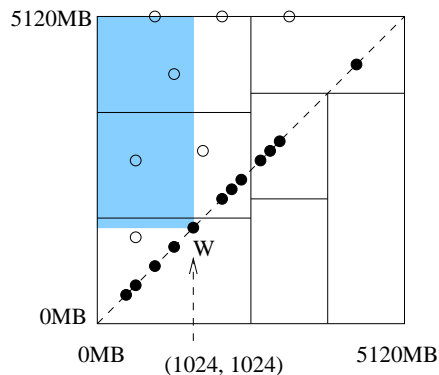


Figure 5. Workstation W needs to search the shaded region in upper left for matching jobs.

per left region of the workstation ad point as shown in Figure 5. In this example, workstation W has 1024Mb of memory, hence, it qualifies for any job whose range specification for memory requirements can be satisfied by 1024MB. This corresponds to the upper left region of point W, which is shaded in Figure 5. The search message is forwarded to the peers in the upper left region via neighboring peers. Any peer in the path of the search checks for any installed ads for jobs that match with the workstation ad. The matching jobs are informed of the match and they can communicate with the workstation to claim the resource.

3. Load Balancing

The process of discovery is dependent upon the forwarding mechanism of the P2P system to efficiently forward a job or workstation advertisement to all peers that can potentially contain a matching advertisement. The implemen-

tation of such a P2P system has to ensure a good balance of load among the participating peers in order to provide scalability. A centralized matching scheme can get overloaded in the presence of a large number of job and workstation advertisements. The P2P system should allow new joining peers to contact loaded peers and share the workload with them, thus preventing any single participant from becoming the bottleneck.

We now briefly describe the load balancing schemes used by our design. Each peer maintains a *load threshold* L_t , which determines the maximum sustainable load that the peer can handle. The peers choose individual values of L_t based on their computational and network capabilities. In addition, each peer maintains a running *load average* L_a , which is the ratio of its current load to its load threshold L_t . Thus the load average is 0 if the peer has no load and 1 if the load has reached the threshold.

The peers communicate their load average to their neighbors at regular intervals. Each peer maintains the load average information about its neighboring peers in the overlay along with other routing information. When a new peer joins the system, it sends a request to an existing peer in the system, the request is forwarded to the most loaded neighbor. The peers keep forwarding the request to the most heavily loaded neighbor until it reaches a peer which is more heavily loaded than any of its neighbors. This peer is referred to as a local hill. The peer at the local hill decides to either split its zone or replicate it based on its load conditions. If the local hill peer is overloaded due to storage constraints, it can split its zone and hand over one portion to the new peer. On the other hand, if the local hill peer is overloaded due to network communication constraints, it can replicate its zone at the new peer. The new peer's zone in this case becomes a replica and can share the messages coming to this zone, thus relieving the local hill peer of communication load.

An interesting observation is that job advertisements will usually specify only the lower bounds on the types of resources that will suit their requirements. This means that the range specifications in the job advertisements will be open ended. For example, a job looking for machines with at least 512MB of memory will result in an ad with a range $512 \leq \text{Memory_MB}$. In such cases, the range is bounded by the maximum memory in the schema definition of the space. The job advertisements with open ranges will get mapped to points on the boundaries of the logical space, because the end of the range is the boundary of the space.

4. Experimental Results

We have conducted preliminary experimental studies of the behavior of this system. We developed a simulator of the system in C++. The experiments were designed to test

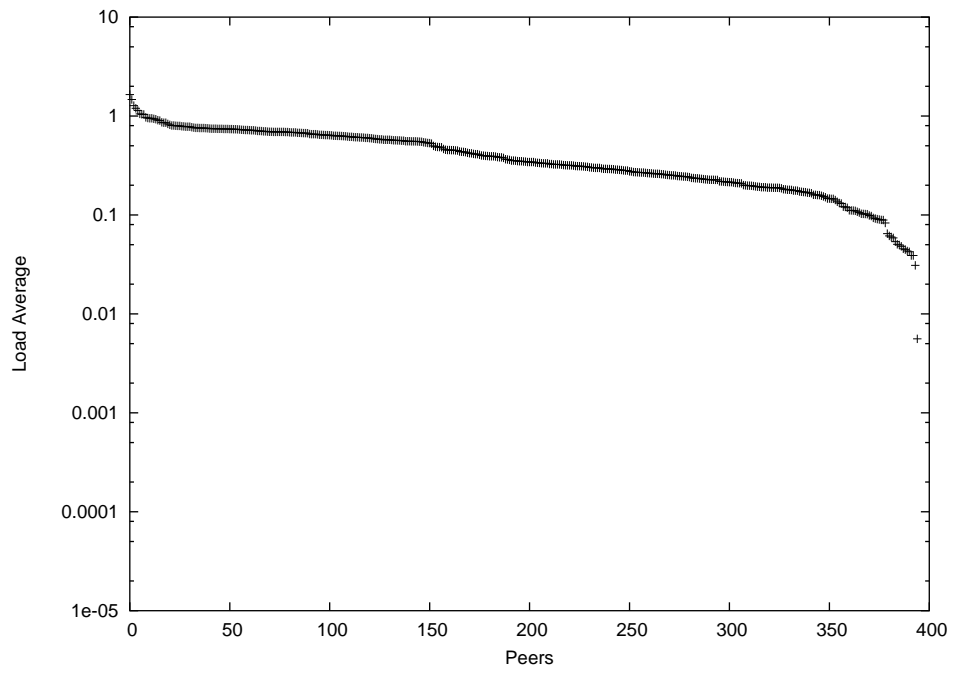
the properties of load distribution among the participating peers, because the distribution of load is crucial for the scalability of the system.

The workstation advertisements for the simulations were generated using template specifications of some machines in our lab. These included Intel Pentium, AMD Athlon, and SUN SPARC machines. We used 5 templates, which were characterized with the above CPU names and the clock speeds, memory size and other characteristics of these machines. The ads were generated by picking a template at random and then using a Gaussian distribution for each of the attributes `Memory_MB`, `Storage_GB`, and `Network_Kbps`. The load for each workstation ad was chosen uniformly at random. The OS was again used from templates of the machines and was one of `Linux-2.4.20`, `Solaris-9`, and `Windows-XP`.

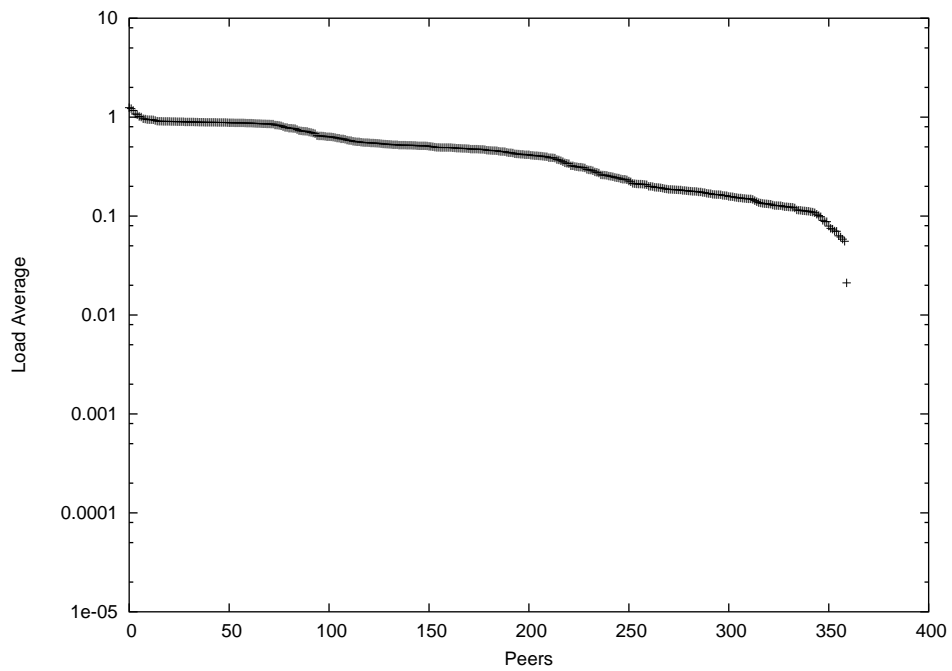
The job advertisements were generated using five template specifications. Two of these templates search for workstations of specific architectures and operating systems with desired CPU clock speed. These templates represent jobs that can only run on specific machine types. Another two templates are designed to search for workstations belonging to any architecture but with restrictions on the minimum clock speeds and minimum available memory and load average of the workstations. These two templates represent jobs that are portable and are looking for machines with high computational capabilities. The fifth template is designed to search for all machines that have a restriction on the minimum storage space and network bandwidth. This template represents jobs that are data intensive and need to transfer and store a lot of data.

We used two event sets for the simulations. In the first set, all the workstations ads are inserted into the system first, and then are followed by the job ads. This is a completely synthetic scenario, but it was used to isolate any effects due to the two different types of ads. This set has a total of 19876 workstation ads followed by 25038 job ads. In the second set, the job ads and the workstation ads were inserted concurrently. This is a realistic scenario for the application. This set has 25917 workstation ads and 14974 job ads.

The results of the experiments are shown in Figure 6 where the number of peers in the system is 400. The y-axis in the graphs represents the load average on the peers participating in the system. Recall that the load average of a peer is the ratio of its current load to its load threshold, hence, 1 refers to a peer that has load identical to its load threshold. The x-axis is the rank of the peer in decreasing order of its load after the simulation ended, and this does not reflect the order in which the peers join the system. From the graphs in Figures 6(a) and 6(b), we can see that the load is



(a) Workstation Ads preinstalled



(b) Concurrent Ads

Figure 6. Load distribution among participating peers

fairly distributed among the participating peers. Almost all the peers have a load average within their specified threshold. There are a small number of peers which have a load average higher than 1, which means that they have a load higher than they are willing. But this is due to the fact that the P2P system is a dynamic application which runs over a time much longer than the simulation itself. The results only represent a snapshot of the system at the time the simulation ended. If the system keeps running longer, the churn (joining and leaving) of peers will cause the loaded peers to share their load with new peers. The fact that there is a very small number of peers that have high load at the time the snapshot of the system is taken shows that the load distribution is working well.

5. Conclusions

The growing scale of shared resource communities like the Grid dictates that the resource discovery services in such environments should scale well. Centralized approaches to resource discovery have limited scalability. In this paper, we have presented a design of a distributed resource discovery service based on P2P design. Our preliminary experiments have shown that the design can effectively use the peers in the system with good load distribution among the peers. In the future, we will explore the scalability of this design to larger systems with thousands of peers.

References

- [1] J. Basney, M. Livny, and P. Mazzanti. Harnessing the capacity of computational grids for high energy physics. In *Proceedings of the International Conference on Computing in High Energy and Nuclear Physics (CHEP 2000)*.
- [2] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organization. *The International Journal of Supercomputer Applications*, 15(3):200–222, 2001.
- [3] J. Frey, T. Tanenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, 2002.
- [4] Globus. <http://www.globus.org/>.
- [5] Gnutella. <http://gnutella.wego.com/>.
- [6] A. Iamnitchi and I. Foster. On fully decentralized resource discovery in grid environments. In *International workshop on Grid Computing (GRID)*, pages 51–62, 2000.
- [7] KaZaA. <http://www.kazaa.com/>.
- [8] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems (ICDCS)*, pages 104–111, June 1988.
- [9] D. S. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou. Process migration. *ACM Comput. Surv.*, 32(3):241–299, 2000.
- [10] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the 7th IEEE High Performance Distributed Computing (HPDC)*, pages 140–147, July 1998.
- [11] R. Raman, M. Livny, and M. Solomon. Policy driven heterogeneous resource co-allocation with Gangmatching. In *Proceedings of the 12th IEEE High Performance Distributed Computing (HPDC)*, pages 80–89, June 2003.
- [12] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of the 2001 ACM SIGCOMM*, pages 161–172. ACM Press.
- [13] O. D. Sahin, A. Gupta, D. Agrawal, and A. El Abbadi. A peer-to-peer framework for caching range queries. In *Proceedings of the 20th International Conference on Data Engineering (ICDE)*, pages 165–176, 2004.
- [14] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, pages 149–160. ACM Press, 2001.
- [15] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, 2004.