Grid Resource Discovery using Small World Overlay Graphs

Kashif Ali
Department of Computer Science and Engineering, York University Toronto, ON, Canada
e-mail: kashif@cs.yorku.ca

Suprakash Datta
Department of Computer Science and Engineering, York University Toronto, ON, Canada
e-mail: datta@cs.yorku.ca

Mokhtar Aboelaze
Department of Computer Science and Engineering, York University Toronto, ON, Canada
e-mail: aboelaze@cs.yorku.ca

Abstract

Computational grids are believed to be an effective and scalable solution to the problem of resource sharing over large, heterogeneous networks of computing devices. Since grids are highly distributed in nature, one of the most challenging problems is the discovery of dynamic resources in a grid. In this paper we use ideas from P2P systems to propose a solution for the problem. Specifically, we classify nodes as consumers and producers, depending on whether they consume or produce more jobs. Our algorithm connects all producer nodes using a overlay network that is a small-world graph (the graph is produced by adding "shortcut" chords to a circle). The consumer nodes hang off the small world graph. The producer nodes are forced to take part in resource cataloging and discovery. This has three distinct advantages – first, it prevents "freeloading" by forcing producers to do useful work; second, it frees the consumers to only do computations; third, the low diameter of the overlay graph ensures that all resources are within a small number of hops.

We simulate and evaluate the performance of our algorithm in realistic traffic conditions. We evaluate the performance of our algorithm using metrics like the average time to answer the query, the average number of requests that were dropped and the average number of hops traveled by query packets. Our experiments show that our algorithm performs well with thousands of nodes.

Keywords— *Computational grids, P2P system, resource discovery, overlay graph, small world networks.*

1. Introduction

Computational grids are a scalable, distributed infrastructure for sharing large number of heterogeneous resources (CPU/memory/disk space) in a distributed network for cooperative problem solving. Grids allow computing devices with different computational and communication resources to collaborate on solving problems, and function as a distributed supercomputer. While the basic idea behind grids has existed for decades, there has been a recent surge in designing user-friendly middleware (often called *gridware* or *grid engines*)for allowing widespread use of grids. In [3], the *Grid problem* was defined as "flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources ...". Thus, grids need to solve several problems, including the discovery of resources, authentication and authorization of messages/jobs, and scheduling of resources. In this paper, we propose solutions for the problem of discovering resources and matching jobs to compatible resources. The highly distributed and dynamic nature of grids makes the resource discovery problem far more challenging than traditional parallel computers or computer networks.

1.1 The resource discovery problem

Since a grid is made of heterogeneous resources, not all nodes on the grid can support a given job. Therefore, a job must first find a processor with compatible resources before it can begin execution. The situation is complicated by the fact that the resources in the network change dynamically over time, due to nodes joining or leaving the network. Note that a solution to the resource discovery problem necessary but not sufficient – another problem that must be solved is scheduling – for example, suppose that only two nodes in the network are compatible with a job. Therefore, the job must find one of the two processors, but then it should (ideally) go to the one that would allow it to finish earlier. It would make little sense if it queued up at one that is backlogged if the other is currently idle. We focus on the resource discovery problem in this paper and defer the scheduling problem for future work.

1.2 Grids and Peer-to-Peer systems

Computational grids have significant similarities and differences with another class of popular distributed system that is used (primarily) for sharing files, viz., Peer-to-Peer (henceforth called P2P) systems. The main similarities are that in both cases the network changes dynamically due to nodes joining and leaving the network. This makes maintaining centralized global views of the network unrealistic in both cases. The main difference is the fact that the resources being shared change dynamically for grids but not P2P systems. For example, a peer that has a file to serve requests for that file at any time, but a node whose hard disk is used up by a job cannot accept another job even though it may satisfy the processor memory and other requirements of the new job.

While they started as completely disjoint streams of research, grids and P2P systems have converged in many respects (see [2] for one account). We believe that the solutions to several problems for the grid can be contained by studying similar problems for P2P networks. The major issues that need to be addressed in both grids and P2P networks include

Indexing of resources: P2P systems can be unstructured or structured – the former do no indexing and locate resources by exhaustive search, while the latter tries to maintain distributed databases of all resources (files) currently in the system.

Load balancing : It is desirable to balance load across different peers when choosing a peer to satisfy a given request. This alleviates both computational and networking loads in the system.

Freeloading : Measurement studies have shown that a majority of users join P2P networks to download files but do not share any files of their own. This phenomenon (called freeloading or free riding) drains resources and needs to be discouraged.

1.3 Related work

Several fully decentralized resource discovery algorithms were proposed and evaluated in [4]. Our work improves on this paper by organizing the nodes in a way that makes resource discovery problem easier.

There has been a lot of research on P2P networks in recent years. We refer the interested reader to the excellent tutorial [8].

In [5], the authors propose a P2P approach to resource discovery. Unlike our work, however, they do not impose any overlay network. Instead, they investigate several request forwarding strategies by peers.

In [6], the authors assume that small world graphs exist due to user actions (and show that this is indeed true in some P2P systems) and investigate different techniques for storing directory information and locating files. However, they do not address the question of explicitly building overlays that possess small world properties.

1.4 Our contributions

The contributions of this paper are as follows.

• We propose to structure the grid using a small-world overlay graph that has several advantages, including prevention of freeloading and efficient resource discovery.

• We develop a very accurate simulation platform for evaluating the performance of resource discovery algorithm under realistic network conditions.

2. Overlay graph construction

In this section we describe the design issues for overlay construction and the solutions we propose in this paper.

2.1 Design issues

The main design issue that we faced when designing an overlay network was the choice between a structured versus an unstructured overlay. We felt that many commonly used structured overlays, including distributed hash tables (DHTs) were not suitable for our purposes since the resources in a grid were more varied in a grid and also varied greatly with time. On the other hand, it was desirable to have some structure as that would improve the efficiency of resource discovery algorithms. Based on these considerations, we chose to have a structured overlay that is not based on partitioning and sharing the address space of resources as DHTs are. Instead, our overlay graph is maintained to be a low-diameter network in which searching is more efficient than in unstructured networks.

Our overlay design reduces the effect of freeloading in the following way. We classify (dynamically) nodes into two classes – *producers* (who produce jobs) and *consumers* (who consume or execute the jobs). If a node currently classified as a consumer has not executed at least $f(W)$ jobs in the last W timesteps, then we change its class to producer. We assign resource discovery tasks to producers but not to the consumers. This makes freeloading nodes do useful work for the system.

Bootstrapping, or finding a node to connect to the system, is a nontrivial problem for most distributed P2P systems. However, we can avoid that problem for grid applications by requiring that a small number of nodes remain in the grid overlay at all times. This is a reasonable assumption in grids (unlike P2P systems). These nodes (called *constant nodes*) can have their addresses published and distributed, thus solving the bootstrapping problem. In order to alleviate their load, we allow constant nodes to pass on any jobs it gets to consumer nodes.

2.2 Topology

Our overlay construction is the dynamic version of a well-known algorithm for generating *small world* graphs, which are a family of graphs with several nice properties, most notably low diameter and low average degree. Small world graphs have been found to be important in many different domains. We refer the interested reader to [9] for details.

The constant nodes allow the overlay graph to exist at all times. The overlay graph has the *core nodes* (the producers and the constant nodes) organized in a ring, and consumer nodes "hang off" the ring. Constant nodes also store the relevant state information needed to classify nodes.

In order to maintain the small diameter property, several extra edges are added to the circle. In addition to each node having edges to its previous and next nodes on the circle, there are two more types of edges:
local edges: these go from node i to $i+2, i+3, \ldots, i+k$.
shortcut edges: these go from i to some randomly chosen node. These edges are responsible for greatly lowering the diameter of the network.
In addition, each core node maintains pointers to a set of consumers. For simplicity, all edges are bidirectional in our overlay.

2.3 Join and leave protocols

As mentioned, a node contacts a bootstrap node and declares itself to be a producer or consumer. It is then connected to the overlay accordingly. Our system trusts incoming nodes to make the correct declaration. Since nodes are classified dynamically, the nodes cannot benefit much from false declarations.

An incoming node is directed by the constant node it contacts to a random location in the ring and it joins the overlay at that location. It then constructs local edges using its neighbors and shortcut edges randomly. Our current design does not require that nodes share its list of consumers. Instead, core nodes cache requests and build up "experience" by caching the results of queries over time. We plan to implement explicit sharing of consumer lists in future.

When a consumer node leaves, it informs all its neighbors so that the latter can update their edge lists. When a producer node leaves, it also informs its neighbors. It then passes its list of consumers to a constant node, who assigns them randomly to nodes. We do not require departing nodes to hand over their "experience" to other node.

The dynamic changing of status of consumer and producer nodes is done very simply by treating each status change as a leave and immediate join by the same node but with a different status. We note that the status changes have significant overhead costs both in terms of storage/maintenance of state at constant nodes as well as network traffic due to updating the constant nodes.

3. Resource discovery algorithm

We designed our system to be a testbed for evaluating new resource discovery algorithms. In this paper, we look at two very simple algorithms. The first one is a simple randomized algorithm that operates as follows. Each resource discovery query is given a hop-count limit and routed to a random neighbor. The process goes on until the hop count is exceeded or some node responds indicating that it has the resources being asked for.

While such a simple algorithm may not perform well for many graphs, especially sparsely connected ones, we expect it to benefit from the inherent structure of small-world networks. Specifically, since all nodes are within a small number of hops from each other and the average node degree is low, the chances of successfully locating resources should be significantly higher than in traditional graphs.

The second algorithm we use is the above algorithm supplemented with the caching of successful queries. This is done simply by sending the query response along the same path it used to arrive. When a query exceeds its hop count, a message is sent to the source using the path of the query. This scheme requires that core nodes must store some state information about the queries while it is being routed. High values of the hop count would result in a large amount of state information while a small value would decrease the likelihood of success.

4. Accurate simulations

One of the objectives of this work was to develop a testbed for fine-grained simulation of resource discovery algorithms. Most resource discovery simulators in the literature do not model the underlying network in terms of topology and do not simulate low level network phenomena (e.g., packet losses, delays). We felt it is important to simulate the network, since some resource discovery algorithms (especially those using flooding) can add significantly to the network traffic, and affect the delay in getting query responses. We chose to use the BRITE package [7] for generating Internet-like topologies for the underlying physical connection graph. We chose to implement our simulator on top of the well-known network simulator ns-2 [1]. While this complicates the implementation in some ways, it gives us implementations of most protocols in the TCP/IP architecture. An unexpected disadvantage that we observed was that ns consumes a lot of overhead and severely limits the scalability of our simulations.

5. Performance evaluation

This project is work in progress and some aspects of the system have not been implemented yet. In this section we outline the status of our work and present performance evaluation of the currently implemented parts.

Among the features we have not implemented yet are node failures or abnormal terminations. We are also

TABLE I
SIMULATION RESULTS: RANDOM

	Random
# of successful queries	67.4%
Ave. path length	5.5
Ave. response time	0.285
Ave. queries dropped/node	0.95

testing the implementation of the dynamic classification of producers and consumers. Hence the results in this section are for situations where producers keep producing jobs and consumers keep accepting and executing jobs.

We evaluate the performance of our algorithm using simulation. We compared the performance of our algorithms using metrics like the average time to answer the query, the average number of requests that were dropped and the average number of hops of query packets.

The following results were obtained by simulating 500 nodes for 500 timesteps. The interarrival time for requests was 20. We had 5 constant nodes. The others were producers and consumers with equal probability. We allow every node to generate jobs, and therefore every node generated 25 jobs on average in the period simulated. The underlying graph was generated by the ASWaxman module of the BRITE package. Each node randomly chooses a set of resources on joining. Queries were generated for random resources.

TABLE II
SIMULATION RESULTS: RANDOM WITH CACHING

	Random with caching
# successful queries	69.13%
Ave. path length	5.29
Ave. response time	0.346
Ave. queries dropped/node	0.88

The two tables show that both algorithms world reasonably well. Random with caching has a slim advantage in getting a higher percentage of queries routed successfully while Random provides a slightly faster response time.

Although the above results are for 500 node networks, we could run simulations for upto 2000 nodes. Beyond that the time as well as resources used by ns became prohibitive.

6. Conclusions

In this paper, we proposed a solution to the resource discovery problem in computational grids. We organize the nodes dynamically in a small-world overlay graph that forces every node to contribute computational resources (CPU/memory/disk space) to the grid. Our overlay graph has the nice properties of having low average node degree and low diameter. This allows our resource discovery algorithms more efficient. We simulated and evaluated our algorithm under realistic network and traffic conditions. Our experiments show that our algorithm performs well and scales to thousands of nodes. We are currently in the process of extending the simulator to handle the scheduling and execution of jobs.

Acknowledgments

The last two authors were supported by NSERC discovery grants.

References

[1] The network simulator: ns-2. Available at `http://www.isi.edu/nsnam/ns/`.

[2] 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)*, February 2003.

[3] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[4] A. Iamnitchi and I. Foster. On fully decentralized resource discovery in grid environments. In *International Workshop on Grid Computing*, November 2001.

[5] A. Iamnitchi, I. Foster, and D. C. 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.

[6] A. Iamnitchi, M. Ripeanu, and I. Foster. Small-world file-sharing communities. In *Proceedings of Infocom 2004*, March 2004.

[7] A. Medina, I. Matta, and J. Byers. Brite: A flexible generator of internet topologies. Technical Report BU-CS-TR-2000-005, Boston University, 2000. `http://www.cs.bu.edu/brite`.

[8] K. Ross and D. Rubenstein. Tutorial on P2P systems, 2004. presented at IEEE INFOCOM, `http://cis.poly.edu/~ross/tutorials/P2PtutorialInfocom.pdf`.

[9] D. J. Watts. *Small Worlds: The Dynamics of Networks Between Order and Randomness.* Princeton University Press, Princeton, 1999.