Cluster Computing on the Fly: P2P Scheduling of Idle Cycles in the Internet

Virginia Lo, Daniel Zappala, Dayi Zhou, Yuhong Liu, and Shanyu Zhao
Computer and Information Science, University of Oregon, Eugene, Oregon 97403-1202
{lo, zappala, dayizhou, liuyh, szhao}@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 identify four important classes of cycle-sharing applications, each with distinct requirements that call for application-specific scheduling strategies. Our Wave Scheduler exploits large blocks of idle time at night, to provide higher quality of service for deadline-driven workpile jobs, using a geographic-based overlay to organize hosts by timezone. To verify the results of workpile jobs, CCOF sends quizzes to hosts and uses the accuracy of the quiz answers to determine trust ratings. Our PoP Scheduler disperses tasks comprising a point-of-presence application, using scalable protocols to discover strategically located hosts to meet application-specific requirements for location, topological distribution, and resources.

The CCOF cycle sharing system encompasses all activities involved in the management of idle cycles: overlay construction for hosts donating cycles, resource discovery within the overlay, application-based scheduling, local scheduling on the host node, and meta-level scheduling among a community of application-level schedulers. Our work with CCOF reveals many of the critical challenges that lie ahead for P2P scheduling systems.

I. Introduction

Peer-to-peer computing, the harnessing of idle compute cycles throughout the Internet, offers an exciting new challenge for P2P networks beyond current information 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 [14], [7], [16]. This creates a compelling opportunity for research in this new juncture between the fields of networking and distributed computing.

This work was supported in part by the National Science Foundation under grant ANI-9977524.

Our research addresses the problem of *peer-to-peer computing*, which encompasses all of the activities involved with utilizing idle cycles from ordinary users in a distributed, open environment. In contrast to Grid computing [10], [12] and other institution-based cycle-sharing systems [17], we are targeting an open environment, one that is accessible by the average citizen and does not require membership in any organization. Peer-to-peer computing represents the next step in distributed computing, providing potentially greater computing power than institutional-based projects while also empowering ordinary users. This view of P2P computing is the focus of several other current research projects [13], [5], [4].

P2P computing in an open environment gives rise to a new generation of resource management problems that are dramatically different from those addressed by traditional scheduling systems, including issues of resource discovery, trust, incentives, fairness, security, and new criteria for evaluating performance. We use the term "P2P scheduling system" to encompass all activities involved in the management of idle cycles: overlay management for hosts donating cycles, resource discovery within the overlay, application-based scheduling, local scheduling on the host node, and meta-level scheduling which involves coordination of efforts among a community of application-based schedulers.

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 four important classes of problems that are particularly well-suited to capturing idle cycles in the Internet: infinite workpile, deadline-driven workpile, tree-based search, and point-of-presence applications.

Popular applications for harvesting idle cycles from ordinary users, such as SETI@home [8], are limited to CPU-intensive workpile applications and require donors of cycles to manually coordinate through a centralized web site. More general cycle-sharing systems, such as Condor [17], provide automatic scheduling but require a centralized matchmaking service and are limited to members of participating institutions. *Our goal is the development of a scheduling infrastructure that supports automatic scheduling of these four broad classes of applications in an open environment.*

In this paper, we discuss the problems faced by P2P scheduling systems that presume an open and large scale environment. We first present a taxonomy of P2P cycle sharing applications and their specific requirements. We then describe the *Cluster Computing on the Fly* architecture and discusses issues and open problems involved in the design of an open P2P scheduling system. We conclude by illustrating how CCOF addresses some of these problems. In particular, we introduce CCOF's *Wave Scheduler*, which harvests night time idle cycles by using geographic timezones to organize the hosts, and CCOF's method for verification of results returned by the hosts. We also describe our *PoP Scheduler*, which utilizes scalable protocols to schedule point-of-presence applications by discovering strategically located hosts to meet application-specific requirements for location, topological distribution, and available resources.

II. P2P SCHEDULING SYSTEMS

A. P2P Cycle-sharing Applications

We organize P2P cycle-sharing applications into four classes whose scheduling needs are starkly distinct, calling for individualized scheduling services that are tailored to those needs.

1) Infinite workpile applications: These applications consume huge amounts of compute time under a master-slave model in which the master delivers code to as many hosts as possible, over long periods of time. 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 at all between slave nodes. Examples of infinite workpile applications include SETI@home [7], the Stanford Folding Project [11], and numerous mathematical applications ranging from number theory to cryptography [14].

Infinite workpile applications need scheduling that can (a) automatically identify large blocks of idle cycles and (b) support validation of results. Performance may be measured by some large-grained metric representing the *yield* of idle cycles, such as tasks completed per day or week. For the protection of the wider community, safeguards need to be installed to provide some notion of fairness among competing workpile jobs, as well as security against denial-of-service attacks from a malicious job that preys on the generosity of participating hosts.

2) Workpile applications with deadlines: Deadline-driven workpile applications are similar to infinite workpile applications, but their needs for compute cycles are more moderate. These applications are deadline-driven because they require that results be returned within a specified deadline (on the order of days or weeks). Examples of this class of application include compute intensive jobs: complex insurance analysis, simulation experiments with a large parameter space, 3D modeling and ray tracing code. These jobs needs to be completed to meet a business presentation deadline, research publication deadline, or school project deadline.

While many scheduling strategies are suitable for infinite workpile jobs with and without deadlines, the urgency of deadlines calls for more aggressive approaches for discovery and scheduling of cycles.

3) Tree-based search applications: This class of applications requires substantial compute cycles, with loose coordination among subtasks requiring low communication overhead. Distributed branch-and-bound algorithms, alpha-beta search, and recursive backtracking algorithms are used for a wide range of optimization problems; these computationally intensive state-space search algorithms are ideal candidates for P2P scheduling.

Distributed branch-and-bound algorithms use a tree of slave processes rooted in a single master node. The tree dynamically grows as slave processes expand the search space and is dynamically pruned as subspaces leading to costly solutions are abandoned. There is a small amount of communication among slave nodes to inform other slaves of newly discovered lower bounds.

The scheduler manages the dynamic population of host nodes by continuously providing new hosts while the search tree is growing. It must also support communication among slave nodes, either indirectly through the master or directly from slave to slave.

4) Point-of-presence applications: PoP applications typically consume minimal cycles but require placement throughout the Internet (or throughout some subset of the Internet). The dispersement of tasks from a PoP application is driven by specific requirements of the job. For example, distributed monitoring applications (security monitoring, traffic analysis, etc.) require widely and evenly distributed placement as well as placement at strategic locations. Testing of distributed protocols requires placement of test-bots

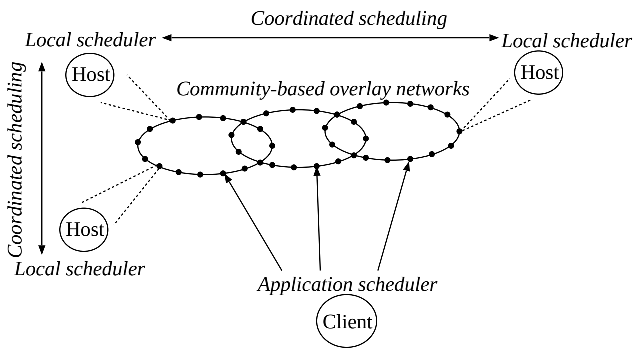

Fig. 1. CCOF Architecture

dispersed throughout the Internet in a manner that captures a variety of realistic conditions with respect to latency, bandwidth, and server performance. In addition, security concerns must be addressed by limiting communication only to the set of PoP tasks

B. An Open P2P Cycle-Sharing Architecture

In the CCOF 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 sets of machines from these overlays. The basic service offered by CCOF is best-effort in the sense that any host may preempt guest code 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, highlight the many complex research issues to be addressed in the design of an open P2P scheduling system which include the following:

1) Overlay management of host communities: An important area of research for P2P computing lies in the organization of communities of hosts willing to share idle cycles. One way to organize such communities is through the creation of overlay networks based on factors such as interest, geography, performance, trust, institutional affiliation, or generic willingness to share cycles. Communities may span multiple organizations, such as a collaborative research project among several research groups. Chess enthusiasts, or participants in the Intel Philanthropic project [16] may form a community based on their hobbies or a spirit of volunteerism. Users may form nested communities based on trust, such as a group of family and friends, co-workers, and customers. We assume that it is possible to exclude untrusted hosts from the overlay using a central certificate-granting authority, as proposed by Pastry [21].

This arena of research exceeds the immediate bounds of scheduling research per se, but has a direct impact on it. Past work on overlay networks [20], [24], [21], trust and reputation systems, performance analysis and monitoring all serve as the foundations for new work in this direction.

2) Resource discovery: The problem of resource discovery is extremely difficult in an open environment in which the set of participating hosts is potentially very large and highly dynamic. Resource discovery takes on new dimensions when the resource (compute cycles) is perishable, cannot be shared, and is dynamic. Moreover, the search for idle cycles may be coupled with the need for other resources such as memory and data. Work has already begun in this area, much of it focused towards institutional and Grid Computing environments.[15], [17], [9].

We conducted a comprehensive study of generic searching methods in a highly dynamic environment for workpile applications [26]. We compared 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 with respect to job completion, 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. With respect to message-passing overhead, rendezvous point dramatically outperforms the other discovery strategies.

3) Application scheduling: The application scheduler is responsible for selection of hosts for a P2P computation from a pool of candidates, for exporting the code to the hosts, and for managing and verifying returned results. Application scheduling requires an analysis of both the application's needs and the nature of the offered resources. The scheduler's decision about which hosts to select becomes very complex since trust, performance, and fairness all come into play. Strategies such as oversubscription and duplication scheduling, may be used for maximum flexibility. Furthermore, if coordination across a set of host nodes is required, it may be desirable to organize the selected hosts into a new overlay to support the interprocess communication needs of the application, further complicating the host selection process.

4) Local scheduling: The local scheduler tracks idle cycles and negotiates with the application scheduler using local criteria for trust, fairness, and performance to decide which tasks to accept. It also interacts with its own native scheduler to inject those jobs into the scheduling queue.

Because each host may implement its own policies, we envision Quality of Service as an important feature of P2P computing. The job of a local scheduler can be viewed as an admission control problem, similar to that faced by integrated services networks [6]. Some hosts may provide guaranteed service by accepting only CCOF jobs – they have no local tasks that may preempt guest code. Likewise, a host may offer predictive service by providing a statistical performance estimate based on past behavior.

5) Incentives and fairness: The choice of policies and mechanisms for fairness in a P2P scheduling system is a non-trivial decision since the notion of fairness is itself open to debate. Several current projects require *strict fairness*, fairly tightly controlled accounting of cycles and resources consumed through a variety of economic and social models (credit-based systems, bartering systems and auction-based systems, ticket-based systems) [13], [5], [1]. Our philosophy is much more open - we presume that the donors of idle cycles are more relaxed: they are concerned with immediate access to idle cycles but not cycle-for-cycle equality. Under this model of *long-term fairness*, it is still necessary to enforce some kind of accounting. In either case, interesting research questions arise with respect to what incentives work best, how to measure contributions and usage for fairness in a large scale and dynamic environment.

6) Trust and reputation: Trust and reputation systems are needed throughout P2P scheduling for the formation of trust-based overlays as described above, for local hosts to use when deciding whether to accept or reject tasks from an application, and for applications to use to select host nodes from multiple candidates. Several new trust systems for P2P networks have recently been developed that can potentially be utilized in an open environment [22], [23]. However, several difficult problems remain: how to effectively utilize the trust values to make scheduling decisions and also how to determine whether the results returned by a host are correct or not. In Section 3 we describe how our CCOF application scheduler probes host nodes using undetectable quiz codes to develop a trust rating for each host, as well as to validate returned results.

7) Security: How can a host defend itself against denial of service attacks, such as when a malicious node occupies large numbers of hosts with useless code? Or worse yet uses hosts to launch a distributed denial of service attack or a worm?

We assume that hosts protect themselves from a variety of attacks by running guest code within a virtual machine monitor, creating a ”sandbox” that protects the host and controls resource usage [3]. Despite this protection, hosts must still protect their resources from being misused. Preventing denial-of-service attacks, particularly those that are launched from the cluster itself, is a difficult problem. Likewise, the CCOF system itself can be abused if malicious users schedule large numbers of useless tasks.

One possibility for coping with these problems is for hosts to deny network access for untrusted clients, using the trust and reputation systems discussed above. Likewise, users can give priority to projects they have deemed trustworthy through any outside form of communication. In cases where a greater degree of openness is desired, it may be possible to use network monitors to detect and take action against attacks. We are working closely with fellow researchers at the University of Oregon who are working on distributed detection of worms and denial-of-service attacks.

8) Performance monitoring: The P2P scheduling environment is radically different from traditional scheduling domains. Evaluation of scheduling performance is thus faced with several challenges. What are are the appropriate metrics, benchmarks, performance monitoring tools and techniques for an open P2P scheduler? What kinds of user interfaces, testing, simulation, and monitoring tools will be most effective?

III. Cluster Computing on the Fly

In this section we present CCOF's support for two classes of P2P applications: deadline-driven workpile applications and point-of-presence applications. Workpile jobs, with their heavy demand for free cycles, are scheduled using CCOF's Wave Scheduler, which captures available night time cycles in timezones from east to west. CCOF also provides a simple yet effective quiz system for verifying the results returned by hosts. CCOF scheduling of PoP applications involves fully distributed algorithms for careful placement of tasks according to topological or performance-based criteria.

A. Wave Scheduling for Workpile Applications

Wave scheduling seeks to capture cycles from the millions of machines that lie completely idle at night.

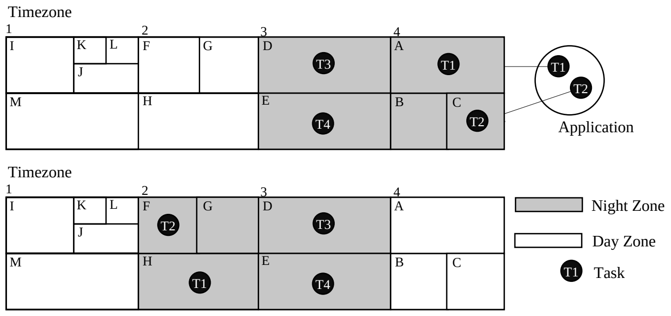

Fig. 2. Task initiation and migration in wave scheduling

By following night timezones around the globe, it continuously gives workpile tasks dedicated access to cycles without interruption from users reclaiming their machines. Wave scheduling is thus particularly useful for workpile applications with deadlines since it provides a higher guarantee of ongoing available cycles. Wave scheduling was motivated by the notion of prime time v. non-prime time scheduling regimes enforced by parallel job schedulers [25], [19] and by wave scheduling in systolic computing systems.

The CCOF Wave Scheduler uses a CAN-based DHT overlay to organize nodes located in different time zones. Our wave scheduling protocol functions as follows (see Figure 2).

1) Timezones in the CAN overlay: . We select one dimension of the d-dimensional mesh to represent time zones. For example, a 1 x 24 CAN space could be divided into hourly zones 1 through 23 based on its second dimension.

2) Host nodes join the overlay: A host node that wishes to offer its night time cycles knows which time zone it occupies, say zone 8. It randomly selects a node label in zone 8 such as (0.37, 7.12) and sends a join message to that node. According to the CAN protocol, the message will reach the physical node in charge of CAN node (0.37, 7.12) who will split the portion of the CAN space it owns, giving part of it to the new host node.

3) Application scheduler selects initial nightzone: The scheduler for a workpile application knows which timezones are currently nightzones. It selects one of these nightzones (based on some nightzone selection criteria) and decides on the number H of hosts it would like to target. This number could be an over-estimation of the number it would ideally like to schedule in order to achieve more flexibility.

4) Application scheduler selects set of target hosts: The scheduler randomly generates a collection of H node labels in the target nightzone. It sends a request message to each target host using CAN routing which finds the physical host managing that host node. After negotiations, the application scheduler selects a subset of those nodes to ship tasks to. It can make its selection based on trust or other criteria.

5) Migration to next timezone: When morning comes to a host node, it selects a new target nightzone, randomly selects a host node in that nightzone for migration, and after negotiating with that host, migrates the unfinished task to the new host.

6) Returning results to the application: Whenever a task finishes computing its results on a host node, it sends the results directly back to the application. If the application is offline, it can store the results in the CAN distributed file system. The application can later retrieve results using the DHT lookup.

We are investigating the use of wave scheduling for deadline-driven workpile tasks to see if it compares favorably with two other models: centralized master-slave approach and distributed master-slave approach. In both latter cases, idle nodes advertise their availability dynamically to the masters and the only option is to sleep when cycles are withdrawn. Our current evaluation includes application harvest yield (results returned per day); utilization (fraction of idle cycles being consumed relative to demand); host impact; and overlay traffic generated by the scheduler, by task migration, and return of results.

Our current implementation and evaluation is simulation-based, but in the future, we would like to do empirical experiments assuming we can find sufficient numbers and placements of volunteer nodes.

B. Trust-based Verification for Workpile Applications

We validate the correctness of results returned by host nodes to the workpile application using a quiz mechanism. The application node sends a set of quizzes to the hosts whose solutions are known beforehand. Based on the hosts performance on the quizzes, the application can then decide whether to accept or reject the results. We are investigating two methods for quizzing hosts. One method uses quizzes that are distinct from the actual application code. These are packaged so that a (malicious) host cannot distinguish quiz code from genuine application code. If the host passes its quiz, it will then be sent another task, this one containing application code. The second method embeds simple short quizzes into the application code. Quiz and application results are periodically sent back from the host to the application. If the application node receives wrong quiz answers, it can immediately reschedule the task on another host. Note that for both methods, when a host does return correct (or incorrect) results, this information can be used to give that host an appropriate trust ranking. This information is stored in a reputation system such as Eigenrep [23] or TrustMe [22].

C. Scheduling Point-of-Presence Applications

Our investigation of scheduling PoP applications is motivated by collaborative work at the University of Oregon on distributed security monitoring systems and distributed massively multiplayer games.

Topological distribution of monitoring tasks involves placement of these tasks on selected hosts throughout the overlay such that ordinary nodes are within t hops of j host nodes. This problem has been proposed abstractly as leader election in distributed systems, and as the dominating set problem in graph theory.

This task is much more difficult in an open, large scale environment. One of our approaches uses a torus-based overlay similar to CAN [20] that provides its regular node labeling scheme and Lee distances [18] to elect leaders using only local computation in constant time. Another approach is a fully distributed protocol that uses gossiping in a tournament-style backoff algorithm. Initially, each node says hello to its k-hop neighbors. Nodes then gossip with its immediate neighbors and either persists as a leader or backs-off based on the information each neighbor provides on the number of non-leader nodes it covers. We have developed several versions of this algorithm that trade off communication overhead and latency for accuracy. We are also investigating placement of game-bots throughout the Internet that meet varying level of performance with respect to network bandwidth and computational power.

REFERENCES

[1] N. Andrade, W. Cirne, F. Brasileiro, and P. Roisenberg. OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing. In *9th Workshop on Job Scheduling Strategies for Parallel Processing*, 2003.

[2] A. Arpaci-Dusseau. Implicit Coscheduling: Coordinated Scheduling with Implicit Information in Distributed Systems. *ACM Transactions on Computer Systems*, 19(3), August 2001.

[3] et al. B. Dragovic. Xen and the Art of Virtualization. In *ACM Symposium on Operating Systems Principles*, October 2003.

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

[5] A. Butt, X. Fang, Y. Hu, S. Midkiff, and J. Vitek. An Open Peer-to-Peer Infrastructure for Cycle-Sharing. Poster, SOSP 2003.

[6] David D. Clark, Scott Shenker, and Lixia Zhang. Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism. In *ACM SIGCOMM*, August 1992.

[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] D.Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: An Experiment in Public-Resource Computing. *Communications of the ACM*, 45, 2002.

[9] F. Berman. High-Performance Schedulers. In I. Foster and C. Kesselman, editor, *The GRID Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[10] F. Berman and G. Fox and A. Hey, editor. *Grid Computing Making the Global Infrastructure a Reality*. John Wiley and Sons Ltd., 2002.

[11] Folding@Home. http://folding.stanford.edu/.

[12] I. Foster and C. Kesselman, editors. *The GRID Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1999.

[13] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat. SHARP: An Architecture for Secure Resource Peering. In *SOSP*, 2003.

[14] B. Hayes. Computing Science - Collective Wisdom. *American Scientist*, March-April 1998.

[15] A. Iamnitchi, I. Foster, and D. Nurmi. A Peer-to-Peer Approach to Resource Location in Grid Environments. In *Symp. on High Performance Distributed Computing*, Aug. 2002.

[16] The Intel Philanthropic Peer-to-Peer Program. http://www.intel.com/cure/.

[17] M. Litzkow, M.Livny, and M.W. Mutka. Condor - A Hunter of Idle Workstations. In *Eighth International Conference on Distributed Computing Systems*, June 1988.

[18] V. Lo, B. Bose, and B. Broeg. Lee Distance, Gray Codes and the Torus. *Intl. Journal of Telecomm. Systems*, Oct 1998.

[19] V. Lo and J. Mache. Job Scheduling for Prime Time vs. Non-Prime time. In *4th IEEE International Conference on Cluster Computing (CLUSTER 2002)*, Sep 2002.

[20] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content Addressable Network. In *ACM SIGCOMM*, Aug. 2001.

[21] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *18th IFIP/ACM Int'l Conf. on Distributed Systems Platforms*, Nov. 2001.

[22] Aameek Singh and Ling Liu. TrustMe: Anonymous Management of Trust Relationships in Decentralized P2P Systems. In *IEEE Intl. Conf. on Peer-to-Peer Computing*, Sep. 2003.

[23] S.Kamvar, M. Schlosser, and H. Garcia-Molina. The Eigentrust Algorithm for Reputation Management in P2P Networks. In *12th International World Wide Web Conference 2003*, 2003.

[24] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *ACM SIGCOMM*, Aug. 2001.

[25] Veridian Systems. OpenPBS v2.3: The Portable Batch System Software, Sep. 2000.

[26] D. Zhou and V. Lo. Cluster Computing on the Fly: Resource Discovery in a Cycle Sharing Peer-to-Peer System. In *To appear, IEEE Intl. Symp. on Cluster Computing and the Grid*, 2004.