Cluster Computing on the Fly: Resource Discovery in a Cycle Sharing Peer-to-Peer System

Dayi Zhou and Virginia Lo
Department of Computer Science, 1202 University of Oregon, Eugene OR 97403-1202
dayizhou|lo@cs.uoregon.edu

Abstract—*Peer-to-peer computing, the harnessing of idle compute cycles throughout the Internet, offers exciting new research challenges in the converging domains of networking and distributed computing. Our system, Cluster Computing on the Fly, seeks to harvest cycles from ordinary users in an open access, non-institutional environment.*

We conduct a comprehensive study of generic searching methods in a highly dynamic peer-to-peer environment for locating idle cycles for workpile applications, which are heavy consumers of cycles. We compare four scalable search methods: expanding ring, advertisement-based, random walk and rendezvous point. We model a variety of workloads, simple scheduling strategies and stabilities of hosts. Our preliminary results show that under light workloads, rendezvous point performs best, while under heavy workloads, its performance falls below the other techniques. We expected rendezvous point to consistently outperform the other search techniques because of its inherent advantage in gathering knowledge about the idle cycles. However, in a peer-to-peer environment, which satisfies requests on-demand, large jobs may dominate, resulting in delays for scheduling smaller jobs.

I. Introduction

Peer-to-peer computing, the harnessing of idle compute cycles throughout the Internet, offers exciting new research challenges for peer-to-peer networks beyond current file sharing applications. Experience has shown that not only are idle cycles widely available throughout the Internet, but in addition, many users are willing to share cycles [7], [9], [3]. This creates a compelling opportunity for research in this exciting new juncture between the fields of networking and distributed computing.

The advantage of using idle cycles has not gone unnoticed. Load sharing tools for traditional distributed systems, such as Condor[17], pool idle cycles within one or more participating institutions. Some current research proposes to peer Condor flocks using a a peer-to-peer overlay [5]. Much research has focused on the convergence of P2P computing with Grid Computing[13], [16], [15], [10], [12]. Other current research is focused on building a framework with a barter economy for sharing cycles [11], [4].

A. CCOF: Cluster Computing on the Fly

Our system, Cluster Computing on the Fly [20], harvests idle cycles available from ordinary users located at the edges of the Internet, makes them accessible to the average citizen and does not require membership in any organization. Popular cooperative-computing projects on the Internet, such as SETI@home [7], the Stanford Folding Project [9], and BOINC [1] are asymetric in that users donate cycles to a project but do not receive cycles or services in return. In addition, these projects require donors of cycles to manually coordinate through a centralized web site. CCOF is more general, and supports automatic scheduling of a variety of applications in a fully distributed model under which any peer can be either a donor or a consumer of idle cycles or both. CCOF also assumes *long-term fairness*, that is everyone has fair access to the system, but does not require close monitoring of donated cycles versus consumed cycles.

Our research addresses the problem of peer-to-peer scheduling, which encompasses all of the activities involved with utilizing idle cycles from ordinary users in a distributed environment. We believe that peer-to-peer scheduling solutions must be driven by the characteristics and goals of the specific applications to be scheduled. We identify three important classes of problems that are particularly well-suited to capturing idle cycles in the Internet: workpile, tree-based search, and point-of-presence applications. *This paper is focused on resource discovery for workpile applications, which consume large amounts of CPU cycles under a master-slave model (embarassingly parallel).*

We propose a peer-to-peer cycle-sharing architecture for CCOF. In this architecture, hosts join a variety of community-based overlay networks, depending on how they would like to donate their idle cycles. Clients then form a compute cluster on the fly by discovering and scheduling set of machines from these overlays. The basic service offered by CCOF is best-effort in the sense that any host may preempt guest codes at any time. Hosts retain local control and can thus offer a range of quality of service options. The components of this architecture, as shown in Figure 1, include the following: Community-based overlay network management, application scheduler, local scheduler, and coordinated scheduling. *The emphasis of this study is on designing the resource discovery mechanism for the application scheduler.* The application scheduler represents the client to discover available hosts, select a subset of the hosts, negotiate access, export the application, and collect the results.

B. Resource Discovery in CCOF

Resource discovery is a unique challenge for an open cycle sharing system, because the set of participating hosts is potentially very large and dynamic.

With its open nature, CCOF will be of much larger scale and be much more dynamic than the traditional institutional-based distributed systems and GRID systems. In traditional

This work was supported in part by the National Science Foundation under grants ANI-9977524 and NCR-9714680.

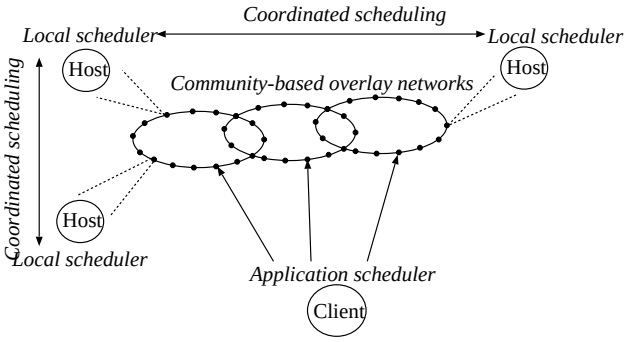

Fig. 1. CCOF Architecture

distributed systems, central servers function as matchmakers to match the request with the available resources or representatives for resource negotiation [17], [18], [8]. This central server approach is not scalable and cannot be adapted to the CCOF environment, as the central servers are performance bottlenecks and traffic hot spots. Also there is no clear incentive for a third party to provide such institutional support in this open peer-to-peer system.

The amount of available resources in CCOF is highly dynamic. Like any other peer-to-peer system, peers may join and leave the system at any time. Furthermore, the amount of available resources in CCOF may change much more quickly and dynamically over time than in traditional content sharing peer-to-peer systems. In the latter systems, once a file appears, it remains in the system as long as the owner of that file stays. Also the file can be replicated in strategic locations and linger a long time, even after the original owner leaves the system. However, the crucial resource of CPU cycles in CCOF changes over time and cannot be replicated. Users may be willing to allow others to download files, provided the bandwidth consumption is small. However, the same group of users may not be so willing to let others share CPU cycles when they are working on their own computer. The foreign jobs should leave that machine or sleep with the lowest priority when the owner of the machine reclaims it. The resource discovery strategy used by CCOF has to be adaptive to this ever-changing environment.

II. Related Work

The goal of resource discovery in CCOF is to find idle hosts in an ocean of machines. It is related to many cycle-sharing systems and peer-to-peer applications. Due to the page limitation, we only list several highly relevant research projects.

Recently, DHT overlays[21], [24], [23] have been proposed to aid efficient content location in a peer-to-peer system. However, the DHT method may not be applicable for locating resources in CCOF, since the dynamic changing resource information about available CPU cycles cannot be naturally hashed into the overlay. Furthermore, the semantics of search keys is not well-suited to attribute-based resource requests, though it is still possible to organize the peers using a DHT overlay for efficient routing.

The project Flock of Condors at Purdue[5] proposed to connect Condor pools using a Pastry [23] overlay. It used an advertisement-based approach, in which a Condor server sends out resource information for its site in a limited scope. On receiving the information, Condor servers of other sites cache and then select the candidate Condor flock when needed. The system is static and the paper does not present direct measurements of how this type of resource discovery performs. Also it does not compare their method with other types of search methods.

The research conducted by Iamnitchi proposed peering multiple sites in a Grid computing infrastructure[15], [14]. She conducted a study on how to satisfy resource discovery requests for a variety of types of resources. The emulation was performed in a static Grid environment in which peers join but do not leave the system. Different models of resource distribution and request distribution were used. The research compared random walk with a learning-based strategy and a best-neighbor strategy to make the request-forwarding decision, and analyzed the latency (in hops) to satisfy resource requests.

The SHARP project [11] proposed an architecture for secured resource sharing. It uses a link state style protocol to propagate resource information, and then nodes compute the route to the available resource. This method is not scalable in a large, highly dynamic environment with the heavy traffic of resource information propagating in the system. The emulation is on a static and small scale overlay network (Planet Lab), so it cannot demonstrate how the protocol works in a large scale peer-to-peer system.

We propose a comprehensive study of generic search methods in a highly dynamic peer-to-peer environment for locating idle cycles for workpile tasks. We compare four different search methods: expanding ring search, advertisement-based search, random walk and rendezvous point. While these search methods have found wide use for a spectrum of network applications, no work has shown how they compare in a dynamic resource sharing system, and whether they can be leveraged into a cycle sharing peer-to-peer system. *The question we want to answer is which of these search algorithms can be leveraged into CCOF and how the performance is influenced by the dynamic nature of the system.*

III. CCOF Resource Discovery Model

In this section, we describe the CCOF resource discovery model, which includes our model of the dynamic behavior of hosts; our profile based model that describes each host's idle cycles; and the search algorithms and scheduling strategies we study.

A. Dynamic Hosts

In CCOF, peers can leave and join at any time. When one peer joins the system, new idle cycles will be available to the whole system; when one peer leaves the system, the foreign task running on that peer will be stopped and that peer will inform the initiator of the task to reschedule or migrate the task.

In CCOF, peers can withdraw cycles when the user reclaims his machine, based on local policy. The peer cannot then be used to do computation, but it can still be used to relay messages for other peers.

In traditional institutional based load sharing schemes, migration is acceptable, because within one institution, the network latency between machines is low and the central server provides quick discovery of newly available resources; thus the migration cost is comparatively small. However, within CCOF, the peers are scattered throughout the Internet. Thus, there is no guarantee about the speed of locating new resources among the peers and transmission speeds may vary greatly. CCOF needs to be carefully designed to avoid unnecessary migration, due to lack of knowledge about the 'remote' host.

In an unreliable environment, peer machines may crash, and then the task dies. The application scheduler will detect the peer failure when it stops receiving heart-beat messages from that machine. The application scheduler then tries to restart the tasks on some other available host.

The results reported in this workshop paper report the impact of hosts' withdrawing cycles from the system due to changes in users' daily schedule. This is a notable phenomenon in systems for sharing idle cycles, while it is not so visible in other types of peer-to-peer systems, as we have described in the section I-A. Frequent changes in machine usage result in frequent resource discovery activities. Methods which are not scalable will then produce excess amounts of resource discovery message traffic.

B. Profile Based Model

We use a profile-based model to generate resource information about idle cycles on each host. The profile we use is based on the observation that people have daily routines for using their machines and that most machines are idle at night. For example, users may process email and browse the Internet in the morning; then their machines will be idle while they are in meetings or attending classes. Such usage patterns can be acquired manually by user input or automatically from a monitoring program running on the local host.

Currently, we only use profiles of available CPU cycles, however other resource information such as memory size and operating system information can be easily added into the profile as a separate attribute.

C. Workpile Applications

Workpile applications consume huge amounts of compute time under a master-slave model in which the master gives out code to many hosts, each host computes intensively and then returns the results back to the master node. The workpile application is 'embarrassingly parallel' in that there is no communication among slave nodes. In this paper, we use the term *jobs* to refer to the collection of tasks initiated by a client node, and we use the term *task* to refer to a unit of work exported from a client node to a host node.

D. Search Algorithms

We evaluate four scalable search methods and one non-scalable search method for comparison purposes.

- **Centralized Search.** For comparison purposes, we implement a centralized search algorithm, which we know is non-scalable. When a peer joins the system and it is willing to share the idle cycles, it reports its profile information to the central server. Clients send requests for cycles to the server. The central server then matches their needs with available idle hosts. In our simulation, the server always tries to find idle hosts near to the source of the task. We assume the central server has global knowledge of the overlay topology even though this is not achievable in practice. We use this central server scenario to provide an optimal (but unrealizable) cycle sharing environment for our performance comparisons.
- **Expanding Ring Search.** When a client peer needs cycles, it sends the request for cycles to its direct neighbors. On receiving the request, the neighbor compares its profile to the request. If it is currently busy or the block of idle time is less than the requested block of time, the neighbor turns down the request; otherwise the neighbor accepts the request by sending the client peer an ACK message. If the client determines there are not enough candidate hosts to satisfy the request, it then sends the request to peers one hop farther. This procedure repeats until the client peer finds enough candidates to start the computation or the search reaches the search scope limitation.
- **Random walk search.** When a client peer needs cycles, it sends the request to k random neighbors. On receiving the request, like the expanding ring search, the neighbor then tries to match the request with its current status. If it has enough time to complete the task, it then sends back an ACK message. The neighbor also forwards the request to its k random neighbors, until the request reaches the forwarding scope limitation.
- **Advertisement based search (Ads-based search).** When a peer joins the system, it sends out its profile information to neighbors in a limited scope. The neighbors then cache such profile information along with the node ID. When a client peer needs cycles, it consults the profiles it has cached locally, and selects a list of available candidates. Since other clients may try to use those hosts at the same time, the client then needs to directly contact each of the candidates to confirm their availability. If a host is not available at this time, the client then tries the next host in the list, until the list is exhausted or the request is satisfied. There are several possible selection schemes, such as choosing the nearest available hosts, choosing the hosts with longest available time or choosing hosts

with shortest matching available time block. Our simulation results show different selection schemes yield similar performance. The result we present in the simulation section uses the scheme of choosing the nearest available host.

- **Rendezvous Point Search.** This method uses a group of dynamically selected Rendezvous Points in the system for efficient query and information gathering. Hosts advertise their profiles to the nearest Rendezvous Point(s), and clients contact the nearest Rendezvous Point(s) to locate available hosts.

We assume an out-of-band Rendezvous Point placement method, which guarantees that Rendezvous Points are geographically scattered in the peer-to-peer system. We note that dynamic placement of Rendezvous Points such that the system is balanced and a sufficient number of Rendezvous Points is within short distance to every peer is still an open problem. When a peer is selected as a Rendezvous Point, it floods information about its new role within a limited scope. On receiving such a message, the peer adds the new Rendezvous Point into a local list and deposits its profile information on this new Rendezvous Point. When a peer joins the system, it acquires the list of Rendezvous Points from the nodes it contacts or acquires the information through some out-of-band scheme. When a client peer needs extra cycles, if it has not already cached information about nearby Rendezvous Points , it queries its neighbors until it accumulates a list of known Rendezvous Points. The peer then contacts Rendezvous Points on the list one by one and each Rendezvous Point then tries to match the request with candidate hosts. This procedure repeats until the request is satisfied or all the known Rendezvous Points have been queried.

E. Scheduling Strategies

After the application scheduler has discovered candidate hosts for the client job, it can choose hosts based on multiple criteria, such as trust value, performance ranking etc. In this simulation, we simplify the application scheduler and use only the availability of cycles as the criteria.

When a client fails to find enough resources to start the computation, the application scheduler can choose to give up or to try to reschedule the task at night, since hosts have maximal available time at night. This is similar to the notion of prime time v. non-prime time scheduling enforced by parallel job schedulers[19]. We also did simulations using exponential back-off scheduling. With that method, when the application scheduler fails to find enough resources, it retries after a random amount of time. If it fails, it can retry multiple times, each time with a doubled back-off time. The simulation results we present in this paper report the first two scheduling methods: no retry and retry at night (The performance of exponential back-off is comparable to retry at night).

After the client finds the idle hosts, in order to to avoid unnecessary competition and migration, it then reserves blocks of time on those hosts to start the computation. If a host withdraws cycles, the host migrates the task.

IV. Simulation

A. Simulation Configuration

Our CCOF resource discovery experiments are conducted using the *ns* simulator. A power-law topology of 4000 nodes is used as the overlay topology, as current studies have shown that peer-to-peer systems exhibit power-law properties [22], [2]. The average node degree is 4.09 and the network diameter is approximately 23.

During the simulation, peers in the system are divided into two non-overlapping groups. One group is the hosts, providing the idle cycles. The other group is the clients, needing extra cycles. The ratio of clients to hosts varies from 0.1 to 1.3. At light workloads, the hosts outnumber the clients; at heavy workloads, the clients outnumber the hosts.

We model the dynamic peer-to-peer environment by varying the probability that a given host will withdraw cycles from the system at any particular time of the day. The idle cycles on that host become unavailable to the system after that time; however the host still relays messages for the other peers.

We conduct a one-day simulation of CCOF. For each host, a 24-hour profile of idle time blocks is generated. The synthetic profile is generated in the following way: The machine is idle during the whole night (from 7pm to 7am); then for each time unit (one hour) during daytime, it is randomly decided whether the machine is idle or not. The probability of a machine being idle in one time unit is 0.3 in this simulation.

Clients submit random numbers of jobs into the system at random times during the day based on two different client request arrival distributions. Two probability distributions are used to model client request arrival patterns. With a uniform distribution, a client is equally likely to submit a job at any time of the day; with normal distribution, there is a peak load at noon.

The jobs are characterized by the number of processors needed and the length of time needed on each of the processors. The minimum requested time block is one hour, while maximum during the day is 4 hours and the maximum during the night is 6 hours. The number of processors ranges from one to a maximum of 10 % of all the peers in the system (400 in this simulation). The run time and number of processors are independently generated using exponentially decreasing functions (e.g. number of jobs vs. number of processors or number of jobs vs. runtime).

We varied the search parameters for all of the search algorithms until we found those values that yield reasonable job completion rates and when possible, acceptable message passing overhead. For expanding ring search, the search scope is 5 hops in the peer-to-peer overlay. For advertisement-based search, the scope of advertisement propagation is also 5 hops in the overlay. Peers may contact up to 5 random neighbors and the search request can be relayed up to 12 hops with random walk search. Using the Rendezvous Point approach, 1 % of the nodes in the system are rendezvous points and a Rendezvous Point informs peers within 7 hops.

B. Simulation Results

Our evaluation of search algorithms is based on how successful they are in finding idle cycles and how scalable they are, based on the number of messages sent. We use the following metrics which are oriented towards evaluating resource discovery. In future work, we will investigate classic job scheduling metrics, such as completion time and utilization, noting that they need to be revisited in an open peer-to-peer environment. The issues of performance metrics for peer-to-peer cycle sharing are discussed further in the section on Future Work.

- **Job Completion Rate.** Ratio of successfully finished jobs over total number of submitted jobs.

The job completion rate can be broken up into the following:
Job completion rate = first submission success rate + second submission success rate - migration failure rate
where, first submission success rate is defined as the number of jobs successfully scheduled the first time they are submitted divided by total number of jobs; second submission success rate is the number of jobs successfully rescheduled at night divided by total number of jobs; and the migration failure rate is defined as the number of jobs failed after migration divided by the total number of jobs.

- **Message Overhead.** This is defined as the number of links traversed by all messages in the resource discovery procedure divided by total number of peers in the system. In this simulation, the message overhead is the total per peer over 24 hours. For Rendezvous Point, the message overhead also includes the messages for advertising the rendezvous points. Since the message count indicates the number of overlay links traversed, the count will be much larger in the physical network.
- **Average Distance.** This is the average hop count in the overlay from a client to the hosts on which its job is scheduled.

B.1 Light Workload

This section shows the results under a light workload, with the ratio of clients to hosts set at 0.1 and assuming uniform client request arrival distribution. The results are similar with a normal arrival distribution under conditions of light workload.

Figure 2 shows that when there are abundant idle cycles, the job completion rate for Rendezvous Point without retry is very close or equal to the performance of a central server, with almost 100% completion rate over the full range of peer cycle withdrawal probability. Due to these two algorithms' inherent advantage in gathering knowledge of idle cycles, the client is guaranteed to find all or many of the hosts.

While the performance of expanding ring search and ads-based search are not as strong as the first two methods, the job completion rates remain greater than 97% (with retry) for peer cycle withdrawal probability up to 30 %. (We note that since scope limit is 5 hops for both, their performance is basically identical). Random walk is the weakest of the algorithms with job completion rates around 95%. The simulation results also show that all the algorithms have less than 5% migration failure rate. In general, all five search algorithms are not greatly impacted by peer withdrawal rates.

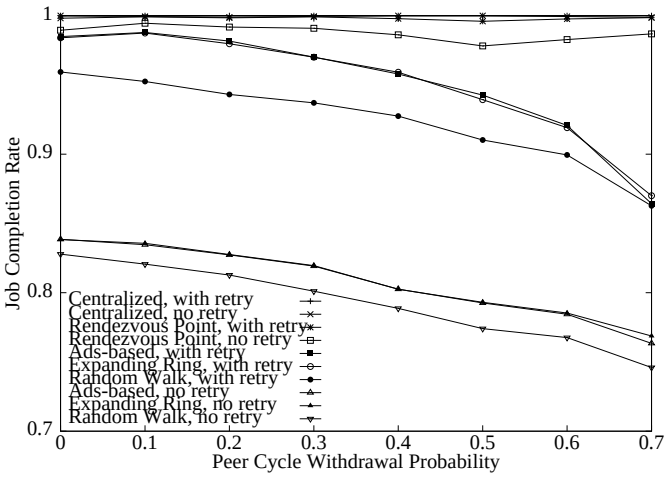

Fig. 2. Job completion rate under uniform workload, when the ratio of clients to donors is 0.1.

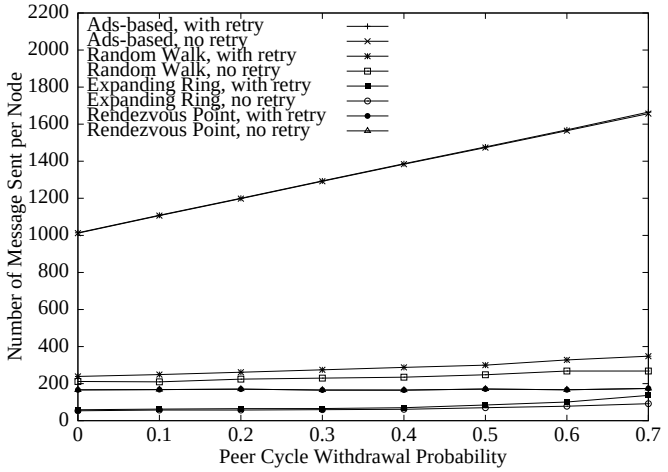

Fig. 3. Message overhead (for 24-hour period) under uniform workload, when the ratio of clients to donors is 0.1.

With a higher success rate on first submission, there is lower latency for scheduling the jobs. Using Rendezvous Point most clients (close to 100%) find enough hosts to start their tasks the first time they submit the tasks into the system. The other three methods only satisfy 75% to 83% of the tasks at first submission.

Figure 3 shows that the message overhead for Rendezvous Point is much lower than other techniques. The amount of messages sent for advertisement-based search is consistently high, since this technique uses flooding in a limited scope to advertise the profile information. The message overhead for random walk and expanding ring are comparatively small, as they are launched on-demand and under this light workload, the requests for extra cycles are infrequent.

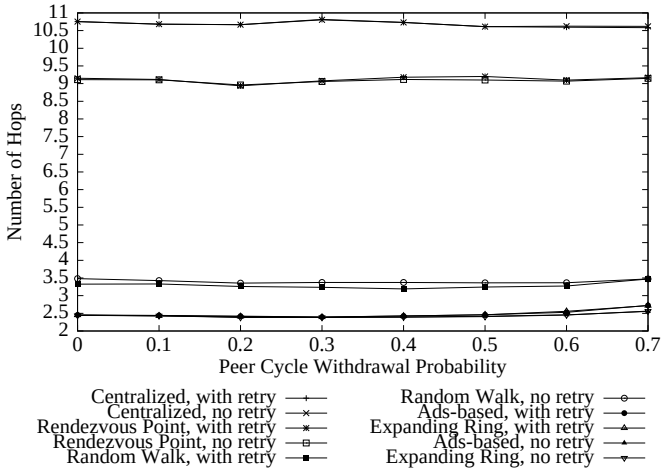

Fig. 4. Average distance from clients to donors under uniform workload, when the ratio of clients to donors is 0.1

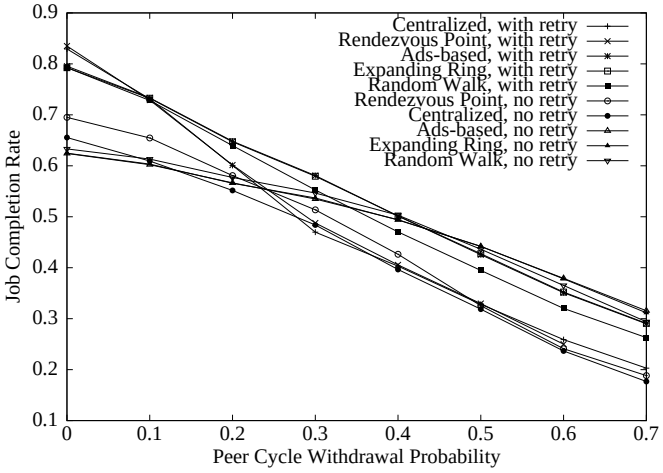

Fig. 5. Job completion rate under uniform workload, when the ratio of clients to donors is 0.7.

Figure 4 shows that random walk, expanding ring and ads-based search can only locate hosts in a local area, while the others can find hosts in a much larger range.

B.2 Heavy Workload

This section shows the results under a heavy workload, with the ratio of clients to hosts set at 0.7. We first present results under uniform client request arrival distribution.

Figure 5 shows that the performance of all of the search methods degrade greatly under a heavy workload. We expected Rendezvous Point to consistently outperform the other search techniques. However, its performance drops below the others when the cycle withdrawal probability becomes higher than 0.1. We investigated this decline in performance and found it is due to the ability of Rendezvous Point to discover more hosts in the system and thus be able to schedule larger jobs. In the CCOF environment, which satisfies requests on-demand, large jobs may block the smaller jobs from being scheduled. Table I compares of the average and maximum size of jobs scheduled by each algorithm with retry option turned on. Evidently, the average and maximum size of jobs scheduled by rendezvous point are much larger than the other three. We will discuss the nature and solution of this problem in section IV-C. In Figure 6 the search algorithms are divided into two groups according to their success rate on first submission. The upper group does not allow retry and the lower group allows rescheduling at night. This graph shows a low first submission success rate for the second group. The jobs that fail to be scheduled during the day are then rescheduled at night, hurting the performance of jobs submitted at night.

The central server and Rendezvous Point algorithms have a higher migration failure rate than the other search methods (see Figure 7). Because the larger jobs occupy more processors, the likelihood that one of its processors will withdraw is higher than for a small job.

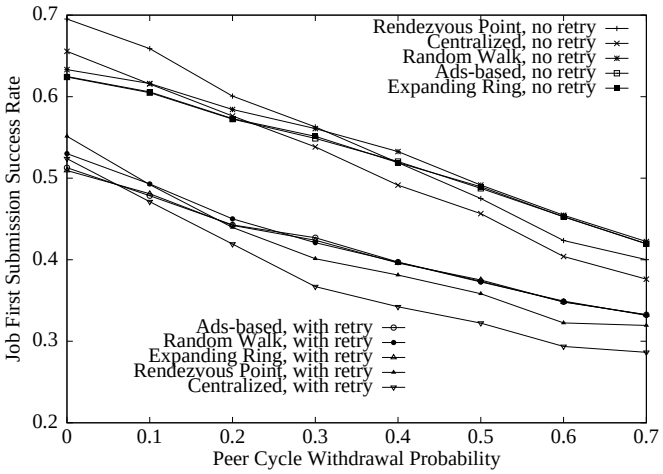

Fig. 6. Job first submission success rate under uniform workload, when the ratio of clients to donors is 0.7

Rendezvous Point has the lowest and most stable message overhead (see Figure 8). However, expanding ring exhibits much higher message overhead for a heavy workload, compared to light workload (see Figure 3). The number of messages sent by it grows rapidly, because as cycle withdrawal probability increases, it needs to frequently probe in the peer-to-peer system with greater scope.

We conducted the same set of simulations with normal client request arrival distribution (See Figure 9). Algorithms that allow retry are significantly better than the corresponding algorithms without retry, while the difference is not as dramatic under the uniform distribution. After the peak arrival rate, there are fewer tasks to compete with the rescheduled tasks.

Peer cycle withdrawn probability	Central Server	Rendezvous Point	Random Walk	Expanding Ring	Ads-based
	Avg-Max	Avg-Max	Avg-Max	Avg-Max	Avg-Max
0	14.9-176	14.8-145	13.4-122	13.1-114	13.0-111
0.2	14.5-132	14.8-145	12.9-109	12.2-108	12.2-104
0.5	13.6-111	13.0-109	11.8-95	10.9-88	10.9-95

TABLE I

Average/Max size of jobs scheduled under uniform workload, when the ratio of clients to donors is 0.7. (Job size is measured in unit of processor-hours)

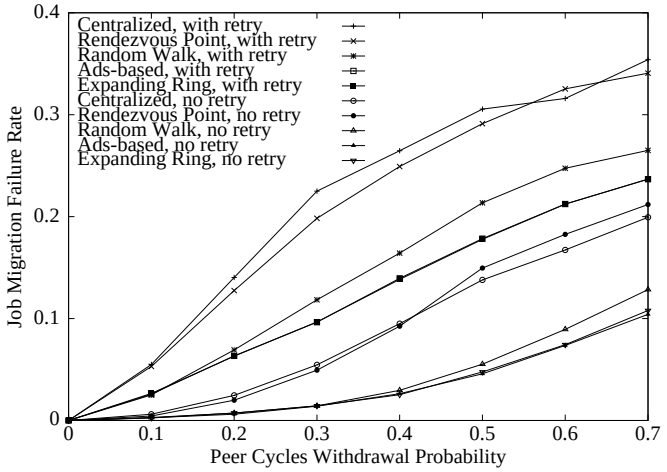

Fig. 7. Job migration failure rate under uniform workload, when the ratio of clients to donors is 0.7

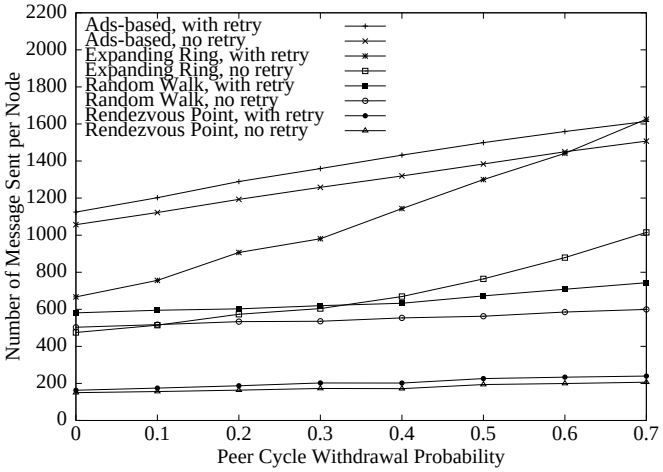

Fig. 8. Message overhead under uniform workload, when the ratio of clients to donors is 0.7

C. Key Observations

We make the following overall observations based on these preliminary simulations:

- When considering all metrics, the Rendezvous Point algorithm performs best overall under both light and heavy workloads because of its consistently low message traffic overhead. The declining performance with respect to job completion rate that we observe under heavy workloads can be addressed through scheduling policies that avoid favoring large jobs. However preventing large jobs from starving small jobs is a hard open problem in a peer-to-peer environment. (This problem has been successfully addressed for parallel machines with a central scheduler using techniques such as backfilling.) Our CCOF system will address this problem in conjunction with its approach to fairness, QoS and security through coordinated scheduling (meta-level scheduling).
- Under light workloads, while Rendezvous Point performs best, the other algorithms also perform well with job completion rates greater than 95%, when retry is allowed. They are stable over dynamic peer cycle withdrawal changes. Only ads-based incurs higher message passing overhead.
- Under heavy workloads, the job completion rate for all algorithms declines to values of 63% to 84% and drops quickly with increasing peer cycle withdrawal rates. The message overhead is still stable for Rendezvous Point, while it increases significantly for the other algorithms.

V. Future Work

This paper is an initial study as part of a more comprehensive analysis of resource discovery for CCOF. In the near future we plan to work on the following topics for workpile tasks: (a) Complete our study of dynamic host behavior by modeling hosts' joining as well as leaving the system, and by hosts' crashes; (b) Use more realistic profiles; (c) Use more realistic traces of client requests: Condor traces and suitable probability distributions for arrival rates and job sizes; (d) Design a more sophisticated Rendezvous Point protocol including scalable methods for dynamic Rendezvous Point selection; (e) Develop coordinated scheduling techniques that prevent large jobs from starving the smaller jobs. (f) Compare sender-initiated resource discovery (push models) with receiver-initiated resource discovery (pull models) in the peer-to-peer environment to see if their performance validates the common wisdom established in [6] for cycle sharing in traditional distributed systems. In these smaller scale systems, push models of load sharing perform better under light loads while pull models perform better under heavy loads.

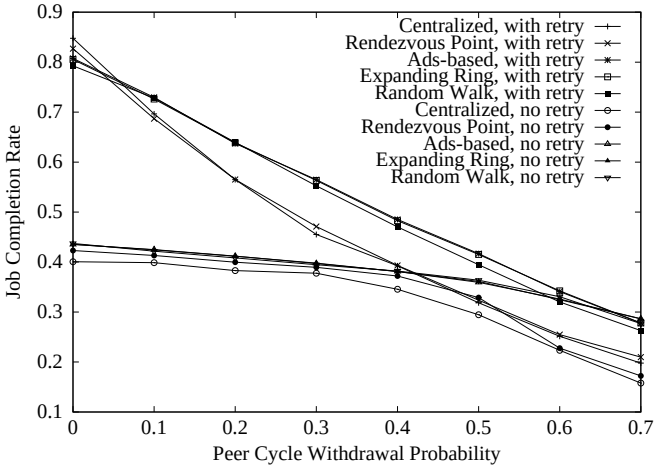

Fig. 9. Job completion rate under normal workload, when the ratio of clients to donors is 0.7

In the longer run, we are investigating resource discovery and scheduling for several important classes of applications beyond workpile that are well-suited to open peer-to-peer computing environments. These include workpile with deadlines; tree-based search such as branch and bound or alpha-beta search; and point-of-presence applications such as distributed monitors for security or traffic analysis, as well as content distribution and caching. We believe scheduling, QoS, and performance metrics and monitoring all involve scrutiny and re-evalution in an open, large scale peer-to-peer environment.

REFERENCES

[1] BOINC: Berkeley Open Infrastructure for Network Computing, http://boinc.berkeley.edu/.

[2] F. Annexstein and K. Berman. Modeling peer-to-peer network topologies through "small-world" models and power laws. In *Proc. IX Telecommunications Forum TELFOR 2001*, 11 2001.

[3] B. Hayes. Computing science - collective wisdom, March-April 1998.

[4] A. Butt, X. Fang, Y. Hu, S. Midkiff, and J. Vitek. An open peer-to-peer infrastructure for cycle-sharing. Poster in SOSP 2003.

[5] A. Butt, R. Zhang, and Y. Hu. A self-organizing flock of Condors. In *Proc. SC2003*, 2003.

[6] D. L. Eager and E. D. Lazowska and J. Zahorjan. Adaptive load sharing in homogeneous distributed systems. *IEEE Trans. on Software Engineering*, 12(5):662–675, May 1986.

[7] D.Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: An experiment in public-resource computing. *Communications of the ACM*, 45:56–61, 2002.

[8] F. Berman. High performance schedulers. In I. Foster and C. Kesselman, editor, *The GRID Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[9] Folding@Home Distributed Computing, http://folding.stanford.edu/.

[10] G. Fox, D. Gannon, S. Ko, S. Lee, S. Pallickara, M. Pierce, X. Qiu, X. Rao, A. Uyar, M. Wang, and W. Wu. Peer-to-peer grids. In *Grid Computing Making the Global Infrastructure a Reality*. John Wiley and Sons Ltd., 2002.

[11] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat. SHARP: an architecture for secure resource peering. In *Proceedings of SOSP 2003*, 2003.

[12] G. For and S. Pallickara. NaradaBrokering: an event-based infrastructure for building scalable durable peer-to-peer Girds. In *Grid Computing Making the Global Infrastructure a Reality*. John Wiley and Sons Ltd., 2002.

[13] I. Foster and A. Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Feb 2003.

[14] A. Iamnitchi and I. Foster. A peer-to-peer approach to resource location in grid environments. In J. Weglarz, J. Nabrzyski, J. Schopf, and M. Stroinski, editors, *Grid Resource Management*. Kluwer, 2003.

[15] A. Iamnitchi, I. Foster, and D. Nurmi. A peer-to-peer approach to resource location in grid environments. In *Proceedings of the 11th Symposium on High Performance Distributed Computing*, August 2002.

[16] J. Ledlie, J. Shneidman, M. Seltzer, and J. Huth. Scooped, again. In *Proc. 2nd International Workshop on Peer-to-Peer systems (IPTPS 2003)*, Berkeley,CA, Feb 2003.

[17] M. Litzkow, M.Livny, and M.W. Mutka. Condor hunter of idle workstations. In *In Proceedings of the 8th International Conference on Distributed Computing Systems*, 1988.

[18] M. Livny and R. Raman. High-throughput resource management. In *The GRID Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[19] V. Lo and J. Mache. Job scheduling for prime time vs. non-prime time. In *Proc 4th IEEE International Conference on Cluster Computing (CLUSTER 2002)*, Sep 2002.

[20] V. Lo, D. Zhou, D. Zappala, Y. Liu, and S. Zhao. Cluster computing on the fly: P2P scheduling of idle cycles in the Internet. In *Proc. 3rd International Workshop on Peer-to-Peer System (IPTPS 2004)*, San Diego, Feb. 2004.

[21] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proc. ACM SIGCOMM*, Aug. 2001.

[22] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal*, 6(1), 2002.

[23] A. Rowstron and P. Druschel. Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proc. 18th IFIP/ACM Int'l Conf. on Distributed Systems Platforms*, Nov. 2001.

[24] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup service for internet applications. In *Proc. ACM SIGCOMM*, Aug. 2001.