Creating a Robust Desktop Grid using Peer-to-Peer Services *

Jik-Soo Kim[1], Beomseok Nam[1], Michael Marsh[1], Peter Keleher[1], Bobby Bhattacharjee[1],
Derek Richardson[2], Dennis Wellnitz[2] and Alan Sussman[2]
[1]UMIACS and Department of Computer Science
[2]Department of Astronomy
University of Maryland, College Park, MD 20742
[1]{jiksoo,bsnam,mmarsh,keleher,bobby,als}@cs.umd.edu
[2]{dcr,wellnitz}@astro.umd.edu

Abstract

The goal of the work described in this paper is to design and build a scalable infrastructure for executing grid applications on a widely distributed set of resources. Such grid infrastructure must be decentralized, robust, highly available, and scalable, while efficiently mapping application instances to available resources in the system. However, current desktop grid computing platforms are typically based on a client-server architecture, which has inherent shortcomings with respect to robustness, reliability and scalability. Fortunately, these problems can be addressed through the capabilities promised by new techniques and approaches in Peer-to-Peer (P2P) systems. By employing P2P services, our system allows users to submit jobs to be run in the system and to run jobs submitted by other users on any resources available in the system, essentially allowing a group of users to form an ad-hoc set of shared resources. The initial target application areas for the desktop grid system are in astronomy and space science simulation and data analysis.

1 Introduction

The recent growth of the Internet and the increasing CPU power of personal computers and workstations enables *desktop grid* computing to achieve tremendous computing power with low cost, through opportunistic sharing of resources [1, 2, 6]. Existing architectures for desktop grid computing are typically based on a client-server model, where a trusted server supplies jobs to a set of client machines distributed across the Internet. Robustness and reliability are guaranteed by the server maintaining the status of all outstanding jobs running on potentially unreliable clients, so that jobs assigned to clients can be re-run if a client does not return a result in a time period determined by the computational complexity of the job. The server must therefore be reliable, otherwise the status of outstanding jobs could be lost. The server typically stores the state of jobs in a (relational) database, which provides some level of reliability. However, this centralized client-server architecture is vulnerable to a single point of failure. No new jobs can be assigned to a client whenever the server becomes unavailable either due to server failure or network partition, which results in inherent shortcomings with respect to robustness, reliability and scalability.

Our goal is to design and build a scalable infrastructure for executing grid applications on a widely distributed set of resources. Such infrastructure must be *decentralized*, *robust*, *highly available*, and *scalable*, while efficiently *mapping* application instances to available resources throughout the system (called *matchmaking*). Fortunately, these are precisely the characteristics promised by new techniques and approaches in Peer-to-Peer (P2P) systems. Using P2P services can provide a robust, reliable, and scalable job submission and execution system that is able to efficiently utilize widely distributed available computational resources. Such a confluence of P2P and distributed computing is a natural step in the progression of grid computing, and has indeed been described as inevitable [5, 7, 10, 13].

Applications that are suited for our proposed system have both large computational requirements and relatively low I/O requirements. With our astronomy collaborators at the University of Maryland, we have identified multiple problem areas with these characteristics, mainly related to physical simulations and data analysis, including *finding habitable planets* through N-body simulations, *formation of asteroid binaries* through gravity simulations and analysis and modeling of data from the NASA *Deep Impact mission*.

*This research was supported by the National Science Foundation under Grants #CNS-0509266 and #CNS-0615072 and NASA under Grant #NNG06GE75G.

Additional astronomy applications may be explored in the later stages of our system development. While we are using the astronomy applications as the initial set for testing the system, applications from many other scientific and engineering disciplines, among others, can make use of the system, as evidenced by the widespread use of computational resources managed by Condor [14] and BOINC-based systems [1].

The rest of paper is structured as follows. Section 2 describes our overall system architecture for executing jobs using a P2P overlay network. Section 3 discusses efficient algorithms for matching jobs to resources, while Section 4 presents related work. We conclude in Section 5.

2 System Architecture

We describe a system composed from a relatively loosely coupled set of distributed, cooperating users (peers). Our goal is to use scalable P2P services to allow users to submit jobs to be run in the system and to run jobs submitted by other users on any resources available in the system, essentially allowing a group of users to form an ad-hoc set of shared resources. The overall system, from the point of view of a user, can be thought of as a combination of a centralized, Condor-like grid system for submitting and running arbitrary jobs [14], and a system such as BOINC [1] or SETI@Home [2] for farming out jobs from a server to be run on a (potentially very large) collection of machines in a completely distributed environment. However, to execute jobs in this decentralized and distributed environment (which are main characteristics of P2P system) we have to address several issues as follows:

1. *Job submission* - How can we submit a job into the P2P network?
2. *Matchmaking* - How can we find a resource that *meets* the minimum resource requirements of a job without any centralized control and information about the system for better scalability?
3. *Load balance* - How can we distribute the load (jobs) across the nodes in the system?
4. *Secure job execution* - Compute hosts should be protected from malicious jobs.
5. *Resilience to failures* - The overall system must be resilient to failures of individual resources.

For all that follows, we assume an underlying *Distributed Hash Table* (DHT) infrastructure [17, 18, 19, 21]. DHTs use computationally secure hashes to map arbitrary identifiers to random nodes in a system. This randomized mapping allows DHTs to present a simple insertion and lookup API that is highly robust, scalable, and efficient. A system can build upon these basic services to allow users to place idle computational resources into a general pool and draw upon the resources provided by others when needed. We insert both nodes and jobs into a single DHT, performing matchmaking by mapping a job to a node via the insertion process, and then relying on that node to find candidates that are able and willing to execute the job.

A *job* in our system is the data and associated profile that describes a computation to be performed. A job profile contains several characteristics about the job, such as the client that submitted it, its minimum resource requirements, the location of input data, etc. All jobs have modest I/O requirements, with individual input data sets for our initial target applications typically on the order of a few 100 KB or less, with correspondingly small output datasets. However, the jobs for each problem are computationally intensive, since simulation runs consist of advancing physical variables forward in time by solving a set of coupled differential equations, and data analysis runs perform complex operations on the data. Finally, the jobs in the system are *independent*, which implies that no communication is needed between them. This is a typical scenario in a desktop grid computing environment, enabling many independent users to submit their jobs to a collection of node resources in the system.

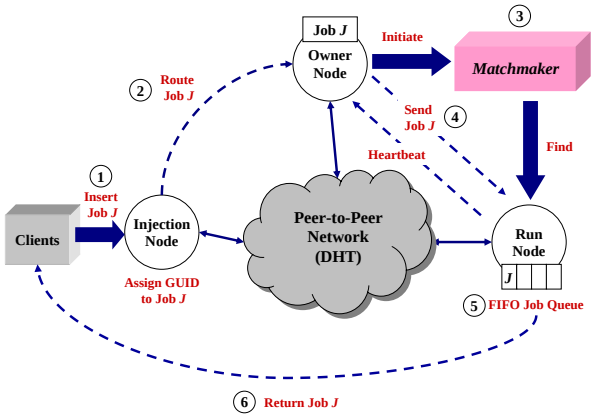


Figure 1. Overall System Architecture

Figure 1 shows the overall system architecture and flow of job insertion and execution in the P2P network. The steps of job execution are as follows:

1. A client inserts a job into a node in the system (the *injection node*). The DHT provides an external mechanism that can find an existing node in the system [17, 19].
2. The injection node assigns a *Globally Unique IDen-*

tifier (GUID) to the job by using its underlying hash function and *routes* the job to the *owner node*.

3. The owner node initiates a matchmaking mechanism to find a *run node* capable of running the job.
4. Once the matchmaking mechanism finds a run node for the job, the owner node sends the job to the run node.
5. The job is inserted into the job queue of the run node, which processes jobs in FIFO order. While processing the jobs, the run node periodically sends *heartbeat* messages to the owner node.
6. When the job is finished, the run node returns the results to the client.

An owner node is responsible for monitoring the execution of the job and ensuring that its results are returned to the client. Whenever a new job is assigned to an owner node, the owner node attempts to find an appropriate node for running the job (run node) through the matchmaking mechanism. Matchmaking is the process of matching jobs with physical resources, and consists of finding an appropriate node for running a job based on the constraints in the job profile and the current (distributed) state of the nodes in the system. The job profile can include several requirements for running the job, such as required CPU speed, amount of memory, supported operating system type(s), etc. Therefore, in the matchmaking process the first criterion in finding a match is whether the job constraints can be met. After finding one or more nodes that satisfy the job constraints, the matchmaking algorithm can consider balancing load across multiple candidates. However, as overall system scales to large configurations and heavy workloads, it becomes a challenge to efficiently match jobs having different resource requirements with available heterogeneous computational resources, to provide good load balancing, and to obtain high system throughput and low job turnaround times, all without any centralized control or information about the system [12].

Once an appropriate run node is found, the new job is inserted into the job queue of the run node. Each run node processes jobs in its job queue in FIFO order and only processes one job at a time. Until a job is completed and its results are returned, the run node periodically sends a heartbeat message to the owner node, which can relay the message to the client that initiated the job. This heartbeat message informs the owner node about the status of the running job and also indicates that the run node is still alive. The run node must generate heartbeat messages for every job in its job queue, including jobs that are not yet running. This soft-state heartbeat message plays an important role in failure recovery during the processing of jobs in our system. By employing the owner node and run node pair, our system can provide a robust environment for processing jobs, as the job profile is replicated both on the owner and run nodes to enable reconstruction of job information in case of failures. If either the owner or run nodes fails, the other node will detect the failure and initiate a recovery mechanism to make progress in the job execution. If both the owner and run node fail before the recovery protocol completes, the client must resubmit the job. To communicate via the heartbeat message, for efficiency we employ a direct connection between the run node and the owner node, for example by a socket connection, rather than using the P2P network routing mechanism.

Besides dealing with recovery from failures, the run node must also be able to ensure secure execution of each job in its job queue, to prevent jobs from adversely affecting the state of a node it is running on, and vice versa. We discuss node and job security more in Section 5. After successful completion of the job, the result can be returned to the client as either a pointer to the result (another GUID) or as the result itself.

As the first step in our concrete system design and implementation, we have concentrated on developing matchmaking algorithms for a decentralized and heterogeneous environment. We next describe the current state of development of our matchmaking algorithms, and provide some preliminary results obtained via simulations.

3 Matchmaking Algorithms

In this section, we briefly describe two approaches for matching incoming jobs to available system resources that we have developed: the *Rendezvous Node Tree*, and *CAN-based resource matching*.

3.1 The Rendezvous Node Tree

The Rendezvous Node Tree (RN-Tree or RNT) uses a distributed data structure built on top of an underlying Chord DHT [19]. Specifically, the RN-Tree copes with dynamic load balance issues by performing a limited random walk after the initial mapping to an owner node, and performs matchmaking by passing information describing the *maximal amount of each resource available* up and down the tree.

An RN-Tree contains all participating nodes in the desktop grid. Each node determines its parent node based only on local information, which enables building the tree in a completely decentralized manner. Due to the uniform distribution of GUIDs of the nodes in the system, the overall height of the RN-Tree is likely to be $O(\log N)$ where N is the total number of live nodes in the system (for details see [11]). Once the parent-child relationship in the RN-Tree is determined, each node periodically sends local subtree resource information (for the subtree rooted by that node) to

its parent node, and this information is *aggregated* at each level of the RN-Tree (*hierarchical* aggregation).

Jobs are injected into the system by mapping a job to a randomly chosen node that becomes the job's owner, which achieves a good initial load balancing by spreading the jobs across the system. The owner initiates a search for a node on which to run the job. The search first proceeds through the subtree rooted at the owner, only searching up the tree into subtrees rooted at the ancestors of the owner if the subtree does not contain any satisfactory candidates. The search is pruned using the maximal resource information carried by the RN-Tree. Rather than stopping at the first candidate capable of executing a given job, the search proceeds until at least k capable nodes are found for better load balancing (*extended search*). More details about the RN-Tree can be found in [11, 12]

3.2 Content-Addressable Network

A Content-Addressable Network (CAN) [17] is a DHT that maps GUIDs of nodes and data to points in a d-dimensional space, so that each node divides up the CAN space into rectangular *zones* and maintains *neighbor* information. Based on this basic CAN, we can formulate the matchmaking problem as a routing problem in a CAN space. By treating each *resource type* as a distinct dimension, nodes and jobs can be mapped into the CAN space by using their capabilities or requirements for each resource type, respectively, to determine their coordinates. Then the matchmaking process becomes somewhat straightforward since we can search for *the closest node whose coordinates in all dimensions meet or exceed the job's requirements*.

A job is inserted into the system by using its requirements as coordinates and defining the owner of the resulting zone as the owner of the job. The owner creates a list of candidate run nodes, and chooses the (approximately) least loaded among them based on load information periodically exchanged between neighboring nodes. The candidate nodes are drawn from the owners of neighboring zones, such that each candidate is at least as capable as the original owner in all dimensions (capabilities), but more capable in at least one dimension.

The basic CAN procedure works in all cases, but may cause serious load imbalance when many nodes have similar, or even identical, resource capabilities. Since the coordinates of a node are defined by its resource capabilities, identical nodes are mapped to the same place in the CAN volume. The best way to distribute ownership of a zone across multiple such nodes is not immediately obvious. Conversely, many jobs might have very similar requirements. For example, many jobs will likely be inserted into the system with no resource requirements at all specified. In this case, all of the those jobs will be mapped to the single node that owns the zone containing the origin in the CAN space.

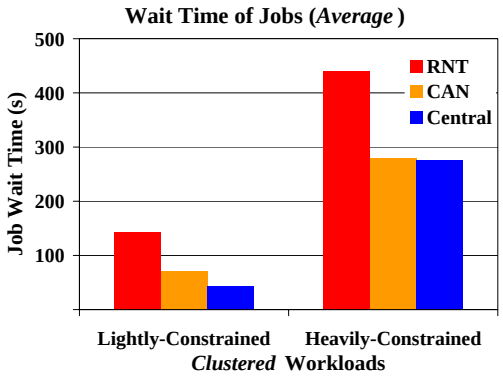
We address this problem by supplementing the "real" dimensions (those corresponding to node capabilities) with a *virtual dimension*. Coordinates in the virtual dimension are generated *uniformly at random*. Whenever a new node joins the system, a representative point for the new node is generated by combining the resource capabilities of the node and a randomly generated virtual dimension value. Therefore, even when multiple identical nodes join the system, they are mapped to distinct locations, and CAN zone splitting is straightforward. Similarly, when a new job is inserted into the system, the new job's coordinates become a combination of the job's requirements and a randomly assigned virtual dimension coordinate. In combination, the randomly assigned node and job coordinates act to break up clusters and spread load more evenly over nodes. More details about CAN matchmaking can be found in [12].

3.3 Initial Experimental Results

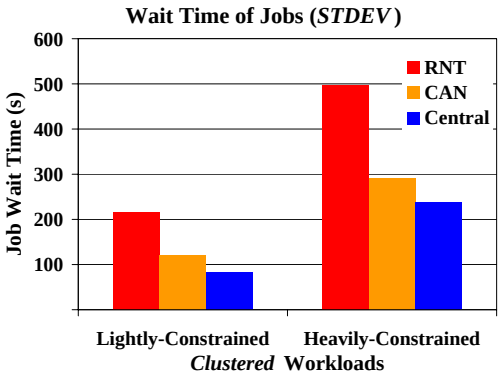
We have employed an event-driven simulator to investigate the basic behavior of a P2P network, namely creating and maintaining the network and performing lookups into the distributed hash table based on peer IDs.

The results for our matchmaking algorithms for different workload scenarios and under relatively heavy loads, with multiple clients submitting jobs over time at different average rates are shown in Figure 2. Our test workloads differ on two axes. Workloads are categorized as either *clustered* or *mixed*. The former divides all nodes and jobs into a small number of equivalence classes (in terms of resource capabilities and constraints, respectively), where all nodes or jobs in a given equivalence class are identical. The latter assigns node capabilities and job constraints randomly. Based on these concepts, the overall problem space for grid computing environments can be divided along two axes, measuring the degree to which the nodes and jobs are either clustered or mixed. Systems such as Condor [14] mainly target mixed jobs running on clustered nodes, while systems like BOINC [1] or SETI@Home [2] often deal with clustered jobs on mixed nodes. Our intent is to effectively support all four scenarios.

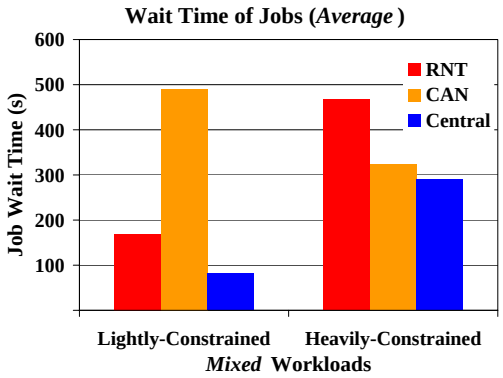
In the experiments presented here, workloads are also distinguished by whether the jobs are *lightly* or *heavily* constrained. For a given job, each type of resource has a fixed independent probability of being constrained: "lightly-constrained" jobs have an average of 1.2 constraints (out of the 3) and "heavily-constrained" jobs have an average of 2.4. As a job has more resource requirements (i.e., heavily-constrained workloads), it is likely to be harder to match the job to the available resources, since fewer nodes in the system can meet those multiple constraints. All of the test



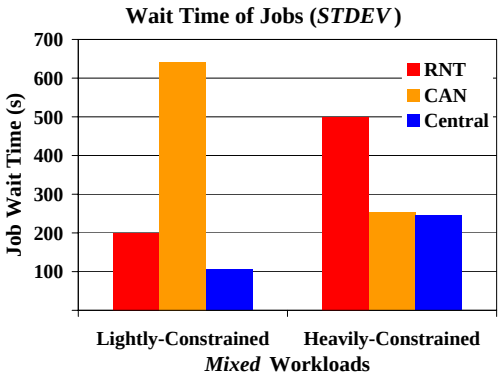
(a) Average Job Wait Time for Clustered Workloads



(b) STDEV of Job Wait Time for Clustered Workloads



(c) Average Job Wait Time for Mixed Workloads



(d) STDEV of Job Wait Time for Mixed Workloads

Figure 2. Job Wait Time for Clustered and Mixed Workloads

workloads consist of 1000 nodes and 10000 jobs, each of which has an average running time of about 200 seconds. The job arrival times are based on a Poisson distribution with an average inter-arrival rate of 0.1 seconds. To see how well the workload could be balanced, we also show results for a *centralized* scheme that uses knowledge of the status of all nodes and jobs. Such a scheme would be very expensive to implement in a decentralized P2P system, but serves as a target for achieving the best possible load balance from an online matchmaking algorithm.

Overall, we found that for most scenarios, the CAN-based matchmaking framework shows very competitive performance in terms of balancing loads, even compared to the centralized scheme, with low matchmaking cost (in results not shown, we have verified that both the RNT and CAN can find an appropriate run node for a job with a small number of hops through the P2P overlay network). However, we found that under some conditions the CAN-based algorithm works very poorly due to serious load imbalance, namely when jobs with few resource requirements are run on nodes with heterogeneous (mixed) resource capabilities (i.e., the lightly-constrained workloads in Figures 2(c) and 2(d)).

In ongoing work, we have improved the basic CAN-based matchmaking mechanism to address this problem by *pushing* jobs into underloaded regions of the CAN space based on *dynamic aggregated load information*. The basic concept is that when a new job is inserted into the system and routed to the owner node, the job is *pushed* into an underloaded region in the CAN space. To determine whether to initiate pushing of a job, a fixed amount of current system load information is propagated along each dimension in the CAN space. If the overall system is lightly loaded, the job can be pushed into the upper regions of the CAN space (far-

ther from the origin) and utilize the more capable nodes in the system. In preliminary experiments not shown here, we have verified that the modified CAN-based matchmaking mechanism dramatically improves the quality of load balancing compared to the basic scheme presented here, still with low matchmaking cost.

4 Related Work

Recently there have been several research efforts to combine P2P and grid computing techniques to improve the robustness, reliability and scalability of commonly available client-server based desktop grid infrastructure.

Research such as [4, 10] proposes a P2P architecture to locate and allocate resources in a grid environment by employing a *Time-To-Live* (TTL) mechanism. TTL-based mechanisms are relatively simple but effective ways to find a resource (that meets the job requirements) in a widely distributed environment without incurring too much overhead in the search. However, such mechanisms may fail to find a resource capable of running a given job, even though such a resource exists somewhere in the network.

Studies on encoding static or dynamic information about computational resources using a DHT hash function for resource discovery have also been conducted [5, 9, 16]. However, there can be a load balancing problem for these encoding techniques, since a small fraction of the nodes can contain a majority of the resource information whenever there are many nodes that have very similar (or identical) resource capabilities in the system.

The CCOF (Cluster Computing on the Fly) project [15, 22] has conducted a comprehensive study of generic searching methods in a highly dynamic P2P environment to locate idle computer cycles throughout the Internet. More recent work from the CCOF researchers, on a peer-based desktop grid system called WaveGrid, constructed a *timezone-aware* overlay network based on a Content-Addressable Network [17] to use idle night-time cycles geographically distributed across the globe [23]. However, the host availability model in that work is not based on the resource requirements of the jobs, and that work does not consider balancing load across the available system resources.

5 Conclusions and Future Work

In this paper, we have proposed an architecture that employs P2P services to allow users to submit jobs to be run in the system and to run jobs submitted by other users on any resources available in the system. Our experimental results obtained via simulations show that the system can reliably execute grid applications on a widely distributed set of resources with good load balancing and low matchmaking cost.

Sequential applications with large computing requirements and relatively low I/O requirements are the intended users of our proposed system. Application areas that fall into this class include bioinformatics applications such as DNA sequence comparison and protein folding, Monte Carlo and other physical simulations in various scientific disciplines, and more esoteric applications such as searching for extraterrestrial life (SETI@Home) [2]. However, unlike existing projects, the proposed system can be used by any participant (peer) to execute any set of jobs desired. It then becomes the responsibility of the system to utilize all available computational resources to execute all submitted jobs in a fair manner, allocating resources to requests from both users submitting large numbers of jobs at once (as in a parameter sweep for a physical simulation application) and from users with smaller resource requirements. We leave this fairness issue as part of our future work.

In our current formulation of the problem, there are no dependencies between jobs, but if computational scientists also use the system for data analysis of results, then the system will have to distinguish between job types (simulation vs. analysis) and perform the jobs in the correct order (analysis after simulation of a given problem), and make the output of a simulation job available as the input for the corresponding analysis job(s). We will investigate using existing software packages, such as Condor's DAGMan [20], for managing dependencies between jobs.

Compute nodes should be protected from malicious jobs through the use of existing technology, such as `chroot` jails, that prevent applications from either reading or writing any files outside of a prescribed set. We will constrain jobs to not be able to access the network, and all output produced is stored on the node executing the job until the job terminates. These policies will be enforced using standard process containment techniques [3, 8]. We will also employ generalized quotas to limit overall job resource usage (e.g., disk space), to minimize the effects of malicious or runaway jobs.

We are in the process of building a prototype system using CAN-based matchmaking, and will characterize its behavior on real workloads, via consultation with our application-area collaborators in astronomy and physics. In the future, we will measure and report on the behavior of our system for heterogeneous environments running real applications.

References

[1] D. P. Anderson, C. Christensen, and B. Allen. Designing a Runtime System for Volunteer Computing. In *Proceedings of 2006 IEEE/ACM SC06 Conference*, Nov. 2006.

[2] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: An Experiment in Public-

Resource Computing. *Communications of the ACM*, 45(11):56–61, Nov. 2002.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'01)*, 2003.

[4] A. R. Butt, X. Fang, Y. C. Hu, and S. Midkiff. Java, Peer-to-Peer, and Accountability: Building Blocks for Distributed Cycle Sharing. In *Proceedings of the 3rd Virtual Machines Research and Technology Syposium*, May 2004.

[5] A. S. Cheema, M. Muhammad, and I. Gupta. Peer-to-peer Discovery of Computational Resources for Grid Applications. In *Proceedings of the 6th IEEE/ACM International Conference on Grid Computing - GRID 2005*, Nov. 2005.

[6] A. Chien, B. Calder, S. Elbert, and K. Bhatia. Entropia: Architecture and Performance of an Enterprise Desktop Grid System. *Journal of Parallel and Distributed Computing*, 63(5):597–610, May 2003.

[7] I. Foster and A. Iamnitchi. On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*. Springer-Verlag, Feb. 2003.

[8] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual-machine based platform for trusted computing. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'01)*, 2003.

[9] R. Gupta, V. Sekhria, and A. Somani. CompuP2P: An Architecture for Internet Computing using Peer-to-Peer Networks. *IEEE Transactions on Parallel and Distributed Systems*, 17(11):1306–1320, Nov. 2006.

[10] A. Iamnitchi and I. Foster. A Peer-to-Peer Approach to Resource Location in Grid Environments. *Grid Resource Management, Kluwer Publishing*, 2003.

[11] J.-S. Kim, B. Bhattacharjee, P. J. Keleher, and A. Sussman. Matching Jobs to Resources in Distributed Desktop Grid Environments. Technical Report CS-TR-4791 and UMIACS-TR-2006-15, University of Maryland, Department of Computer Science and UMIACS, Apr. 2006.

[12] J.-S. Kim, B. Nam, P. Keleher, M. Marsh, B. Bhattacharjee, and A. Sussman. Resource Discovery Techniques in Distributed Desktop Grid Environments. In *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing - GRID 2006*. IEEE Computer Society Press, Sept. 2006.

[13] J. Ledlie, J. Schneidman, M. Seltzer, and J. Huth. Scooped, Again. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Lecture Notes in Computer Science. Springer-Verlag, Feb. 2003.

[14] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor – a hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems*. IEEE Computer Society Press, June 1988.

[15] V. Lo, D. Zhou, D. Zappala, Y. Lin, and S. Zhao. Cluster computing on the fly: P2P scheduling of idle cycles in the Internet. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS '04)*, Lecture Notes in Computer Science. Springer-Verlag, Feb. 2004.

[16] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Design and Implementation Tradeoffs for Wide-Area Resource Discovery. In *Proceedings of 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14)*, July 2005.

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *In Proceedings of the ACM SIGCOMM 2001 Technical Conference*, 2001.

[18] A. Rowstran and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, pages 329–350, Nov. 2001.

[19] 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 ACM SIGCOMM*, Aug. 2001.

[20] D. Thain, T. Tannenbaum, and M. Livny. Condor and the Grid. In F. Berman, A. J. Hey, and G. Fox, editors, *Grid Computing: Making The Global Infrastructure a Reality*, chapter 11, pages 299–336. John Wiley, 2003.

[21] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A Resilient Global-scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communications*, 22(1), Jan. 2004.

[22] D. Zhou and V. Lo. Cluster Computing on the Fly: Resource Discovery in a Cycle Sharing Peer-to-Peer System. In *Proceedings of the 4th International Workshop on Global and Peer-to-Peer Computing*, Apr. 2004.

[23] D. Zhou and V. Lo. WaveGrid: a Scalable Fast-turnaround Heterogeneous Peer-based Desktop Grid System. In *Proceedings of the 20th International Parallel & Distributed Processing Symposium*. IEEE Computer Society Press, Apr. 2006.