P2P Grid: Service Oriented Framework for Distributed Resource Management

Prem Uppuluri, Narendranadh Jabisetty, Uday Joshi, Yugeyung Lee
School of Computing and Engineering
University of Missouri – Kansas City
{uppulurip, njn88, umjzn9, leeyu} @ umkc.edu

Abstract
With the increasing number of computers on the Internet, there is a growing interest in harnessing the unused and inexpensive computational resources over the Internet. However, current approaches such as the Grid computing paradigm are not sufficient. We present our preliminary work that uses extends Peer-to-Peer (P2P) computing with a framework that allows Grid computing over the internet.

1. INTRODUCTION
It is difficult to port the Grid computing paradigm to harness the idle services that are available on the Internet for ordinary users to run large computational tasks. These services are provided without prior user agreements and keep changing dynamically. However, current Grid [8] computing approaches are (a) focused on architectures for niche problems, and, (b) cannot accommodate dynamically generated services. In particular, the distribution of resources and applications that execute on Grid are only suitable within specific Grid infrastructures, e.g., Globus[18] can only be implemented in an Intranet. In addition, without access to dynamically generated services, there is a gap between the service consumer and the service producer's view leading to low service utilization, which in turn becomes pronounced, when it needs interaction with other services.

We are working on an approach that addresses these two issues. Specifically, we are looking at using Peer-to-Peer (P2P) mechanisms to support Grid computing. The salient contributions of our work in the new P2P Grid framework are:

- Extending P2P’s keyword based search to a more powerful matchmaking model. The model allows and matches the specifications of available services at a fine granularity, and, presents a ranking among the matches to select the best possible matches using the multi-attribute decision models (MADM) [5] (Sections 3 and 4).
- Evaluates existing P2P technologies. (Section 5).
- Demonstrates key implementation issues with a scalable proof of concept. (Section 6).

2. RELATED WORK
Key research in Grid computing [6,7,12] is focused on developing standards and toolkits and hence cannot naturally be applied for harnessing the power of the internet. In our approach we develop the Grid on top of a P2P network. Work in this area usually differs in the type of search algorithm used: Gnutella[22] and enhanced Gnutella[9] use a distributed search, while Pastry[11] utilizes routing mechanisms to achieve a greater scalability. P2P and Grid differ in their focus. P2P systems deal with issues such as: networking the hosts, building indexes of data to be shared, enabling query searches, and file transfer. On the other hand, Grid operates at a higher level of abstraction and assumes the presence of a network, but does not depend on the lower-level details being used. Comparisons and synergy between them have been provided by [12,7,2,1,4,15]. Our approach differs from theirs in terms of resource specification and matchmaking. For the former, several languages exist including Condor [17], and, the Metacomputing Directory Service (MDS) in Globus, distributed repositories services [3]. For matchmaking, [16] proposed an ontology, [13,14] used Gang-Matching for multiple independent resources to be matched simultaneously and [10] adopted a set-matching approach. All these, either require central servers, are static or need knowledge about prior servers. Our approach is distributed, dynamic in resource discovery and avoids central servers.

3. THE P2P GRID FRAMEWORK
To understand the needs of resource specification, we surveyed the current resource types and their computing needs in Grid architecture. Table 1 illustrates some of these resource needs. A more comprehensive survey is available at [23]. We use this table to discuss issues with specification, publishing and discovery of resources.

Resource Specification. Accurate resource specification ensures faster selection of service providers. However, this is difficult: there are a large number of resources, they can be dynamic e.g., USB devices, and furthermore, they can be expressed at different levels of granularity depending on their characteristics. For instance, CPU requirements can be specified either at a high-level in terms of its architecture class (e.g., SPARC Vs. x86) or at a low level in terms of its L2 cache size. Additionally, they need to be assigned a weight to rate the “quality” of a match and allow for partial matches. For instance,
specification for a service requiring specific s/w can have less stringent-requirements than the h/w.

<table>
<thead>
<tr>
<th>Grid Applications</th>
<th>Grid name</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed Telepresence</td>
<td>NEES Grid</td>
<td>Sensors</td>
</tr>
<tr>
<td>Scientific Data Federation</td>
<td>World-wide telescope</td>
<td>Computers/Detectors</td>
</tr>
<tr>
<td>Medical Data Federation</td>
<td>BIRN Grid</td>
<td>Instrument Resources</td>
</tr>
<tr>
<td>Knowledge Integration</td>
<td>myGrid®</td>
<td>Centralized Servers</td>
</tr>
</tbody>
</table>

Table 1: Example Grids and required resources.

To solve these issues, we classified resources into abstract entities as: physical and logical resources. In addition, we also consider a resource sharing policy that specifies constraints on service use such as allowed users. This allows resource providers to limit the usage of resources depending on their convenience and security. We follow the Common Information Model (CIM)[19] standard to avoid the pitfall of the same resource having multiple names. The following shows the XML resource specification in terms of the consumer (the producer has a similar specification):

```xml
<? XML Version='1.0'>
<Consumer Resource Specification>
  <Physical Resources>
    <Unitary Computer System>
      <Processor> <Family='Intel'/> <Family> </Processor>
      <Operating System> <OSType='LINUX'/> </Operating System>
    </Operating System>
  </Physical Resources>
  <Logical Resources>
    <Database System> <Name> </Name> </Database System>
  </Logical Resources>
</Consumer Resource Specification>
```

**Physical resource specifications** define restrictions on individual/aggregated computing and storage systems. For instance, computing resource abstraction specifies compatibility of the service with the requirement. E.g., a user job with strict sequentially executable sub process modules can only be executed on a single processor system with required memory and processor types. Some jobs may be executed on multi processor systems, since the sub modules are not strictly sequentially executed.

**Logical resource specifications** such as software are needed for accuracy in matching. For instance, a user who wants a desktop system for analyzing a data file requires that the target system be equipped with analysis software. Without logical specification, the matchmaker might choose a service provider who has the required h/w but not the s/w. This necessitates transferring the software system causing several latency and throughput delays. **Resource sharing policy** allows end users to share their resources. It defines which resources are shared, their quantity and how they are shared. This policy will also allow users to set preferences. For instance, a user may specify that she wants to share only 50% of memory. Following 3 types of policies are supported in our model: (a) installation: what can or cannot be installed, (b), usage: controls configuration and selection of specific objects from the system environment, and, (c) security: to authenticate, account and audit clients. Both the sharing policy and resource details are stored on the host computer and are never transmitted for security of the end user. However, partial information in these specifications are used to form the query for searching Grid applications. We extended the CIM[19] model incorporating weighted physical and logical resources.

### 4. P2P GRID AUTONOMOUS SERVICES

**Resource Publishing/Registration/Leasing.** Current Grid approaches use different central service registries for publication and discovery. However, this is unsuitable for autonomic service publication. The intended registries for autonomic services should capture capabilities of services rather than recording simple attributes. The techniques in Teragrid[20] can be used to extend non-capability representation registries to capability representation registries. In our framework, the peers exchange requirement specifications and disseminate the information through the P2P network. This eliminates the need for central services. **Execution of a service** is another key issue. Customers may want to execute service in an environment different from the owner, e.g., with their own security policies. Hence, **services can be leased**. They can be executed in any domain with knowledge of target service. However, **unless a service is autonomous, it cannot support such leasing.** We identified two types of software services: remote and local. Remote services run on a provider’s machine, get input from the user, executes and returns the results. However, in certain scenarios, such as, input data is too large or confidential, this may not be possible. In our architecture the user obtains a copy of the software along with certain owner defined constraints on its execution. Constraints are enforced by a secure mechanism that cannot be circumvented.

**Resource Discovery.** In the current Grid a user specifies a service name and the associated attributes. The central registry then gives access to a matched Grid service file. However, this is inefficient and not scalable. An alternative is semantic matching of services [16], which uses subsumption reasoning. However, its resource description is extended with rich semantics, and hence, matchmaking is expensive. As an alternative, we use feature-based resource matching that uses weighted resource classification...
rather than syntactic/semantic matching. It ranks the requested and available resources according to their degree of closeness. Two resource descriptions are considered matched if their features are similar. The degree of the similarity is computed using Multiple attribute decision making (MADM)[5] which scales the scores into a range [0,1], applies a weight to each similarity aspect (zero is default), computes the sum, and compares it with a user defined threshold $\alpha$.

5. P2P GRID ARCHITECTURE

Figure 1 illustrates the abstract P2P Grid architecture. Our key contribution is the connectivity layer of the standard Grid architecture [6]. The architecture is essentially same as in the Grid except that the collective layer is separated into a management and a service layer for clarity. We believe that P2P based connectivity provides better flexibility in resource searching and matching without compromising on efficiency. Table 2, provides a comparison of P2P, Grid and P2P-Grid. It provides architectural aspects relevant to extend the P2P system to act as a Grid.

6. IMPLEMENTATION AND EXPERIMENTS

We implemented a proof of concept using Phex[21], a Gnutella[23] client written in Java. Our implementation is based on ultrapeers, peers and leaves of standard Gnutella. ultrapeers are connected to other leaves, peers and ultrapeers. They store file information for several “leaf” nodes connected to them. They act like resource aggregators for subdomains, which forward queries to peers/leaves and expect a positive answer.

In our implementation standard search features were assumed so that the ultrapeer first searches its local repository to see if it or its leaves can respond to the incoming query. It then forwards the query to leaves that have high probability of hitting the search. When it cannot generate any response for the query it forwards the same query to other ultrapeers to which it is connected. We also added a new message type RQUERY, to search resources based on an XML specification instead of string based file search. The goals of our experiments were to (a) measure scalability of the system coping with increasing requests, (b) evaluate potential reduction in scalability of the system due to the increasing load of ultrapeers and (c) estimate time complexity of resource searching using our approach. We performed our tests on Intel P4 2.8 GHz, 512 MB RAM running Linux in a LAN. We must note that our experimentation was limited nevertheless, it demonstrates the practical feasibility of our approach. Figure 2&3, shows the results. Briefly: (a) Even with a flood of 50 requests per second there was no apparent performance loss. (b) As no of leaves/peers connected to an ultrapeer

<table>
<thead>
<tr>
<th></th>
<th>P2P</th>
<th>Grid</th>
<th>P2P Grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>General/commercial information/computing</td>
<td>Global problem solving environment</td>
<td>Grid apps. for both scientific and desktop computing</td>
</tr>
<tr>
<td>Role of entities</td>
<td>Peer – both server and client</td>
<td>Grid – server</td>
<td>Service</td>
</tr>
<tr>
<td>Operation</td>
<td>Client-based</td>
<td>Server-based</td>
<td>Service-based operation:</td>
</tr>
<tr>
<td>Participants</td>
<td>Voluntary in/out anytime</td>
<td>Predetermined, registered</td>
<td>Policy-based collaboration</td>
</tr>
<tr>
<td>Reliability</td>
<td>Partially: untrusted peers</td>
<td>Guaranteed trust</td>
<td>Scalable, dynamic, distributed</td>
</tr>
<tr>
<td>Security</td>
<td>Insecure</td>
<td>Secure</td>
<td>Grid layer manages security.</td>
</tr>
<tr>
<td>Control</td>
<td>Limited - not central</td>
<td>Centralized</td>
<td>Policy-based control</td>
</tr>
<tr>
<td>Service Initiation</td>
<td>Client (request)-based</td>
<td>Server(service)-based</td>
<td>Client/Server-Based</td>
</tr>
<tr>
<td>Service Execution</td>
<td>Download resource (e.g file)</td>
<td>Server-side execution</td>
<td>Both</td>
</tr>
<tr>
<td>Connectivity</td>
<td>Occasional. Low capacity</td>
<td>Static high speed</td>
<td>P2P layer adds flexibility</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Grid layer: security, scheduling</td>
</tr>
<tr>
<td>Resource Discovery/Job Management</td>
<td>Limited addition of a new host on the network</td>
<td>Static central registration, in a hierarchical fashion</td>
<td>Peers act as grid service clients. Distributed resource registry/discovery to the Grid services. Grid Job scheduling</td>
</tr>
</tbody>
</table>

Table 2: Comparison of Grid, P2P and P2P Grid

Figure 1. Grid Architecture based on P2P Middleware

or the number of resources shared by the leaves increase, the amount of data stored by the ultrapeers
also increases. We increased the number of clients connected to the ultrapeer the number of resource descriptions stored by them. The results confirm that the increasing time spent for searching is not proportional to the increasing resource requests. Thus the presence of ultrapeers improved the scalability of the system. Note that 2500 is the highest number of resource descriptions to be stored in a host in our computing setting. To get an estimate of the time required for locating appropriate resources, we collected records of time required to answer the same query as the number of hops between the requester and the destination host increased. Our results show that the increase in search time is indeed marginal.

7. CONCLUSION
In this paper, we discussed a P2P Grid architecture. We considered two key issues of using P2P for resource discovery in Grid computing: efficiency of resource specification, publishing and matching and the different P2P search mechanisms. Based on our prototype and evaluations of multiple P2P mechanisms we conclude that P2P over Grid is practically feasible. Performance measurements suggest that the extending P2P with resource discovery does not affect its performance.

8. REFERENCES
5. Z. Fan, J. Ma, An Approach to Multiple Attribute Decision Making Based on Incomplete Information on Alternatives, ICSS– 1999