Karan Bhatia

Peer-To-Peer Requirements On The Open Grid Services Architecture Framework

Status of This Memo

This memo provides information to the Grid community regarding the use of the OGSA framework for Peer-to-Peer applications. Distribution is unlimited.

Abstract

As the next generation of grid computing protocols centered on web services and the Open Grid Services Architecture (OGSA) are developed, the Peer-to-Peer community must determine how these protocols can be used for building Peer-to-Peer applications. Such applicability is not immediately obvious since Peer-to-Peer systems have significantly different properties compared to traditional server-based grid systems. For example, Peer-to-Peer systems exhibit different models of security and trust (machine user is typically also the administrator), have different connectivity characteristics (machine IP addresses may change due to mobility or firewalls/nats), and different usage models (instant messaging, file sharing, and collaboration). Because of these differences, Peer-to-Peer applications are likely to require different capabilities in the infrastructure than server-based grid applications. Yet despite these differences, the shear numbers of desktop systems available today make the potential advantages of interoperability between desktops and servers into a single grid system quite compelling. Hence, the focus of this document is on developing a set of requirements for OGSA to be used to build Peer-to-Peer applications.

Contents

1. Introduction: The need for a new global infrastructure

It is expected that in future ubiquitous networks connections and interactions between devices, systems, services, people, and organizations will be the rule rather than the exception. The realization of the full potential of all conceivable patterns of interaction and collaboration will require a sophisticated global infrastructure on top of which service providers can develop their applications. However, today's infrastructure is rudimentary, making the development of new services both difficult and expensive. A new global infrastructure is needed to handle the sheer complexity of new and varied models of interaction and collaboration.

The scientific and industrial communities both generally recognize this need and have both been developing core infrastructure for meeting this need. The scientific communities have in general adopted the Globus framework [11]: a set of tools to enable the creation and management of *virtual organizations*. The industrial community has adopted an alternative infrastructure developed out of the web server technologies called *web services*[9].

Recently, in the past year and a half, both communities have been working to combine the two infrastructures: the scientific community has been developing and standardizing the *Open Grid Services Infrastructure* (OGSI) [16] on which the Globus Toolkit version 3.0 is based. The OGSI specification builds on top of Web Services and associated standards but extends their definition in a few key places. One of the goals of refactoring the Globus infrastructure into a services-based infrastructure is to leverage industry development tools and runtime infrastructures. It is recognized that as Grid Computing enters the mainstream, scalability, performance, and open development environments are essential to its success.

In parallel to these developments, there has been an explosion of interest in both the academic and industrial communities regarding *Peer-to-Peer* technologies [5, 13]. Roughly speaking, "the idea behind P2P computing is that each *peer* … can act both as a client and as a server *in the context of some application*" [5]. The *peers* are typically desktop computers or devices that interact directly with users. Hence, another phrase that has been used to describe Peer-to-Peer computing is: *computing at the edge of the network*. When peers can communicate with each other, centralized servers are not needed; therefore Peer-to-Peer systems are associated with being more *decentralized*.

Specifically, a *peer* in a P2P system is an endpoint for communication. A peer will run on a device, perhaps a desktop computer, a cellular phone, or a PDA. It exposes one or more interfaces to other peers and provides some service through the interfaces. The service may be to provide a proxy for a human user as in the case of instant messaging, or it may be a proxy for the device on which it is running.

Peer-to-peer systems such as Seti@home [14] and Napster [3] aggregated literally millions of peers together to create a resource that was undeniably unique in its power and capabilities. Other systems soon followed with new applications and architectures: Groove Networks, Entropia, United Devices, Gnutella, etc. On the academic front, interest in Peer-to-Peer computing has also been developing out of a need to increase scalability and support wide-area computing. Technologies such as distributed hash tables, gossip protocols, and network overlays have been developed and continue to be active research topics.

Recently there have been a number of articles suggesting a synergy between Peer-to-Peer and Grid Computing [8, 12, 15]. The arguments are both on the applications and the technology front: peer resources (accessed through Peer-to-Peer applications) could be an important resource within the Grid Computing infrastructure; and Peer-to-Peer technologies could enable larger scale, higher performance Grid systems. Acknowledging the interest in Peer-to-Peer technologies and applications, the Global Grid Forum created a research group called the

OGSAP2P research group that was directed to determine how the nascent OGSI framework could support the development of P2P applications and how these applications could be integrated into a more traditional high-performance computing (HPC) grid computing environment. This document is the result of the work of this research group.

1.1 Prototypical Peer-to-Peer Applications

Broadly speaking, the P2P applications community has been focused on three different application domains: computing, collaboration and file sharing. Computing, also known as "cycle-stealing" or "PC Distributed Computing" utilizes otherwise-idle cycles on desktop and laptop computers for large-scale computation. Condor[1], Entropia [10], United Devices [4], and Data Synapse [2], each are examples of such P2P computing applications. These systems have shown that for some problem domains, the volatility, security and data distribution issues can be resolved such that PC Grids can compete in performance with traditional cluster technology.

P2P collaboration applications allow peers to construct and discover ad-hoc groups, join and leave groups, and perform some shared operations while a member of a group. Groups may be long-lived, composed of friends, colleagues, and organizations or everybody. Groups may also be very short-lived and spontaneous when they are groups of people in the same place at the same time. Sun's JXTA, for example, builds group support into the infrastructure, enabling applications to leverage this group support. Groove Networks is another example: they provide integrated solutions to enable users on desktops or laptops to share and jointly work on documents.

The third class of applications is perhaps a special case of the previous domain: file sharing applications such as Napster, Gnutella, Kazaa among others, enable a large number of users to share content. In some cases the content files are small such as music files or images, and the challenge is to locate any instance of the content quickly among the copies that are available at that instant; in other cases, the files are much larger and Peer-to-Peer technology is used to stream the content to the user from nearby resources.

1.2 This document

The purpose of this document and the OGSAP2P research group is to develop a set of requirements for Grid Computing standards that will enable the construction of Peer-to-Peer applications. These requirements are developed based on the experience of the Peer-to-Peer computing community.

This document is structured as follows: Section 2 describes each of the six areas of work, Section 3 provides a set of use-cases for a few prototypical P2P applications, and Sections 4-10 develop and explain the requirements for each of the six areas. Each requirement is shown as a table indexed by section and category as follows:

4.a	Requirement Category
4.a.1	Requirement 1.
4.a.2	Requirement 2.
4.a.3	Etc.

Sections 4 through 10 describe the various requirements. Finally Section 11 concludes.

2. Areas of work

In developing the set of requirements, we have identified seven general areas where we believe that significant differences between Grid Computing and Peer-to-Peer computing exist:

2.1 Scalability

It's clear that one of the strengths of the Peer-to-Peer systems is their focus on scale: scale in terms of the number of resources, the number of users, and the number of administrative domains covered. When dealing with hundreds of thousands to millions of entities, different design decisions will need to be made. Architectures will favor decentralization, automatic bootstrapping, and best effort as means to provide continued service in a highly dynamic environment.

2.2 Connectivity

The "peers" in Peer-to-Peer systems are typically desktop computers running in a complex network environment. In home networks, Internet service providers may block certain types of network traffic, network address translators may hide resources behind a common name, even the network address of a peer may change quite frequently. Corporate networks are also becoming quite complex and the issues described for the home networks are also challenges in the enterprise. The connectivity requirements explore these issues in greater detail.

2.3 Dynamic and Distributed Discovery.

Discovery services are used in most distributed systems. Sometimes this is referred as lookup service, given that lookups are sent to servers to discover information (e.g. DNS, directory servers, etc.). A Peer-to-Peer system is another type of distributed system, and given its dynamic and decentralized nature, it depends heavily on discovery mechanisms.

2.4 Security

Security on a traditional server-based grid typically assumes strong user identities, secure machines locked away in machine rooms, and trusted and knowledgeable administrators. Peer-to-peer systems, in contrast, have operated in environments where these assumptions are false and have had to develop alternative mechanisms for achieving security, including community-based trust (user ratings) and replication and verification. The differences are not just as functions of the environments, even the goals of the systems are different and in some ways opposite: server-based grids focus on accountability and auditability while some Peer-to-Peer systems focus on anonymity and user privacy. Reconciling these seemingly opposite goals is challenging.

2.5 Resource Availability and Failure Management

The requirements with respect to failures are to some degree a function of the scale and security models of Peer-to-Peer systems. However, we felt that despite the overlap, there are some fundamental architectural differences in dealing with the availability and unavailability of resources that should be explored in its own right. For example, the use of service replicas and

the ability to failover from one replica to another; and consideration of alternative failure models over the simple failstop model. We explore these issues in the failure requirements section below.

2.6 Location Awareness

Location Awareness is the capability to leverage proximity, absolute, relative or contextual, from within the application. There are a variety of reasons why location information could be important for the application, from providing location-based services to system-level optimizations. The former is an important capability in pervasive, ubiquitous computing systems. In addition, mobile phones have recently been required to provide geographic location information for emergency 911 calls, and mobile systems providers are already leveraging this capability to provide prototype location-based services to their users. Examples of location-aware system optimizations are also abundant: consider the Peer-to-Peer content deliver network described in the use cases above. In this application, large files are retrieved from peers that are located "close-by" in terms of network distance. Grid systems built over commodity networks with applications that use large files must also address this problem.

2.7 Group Support

Peer-to-peer collaborative systems allow for the dynamic creation and management of ad-hoc groups. For example, file sharing systems allows a group of families and friends to share photos, music and even calendars. The groups are created and managed easily without requiring permission or authority from any system administrators. Access can be given and revoked purely as a local decision. Such ability goes to the core of collaboration aspects of Peer-to-Peer. We explore these requirements below.

3. Application Use Cases

In order to understand “Peer-to-Peer” applications and their requirements, we provide a set of use cases that cover three of the more common Peer-to-Peer applications: PC grid computing, file sharing, and content delivery. In addition, we provide a fourth use case describing dynamic discovery. While not an end-user application, discovery services are a common service (similar to DNS and domain name resolution) required by most applications.

3.1 PC Grid Computing

The overall objective of PC Grid Computing is to leverage the unused processing power of desktop computers for non-desktop related activities. Such activities may include data services, compute services or some combination. Here, we focus on compute services.

Figure 1 shows the overall architecture of a generalized PC Grid Computing system. There are three main actors:

- *Users* define jobs and decompose a job into a set of tasks that are submitted to the system. Each task description includes all necessary data file, executables (or their locations).
- The *Desktop Clients* receive tasks from the system and manage their execution on the host machine. The client may run in private networks and may be connected to the system manager through NATs and/or firewalls. When the task description is received from the system manager, it may be necessary to download the required files. During execution of the task, the client may actively or passively monitor the execution, providing security and trust guarantees.
- And finally, the *System Manager* maintains resource and capability information regarding the desktop clients, maintains task resource requirements, schedules tasks to clients, and manages the overall fault tolerance and system behavior.

Note that different implementations of such a system may implement the actor’s rolls with one or more software components and that the components may themselves be distributed, replicated or centralized. The functions of the system manager, for example, may be implemented by software running directly on the desktop machines.

The scale of the system may also vary from implementation to implementation, but most commercial systems are designed for thousands to millions of desktop clients and hundreds of users. The number of tasks and duration of each task are such that the resources are kept busy, typically the duration must be much longer than the time to move data from the user to the client.

One can argue that many of these systems technically are not p2p but actually client-server since the Desktop Clients only interact with the System Manager and not with each other directly.

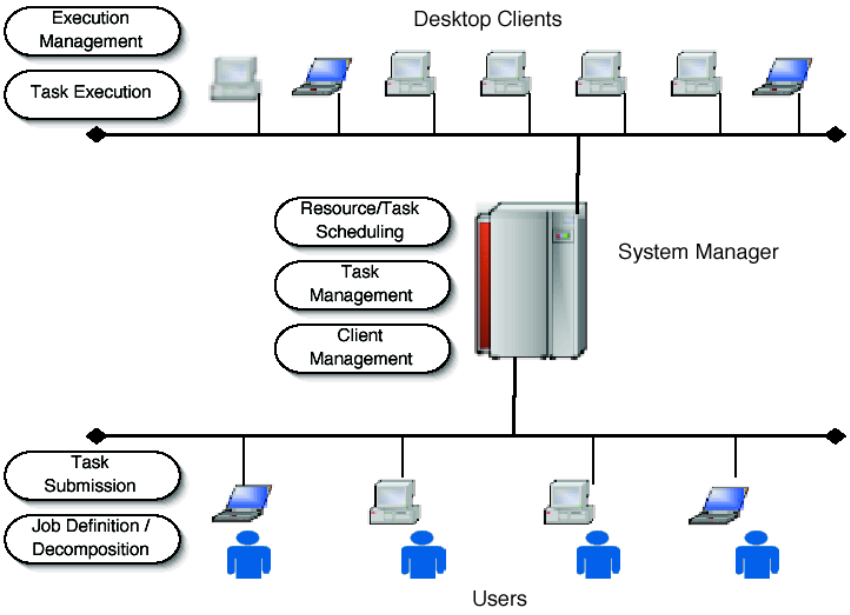

Figure 1: Generalized Architecture of a PC Grid Computing System.

We describe the following use cases:

- User submits Job,
- User Queries the System Manager for Status,
- Desktop Client Joins the System,
- Desktop Client Executes a Task,
- Job Completes and results are returned to the User,
- Desktop Client Leaves the System,
- Desktop Client fails.

3.1.1 User submits Job

A user interfaces to the system through a desktop computer. This computer may or may not function as an execution resource (for the purposes of this document, we assume that it does not). There may be a single user (eg. Seti@home), or a many users (we assume many users).

A job is a computational task that the user wishes to execute. A job is composed of a set of tasks, where each task is a single computational unit. Job decomposition may or may not be part of the PC Grid Computing system, and may range from trivial (eg. parameter sweep style jobs) to quite difficult (database decomposition).

The job is decomposed into a set of task descriptions that includes all the information necessary to execute the task: command executables or locations where they can be found, input parameters, input files or their locations, required output files or their locations, and resource requirements. These task descriptions are sent to the System Manager for scheduling and the System Manager returns a unique job ID for that batch of tasks. The JobID can be used to query the System Manager for the job status.

Note: there may be a firewall or NAT between the User and the System Manager.

3.1.2 User Queries the System Manager for Status

Using the jobid returned by the System Manager for a previously submitted job, the user can query the system to determine the status of the set of tasks. The System Manager returns the job statistics (number of tasks completed, number of tasks currently being executed, number of tasks pending, number of errors, etc).

Note: there may be a firewall or NAT between the User and the System Manager.

3.1.3 Desktop Client Joins the System

When a new Desktop Client joins the system (that is, when the client installs the PC Grid Computing Client software), the software first determines the resource characteristics of the machine: laptop versus desktop, operating system and version, available RAM and disk, processor speed, etc. These resource characteristics are sent to the System Manager that records the characteristics of the machine, its current IP address and generates some unique client ID that is returned to the software.

The client ID is used for all future communications with the System Manager. Note: the client ID cannot be the IP address of the machine, since in this environment, IP addresses may change dynamically. By tagging all communication by a unique client id, the System Manager can identify client machines as their IP addresses change.

After receiving the client ID, the client and System Manager periodically communicate to ensure that the client is available.

Note: there may be a firewall or NAT between the Client and the System Manager.

3.1.4 Desktop Client Executes a Task

When the System Manager schedules a task for execution by allocating the task to an appropriate client. The client, through either a push or a pull mechanism, receives the task description and downloads the necessary files (input files, databases, executables, etc). The client then launches the task typically in some type of sandbox, where the client software can have tight control of the task's execution.

The client software monitors the execution of the task, either through active or passive mechanisms. Passive monitoring simply executes the job and notifies the client software then the job is complete. Active monitoring ensures that the task does not cause any interference with the desktop user and if so, kills or pauses the task. The client software may also employ techniques to protect the application from the desktop user: file encryption and hidden directories are some of those techniques.

The task may need to communicate with other tasks during its execution. For example, when running tightly coupled MPI applications, communication from one desktop client to another may be necessary.

When complete, the a signal is sent to the System Manager that the task completed, and the resulting files (or locations) are copied to their external locations, as described in the task description.

If the desktop client is a laptop, it may disconnect from the network and continue to execute the task. When the task is complete, the client software waits until the network is available, and then sends the notification to the System Manager. When disconnected, communication with other tasks is not possible and the task blocks.

If the desktop client is rebooted, the client software is automatically restarted and the task is either continued from a checkpoint file (if available) or restarted from the beginning. If tasks are communicating, then restarting a task due to failures must be done in such a way to ensure consistency and validity of the results.

Note: there may be a firewall or NAT between the client and the System Manager and between any two clients.

3.1.5 Job Completes and results are returned to the User

When the set of tasks are complete, the result files, and return codes are sent to the user's desktop machine. Future queries of the task show the task as complete.

3.1.6 Desktop Client Leaves the System

A desktop client could leave the PC Grid Computing system for a number of reasons, both expected and unexpected. Expected leaves can occur when the software is uninstalled, or the machine is gracefully halted. Unexpected leaves can occur if the machine crashes or has a hard stop. In the first case, the client software can send a notification to the System Manager that the client machine will no longer be part of the PC Grid Computing system. In the latter case, the periodic communication to or from the System Manager will eventually detect that the machine is no longer responding.

3.1.7 Desktop Client fails

Desktop clients may *fail* for a number of reasons: the desktop user may have turned off the machine, the client machine may crash (desktop machines crash more often than server-class machines, or the desktop user may simply uninstall the software. In all cases the system manager must (either pro-actively or reactively) reschedule the task that was running on that client.

3.2 File Sharing

File Sharing applications such as Napster and Gnutella are still the most popular Peer-to-Peer applications in the Peer-to-Peer space with on the order of millions of users (Napster alone had by some estimates over 57 million individual users). Figure 2 shows a generalized file sharing application. It consists of a number (perhaps millions) of Users and a System Directory.

- A *user* has a computer or device that represents it within the network. It holds files that are "shared" with others and provides the user with means to search and download others' files.
- The *system directory* maintains information about each user/peer, and about each file that is shared in the network. It also provides a mechanism to search the set of files.

The users may belong to a number of different groups, where each group has a unique name. Users have files that they share with other members of groups to which they belong. Groups may be created dynamically by any user who is then identified as the "owner of the group". The owner of the group can control the subsequent membership of that group. All users belong to

the special group “allusers”. The system directory maintains the list of groups, their owners and members, and some metadata of the files that are shared for every user and group. The system directory allows users within a group to search for files of certain type or content. This directory is a logical entity. It may be implemented in either a centralized way with a single group that includes all users (eg. Napster), or in a distributed fashion (eg. Gnutella). We make no assumptions about implementation.

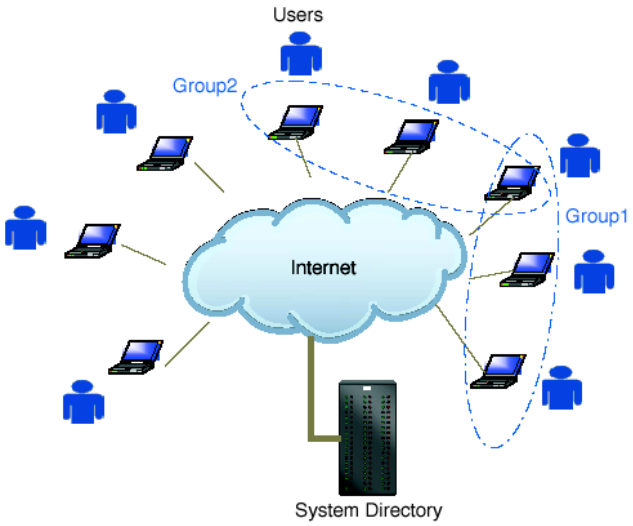

Figure 2: Generalized File Sharing Application

Use cases:

- user joins the network and logs in,
- user creates new group,
- user joins a group,
- user publishes files to a group,
- user searches for and downloads files from a group,
- user leaves the network.

3.2.1 User Joins the Network

The user installs the client program on the computer or device and initiates it. The client program contacts the system directory and registers with it the system capabilities of the device (eg. bandwidth, CPU, etc).

3.2.2 User Creates a New Group

Any user in the system may be able to create a new group. The user runs a client program on his or her desktop computer that connects to the system and authenticates the user. The user can then request a new group to be created with a particular group name. A group is created by the system directory on behalf of the user who is then acknowledged as the “owner of the group”. A group has a group name and a group id. The name is arbitrary, but the group id is unique and assigned by the system directory and returned to the user’s client program. The user is automatically added as the first member of that group.

The group name and id are stored on by the system directory and are indexed so as to enable discovery by other users. Groups may be hidden or visible, and have open or closed enrollments, all defined by the user at the time the group is created. Visible groups can be discovered by other users not in the group, hidden groups can not. Open-enrollment groups allow any user to become a member automatically; closed-enrollment groups require approval from the owner.

3.2.3 User Joins a Group

In order to join a group, a user must either know the group id of the group he/she wishes to join, or search through the directory of visible group stored by the system directory. The user then issues a join request to the system directory.

If the requested group has an open enrollment policy, the user is automatically added. If it is closed, the user is marked as "wanting to join" and awaits approval from the owner of the group. Once given, the user is added to the membership.

Once a member of the group, the shared files that the user has on his desktop computer are indexed and their metadata are sent to the system directory to allow other member of the group to discover them.

3.2.4 User Shares Files Within a Group

Once a user is a member of a group (and note that all users are members of the special group "allusers"), they can search for content (i.e. files) that are shared among all the members of that group.

Using the client program on the user's desktop machine, the user contacts the system directory with the search criteria. The system directory returns to the user's client program a list of member that have a related file and that are currently connected and logged into the system. The user's client program can then contact the "peer members" directly and request a copy of the file of interest.

If the peer member disconnects or logs off, then the user's client program must retry the transfer with a different peer member.

Note that most users are likely to have firewalls or NATs installed between themselves and the system directory.

3.2.5 User Searches for Files Within a Group

Once joined to a group (or set of groups), the user can search for files owned by other users in the group. The user's client program typically provides some query interface to the system directory. The user inputs his search criteria and the client program contacts the system directory that performs the search. The client program returns the results (possibly along with other metadata such as location) to the user.

After selecting the set of files, the user can download them directly from the other peers.

3.2.6 User leaves the system

A user may uninstall or stop the client program. In such a case, the system directory updates its information and removes the files located on that machine from future searches.

3.3 P2P Content Delivery

Peer-to-peer content delivery addresses the problem of delivery of large files from a single source server to multiple client destinations. In general, Content Delivery Networks (CDNs) use

specialized content caches located throughout the network to deliver common content such as large images, software updates, or streaming media. Doing so reduces the bandwidth bottleneck at the originating server and improves performance from the client's perspective.

Peer-to-peer content delivery applications are based on the same idea, but instead of using specialized machines that serve as caches, they use the client's themselves. Figure 3 shows the general idea:

- The *Server* is the publisher of the content.
- The *Client* is the consumer of the content that is published on the server.

When the client on the left requests the same content from the server. Instead of pushing the content through the internet once again for the second client, a Peer-to-Peer content delivery network would redirect the request to the first client who, after all, has an identical copy of content. This redirection is seamless to the second client and the operation completes faster (since the bandwidth within the enterprise is typically larger than the internet) and more cost effective (since no bandwidth is used by the server).

In order to provide this redirection, the server actor must store for each published content C, the set of clients that have a copy of C, the network locations of each of the clients, and the characteristics of each client. When it receives a request for C, the server must redirect the request to an appropriate client.

The clients in this use case are home or work desktop machines. One or both of the clients may be behind NATs or firewalls and may have restrictions on bandwidth (e.g. home user connected over dialup). The server is typically in the public IP space and is a server class machine.

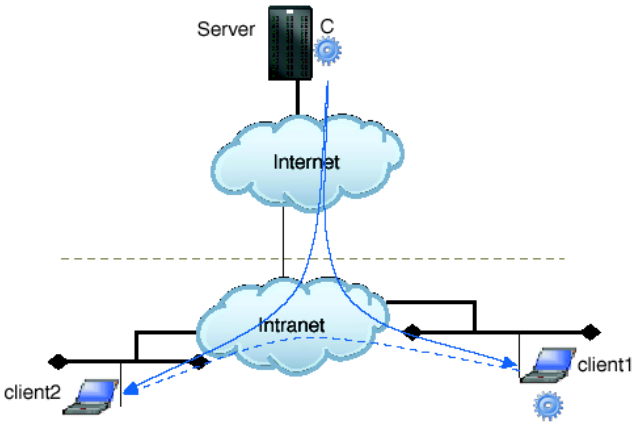

Figure 3: Peer-To-Peer Content Delivery

We consider the following use cases:

- Client retrieves the content file from server (no peer caches are available),
- Client retrieves the content file from a peer cache at client1.

3.3.1 Client Retrieves Content Directly from Server

For this example, the server holds the content file marked C and receives a request for C from client1. Client1 is situated within an intranet, perhaps a corporate network. The content may be a large file such as a streaming media file, or a software update. C has meta-data associated

with it that describes the version information and is not updated by the clients (i.e., the content is read only).

Client1 sends a request to the server for file C. They Peer-to-peer Content Delivery Network intercept that request and determine that there are no peer caches that hold C. Therefore, the request is sent directly to the server who then sends the file to the client.

When C is received by the client, the client software creates a peer cache at the client and stores both the content and metadata associated with the content. The P2PCDN is updated with client1's new cached information.

3.3.2 Client Retrieves Content from Peer Cache

When Client2 requests content C from Server, the P2PCDN again intercepts the requests and determines that C is available from a nearby peer. The content request is redirected to client1 and C is retrieved. Client2 then also becomes a peer cache and the P2PCDN system is updated to reflect this cache for future requests.

3.4 Discovery

A characteristic commonly shared in distributed systems, and in particular in Peer-to-Peer systems, is the ability to dynamically discover other peers in the network, as well as other resources and information. In this particular scenario, although discovery applications could be created, we mainly emphasize the fundamental need for a discovery service that most Peer-to-Peer applications utilize.

As can be seen in Figure 4.a., three physically partitioned groups of peers are shown: the first group consists of peers distributed across the Internet, the second group consists of peers on a local area network that is disconnected from the Internet (e.g. a corporate intranet), and the third group consists of peers on a wireless network that is also disconnected from the Internet. A number of these peers have joined a logical group called Group1. Since the peers are physically disconnected, none of them can see all of the peers in Group1. However, each peer that joins Group1 can discover the members of Group1 within their physical partition.

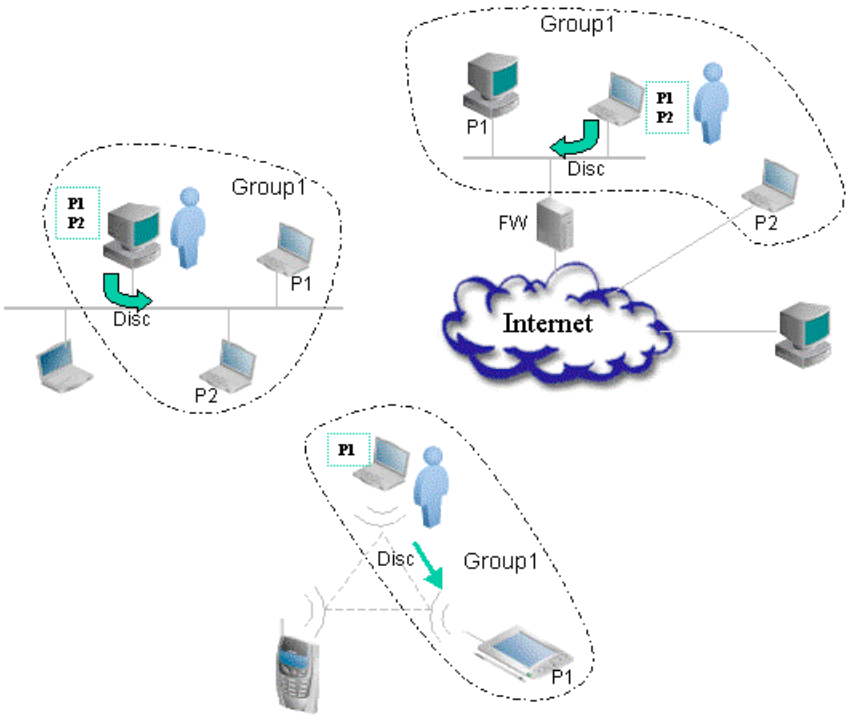

Figure 4: Group1 and its peers in three disconnected group of peers.

We describe the following use cases:

- Peer joins the network.
- Peer discovers peer groups in the network.
- Peer joins a particular peer group.
- Peer discovers other peers in peer group.

3.4.1 Peer joins the network.

A user decides to become a peer in a Peer-to-Peer network. Specific software is run on the user node, and after the corresponding authentication process, the user logs in to the Peer-to-Peer network with a particular peer identity or as an anonymous peer. This peer is now part of the Peer-to-Peer network and can discover other peers and peer groups in this network. In Figure 4, the new peers will be able to discover only those peers that are connected to the specific network.

3.4.2 Peer discovers peer groups in the network.

A peer that have logged in to the Peer-to-Peer network utilizes the discovery service to find the list of available peer groups. After the peer receives a list of existing peer groups, it can select one or more peer groups from the list, and join them. In Figure 4, the peer group that is discovered is Group 1.

3.4.3 Peer joins a particular peer group.

A peer can request to join a particular peer group. Depending if authentication is required for this peer group or not, this peer might need to authenticate when joining a group. In this case, the group to join is Group1.

3.4.4 Peer discovers other peers in peer group.

Once a peer has joined a particular peer group, it could discover other peers that are part of this group, or other peer groups that are subgroups of this group.

In the previous set of use cases, the peers are able to discover peer groups and their peers, as well as discover part of the Peer-to-Peer network topology (at least a portion of it). It is important to note that there is no assumption made on the type of computing device a peer is using, the type of network access that each peer has, or the different security and network devices/appliances that are used on the underlying communication infrastructure. It is also important to note that no initial knowledge about the Peer-to-Peer network, its active peers or the available services is assumed – they could be discovered dynamically.

Figure 5 includes the same group of peers as in the previous figure; however, this time at least one peer of each group is connected to the same network, making all peers part of the same peer group "Group1". In this scenario any peer can discover the different services/resources (i.e. grid services) installed on a particular peer, or on Group1 it self.

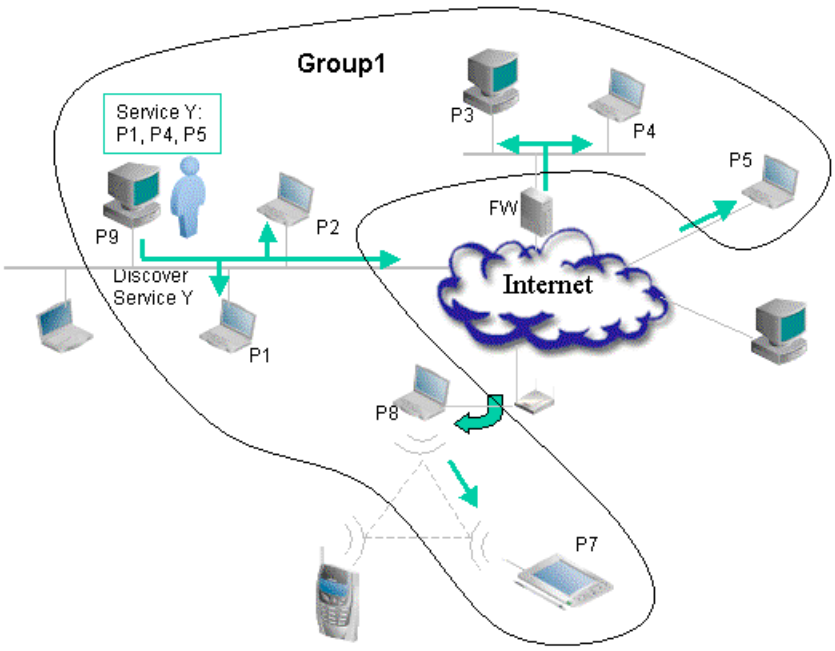

Figure 5: Discovering Service Y in Group1. All peers are virtually connected.

We describe the following use cases:

- Peer sends a query to the network
- Peer may or may not receive a response to the query

3.4.5 Peer sends a discovery query to the network

Peer 9 needs to determine if a particular service Y is available in Group 1. To do so it sends a discovery message that queries the different peers in Group 1. If Service Y is found it should be able to obtain the service characteristics, availability, access policies and so forth.

3.4.6 Peers that receive such query may or may not respond.

As the discovery query propagates through the Peer-to-Peer network, peers that are part of Group1 and know the result of such query could reply with an answer back to the request.

4. Scale

One important distinction between traditional grid computing and Peer-to-Peer computing is in the scale of the resources, both the sheer number of resources and the geographic distribution of those resources. Peer-to-peer applications are scaling to hundreds of thousands if not millions of users (SETI@HOME, instant messaging, etc) spread across the globe. To deal with this scale, P2P systems have successfully employed decentralized architectures (e.g. gnutella) that are capable of surviving in a very dynamic environment. In this section, we explore the different dimensions of scale.

P2P systems and grid systems must be capable of dealing with more-or-less **infinite** set of resources. This may be impractical, but it is very necessary. There is active discussion on and initial implementations of systems whose goal is to implement datasets to persist for over a hundred years. In addition, the grid services community considers each and every file on a particular machine to be a separate resource. Therefore, thousands to millions of files on any modern computer times thousands to millions of computers connected to the internet times hundreds of years is effectively unbounded.

While determining the total number of resources is less than fruitful, there are some tighter bounds and other properties that can be determined.

The number of active resources in a system is roughly 10^12. Certainly, the largest systems that we see today fall well within this bound. The active number of machines in Seti@home is in the hundreds of thousands

The resources exhibit a high *Churn Rate*, that is, as many as 50% of the resources come and go within the system in a 24-hour period. With laptops, PDAs and cell phones, that rate will likely go higher.

4.1 Number of Resources

The belief within the OGSA community is that everything will be a service. This includes applications, files, and instruments. If we consider just the files, there are roughly one million files on any particular machine. If we assume roughly one million machines, then the total number of resources is 10^12.

4.a	Scalability in the number of resources 10^12.
4.a.1	Unique naming of the resources.
4.a.2	Resource Discovery of resources (see resource discovery section).
4.a.3	Accounting and logging resource use.

4.2 Number of Users

Current grid systems have on the order of hundreds of users. As grid technology enters the mainstream, it could have many millions of users. Peer-to-peer systems with millions of users include instant messaging and file sharing applications.

4.b	Scalability in the numbers of users in the system 10^6.
4.b.1	User certificates and issues of trust

4.b.2	Access control

4.3 Number of Organizations

Current grid systems support 10's of organizations, but need to scale in the short term to hundreds of organizations and thousands of organizations in the long term. Currently, establishment of such a "virtual organization" (VO) that spans multiple organizations is difficult and requires significant and ongoing co-operation among systems administrators at each site. For example, Globus user mapfiles must be shared amongst the members of the VO, and grid software installations must be configured to accept certificates from other certificate authorities. Once configured, the trust is complete; there are no gradients of trust based on different organizations. It may be desirable to authenticate and authorize access to services dependent on attributes such as organizational membership and reputation.

In addition to the shear number of organizations in the VO, as many grid systems become operational, a related issue is an organizations membership in multiple VOs. In this case, a single resource may be accessible to users in both VO's. Currently, such partitioning of a resource in multiple grid systems is not possible.

4.c	Scalability in the numbers of virtual organizations the thousands, 10^3
4.c.1	Simple low-overhead mechanisms to allow an organization to join the VO.
4.c.2	Mechanisms to allow a single organization or resource to be a part of multiple VOs.

5. Connectivity

A driving premise behind Peer-to-Peer computing is the access of computational resources, data, and services at network edges in a transparent manner to applications and users while lowering the total cost of ownership and participation. To permit decentralized sharing of computing resources, collaborative workspaces, information and services, it is necessary for the peers at the edge of the network to communicate with one another and with the services at the heart of the network. Therefore ensuring communication is possible in a bidirectional and transparent end-to-end manner is critical to the success of Peer-to-Peer computing over and between public and private networks.

Over the past few years, security, resource availability and cost issues have required the widespread deployment of mechanisms that impede communication between peers on the internet – that is, reduce end-to-end network transparency. Today it is arguable that nearly every user's access to public Internet web services and information must pass through one or more of these devices. These mechanisms, known as Network Address Translators (NATs) and firewalls, often accommodate client-initiated client-server usage models, but they prevent transparent, bi-directional Peer-to-Peer communication.

Furthermore, the edge of the network may have fundamentally different characteristics than traditional server-based environments. For example, data communication over wireless phone networks may have a significantly different cost model that should be considered before sending arbitrary data. Caching and replication policies may also have to be adjusted to take into account different available bandwidth capability.

In this section, we examine the issue of connectivity in greater depth and derive some requirements that need to be met for grid services to be used as foundation for Peer-to-Peer applications.

5.1 Network Address Translators (NATs)

A NAT typically resides at network domain boundaries and translates network addresses from one network domain to another. NATs are most often used to share limited IP public network addresses and/or to translate addresses between two networks with incompatible address domains. Two such domains are the public IP address domain and the private IP address domain. Addresses in the public IP address domain must be assigned through an Internet Assigned Numbers Authority (IANA) and are routable and guaranteed unique from any connected component of the Internet. IANA has also reserved three blocks of IP addresses for private networks. Addresses within these blocks are routable and unique only within the scope of the private network in which it belongs.

Hence, to route packets from one domain to another requires the use of a NAT that has two interfaces, one in each of the address domains that it maps between. Figure 6 shows a common scenario where NATs are used. The ISP allocates one public IP address for use (132.235.95.202). A NAT runs at this address and in addition has an address in a private network (192.168.1.1). When a client in the private network (say 192.168.1.102) initiates a connection to a server in the public network (www.globus.org), those packets are routed through the NAT. The NAT rewrites the source address in the packet headers from 192.168.1.102 to 132.235.95.202 and stores the mapping in memory. When the server receives the connection request packet, it replies to the source address, now 132.235.95.202. The reply packet is received by the NAT that provides a reverse mapping using the in-memory table and rewrites the destination address of the reply packet as 192.168.1.102 that is routed in the private network to the correct client.

Note that because the NAT modifies the packets in route to the destination, security mechanisms used to detect whether packets have been tampered with and that include the header addresses will in general fail. IPsec is one such protocol. It carries a checksum computed over the entire header of the packet. Hence, modifications of the source address will cause the packet to fail the security check.

5.a	No use security protocols that create checksums that include the header.

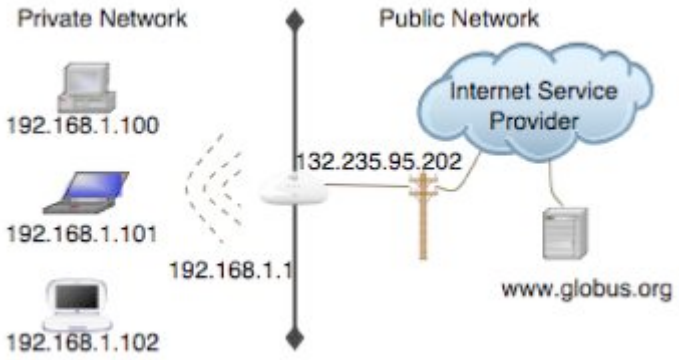

Figure 6: A common setup using a NAT to bridge a private network with a public (ISP) network.

There are many different types of NATs, including a traditional or outbound NAT (described above), network address and port translators (NAPTs), Bi-directional or two-way NATs, and twice NATs. Except for Bi-directional NATs, the basic problem with NATs is that they do not allow connection establishment from the external network into the private network, and hence block bi-directional communication capability.

The use of a NAT imposes several difficulties for Grid Services in particular. In the example in Figure 6, assume that two grid services are running, Grid Service **A** on 192.168.1.102 in the private network and Grid Service **B** running on www.globus.org in the public network. When **A** starts and registers itself in the registry (not shown in the picture but located in the public network) it registers its WSDL and grid service handle. While the NAT has rewritten the TCP packets involved in the registration process, the NAT only modifies the headers while the grid service handle is in the payload of the packet and is not modified. Therefore, when registering with a registry outside of the private domain, grid service **A** may register itself with the incorrect (or inaccessible) handle.

In general, whenever a WSDL document traverses a network domain boundary, its content must be rewritten so that the endpoints remain valid in the external domain.

5.b	A WSDL document that traverses a network domain boundary must be rewritten so that the endpoints remain valid in the external domain: ALG-GS.

This is a common problem for various protocols and there are some existing techniques that can accomplish this. A NAT can be configured with various optional Application Level Gateways (ALGs). Common ALGs include ALG-DNS for domain name information and ALG-FTP for dealing with the FTP protocol.

Even after registration, the external grid client will not be able to initiate communication with the grid service. The NAT requires the communication to be initiated by the host inside the private network. So therefore, grid service **B** acting as a client will not be able to communicate with grid service **A**. Furthermore, grid service **A** will be restricted even as a client: third party transfers (such as with gridftp) and delivery of notifications will not be possible. Consider an example in Figure 7. Grid Service **A** requests a file from grid service **B** that in turn uses a third-party grid service **C** to perform the transfer. In such a case, a traditional outbound NAT considers the communication from C to A as an initial connection request and will not forward the packets to grid service **A**.

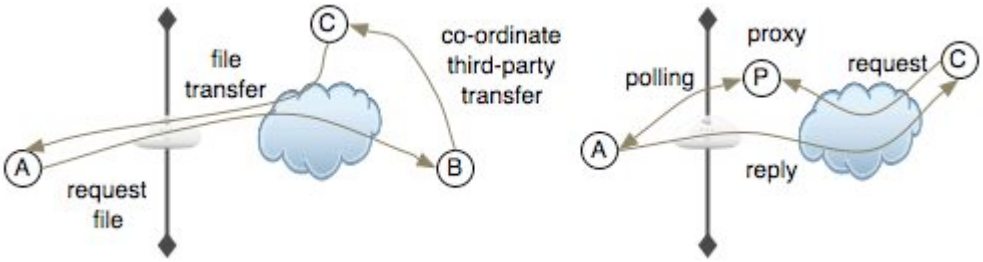

Figure 7: Example of a third-party transfer.

One typical solution to this asymmetry is to use rendezvous servers or simple relay mechanisms and have the host in the private network poll for messages to these intermediary services. The grid services infrastructure should support these solutions as well. Figure 7 shows an example of such a solution. A general grid service proxy runs in the public address domain and acts as a collection point for SOAP messages sent to grid service **A**. Grid service **A** periodically polls the proxy and retrieves the messages and responds as appropriate.

With this solution, the following capabilities are necessary:

5.c	Support for proxy forwarding.
5.c.1	Automatic tooling of grid service A to poll for messages at its proxies
5.c.2	An endpoint proxy should be constructed and store SOAP messages until the grid service can collect the messages.
5.c.3	Grid service clients should be able to tell if the service it is using is behind a NAT.

The use of these intermediaries, however, is neither ideal nor complete. If both the grid service and the grid client are behind NATs, then the store and forward solutions can become complex and unscalable. In addition, multiple levels of NATs are becoming common as ISPs are beginning to use NATs for their domains as well. This is not a problem specific to grid services. This is a general network connectivity problem present for many application frameworks. There exist many draft standards that try to address this issue, fe: Universal Plug and Play (UPnP), MIDCOM - IETF. As solutions and protocols are developed, the OGSA framework should support industry solutions.

5.1.1 Universal Plug and Play (UPnP)

In October 1999, the UPnP forum created the Internet Gateway Devices working group to address this problem for the home/SOHO environments. In essence, the protocols developed

will allow NATs to be detected by clients within the private network, and allow port forwarding to be programmatically configured by applications. Thus UPnP will require application modifications to be effective.

5.d	Support for UPnP detection and configuration.

5.1.2 IETF Middlebox Communication

There are a number of emerging standards for traversing NATs.

5.e	Support for middlebox communication (IETF) solutions to the NAT problem.
5.f	Support for Teredo.
5.g	Support for STUN

5.2 Firewalls

Firewalls are security devices that selectively filter (pass or drop packets) inbound and outbound network traffic based on site rules and policies. Firewalls employ three basic techniques: packet filtering, proxy services and stateful-inspection. A *packet-filtering* firewall examines network traffic passing through it and selectively passes or drops the packets. *Proxy service* firewalls also examine and filter packets but also act as a relay between hosts behind the firewall and the external network domain. Therefore, a proxy service firewall must perform address translation functions similar to that of a NAT. Finally, *stateful-inspection* firewalls monitor communication protocol and application-level state and develop a detailed context that is then used to more carefully inspect the communication protocol stack, enforce security policies, and anticipate protocol communication actions and requirements.

Typical firewall configurations allow http-based packets, that is, packets destined to or from port 80, but may drop packets destined to other ports. *Http-tunneling* refers to the use of http ports for non-http-based applications. Because the firewalls are typically configured to allow such packets through, the applications simply encode the application-specific protocols as http requests and replies.

5.f	Support for HTTP Tunneling.

This requirement may not be very important as system administrators are become more savvy about HTTP Tunneled data. Solutions that work with the needs of the administrators (as opposed to circumventing their precautions) are needed.

5.3 DHCP

Another issue common in Peer-to-Peer environments is lack of fixed IP address assignment. In many environments IP addresses are assigned by DHCP (Dynamic Host Configuration Protocol) servers or may be provided by the NAT directly. When a machine boots up or is connected to the network, the machine broadcasts a IP address request to the DHCP server that assigns a IP address and possibly a resolvable hostname to the machine for a given reservation period. After

the reservation period has expired, the machine will renew its request. The renewed address is not guaranteed to be the same as before.

In Peer-to-Peer environments there is evidence to suggest that this is not a trivial or limited problem. In [6] the authors examination of IP address aliasing finds that

- 40% of hosts use more than 1 IP address in the span of a single day,
- 32% of hosts use 5 or more IP addresses (over the course of the 15 days that they measured).

If addresses are changing on a daily basis, it's not enough to use soft-state mechanism (such as time-out and re-registration). Some mechanisms need to be provided to reroute messages as IP addresses change.

5.g	Support for IP addresses changes.

Supporting IP address changes brings up other issues:

- Should the client be notified of the change and manually renegotiate, re-register, etc? Or should the change be completely transparent?

5.4 IP Address Mobility

IP addresses may change also due to physical movement of the machine from one network to another. This is the case when a laptop user moves from one place to another. In many corporate environments, the use of laptops far outpaces the use of personal desktop machines and therefore must be supported.

The requirement to support mobile systems is the same as in the previous section: no new requirements are needed. It should be noted however, that IP address mobility is not the same as MobileIP which is a particular solution to the problem of mobile users but operates at the IP layer.

5.5 Network characteristics and peer profiles

Peers will be running in all sorts of different computers/devices and it is not known ahead of time what kind of characteristics they would have. They could certainly have smaller resources available or if not smaller, maybe the cost of using them is higher. In the same area, network characteristics could be very different from one peer to another peer. Latency, bandwidth, reliability, cost, are some examples of possible factors that need to be considered when connecting two peers together and before they agree to start exchanging information. For example:

- Two peers that want to communicate have very distinct bandwidth to access the Internet. (Cell phone vs. PC using DSL)
- Some peers will have limited resources (i.e. no storage or very limited, like a PDA)
- Some peers may not even have a hard disk
- For some peers, reception cost might be higher (cell phone receiving a big file)
- Some networks are less reliable
- Some peers need to transfer data faster

These factors should be considered when a peer participates in grid, and it will very likely be related to QoS, SLA and policies.

5.h	Should be possible to determine the characteristics of a peer and its environment, and use it as a basis to establish the communication attributes and quality of the service.

6. Dynamic distributed discovery

Discovery services are used in most distributed systems. Sometimes this is referred as lookup service, given that lookups are sent to servers to discover information (e.g. DNS, directory servers, etc.). A Peer-to-Peer system is another type of distributed system, and given its dynamic and decentralized nature, it depends heavily on discovery mechanisms.

It is important to note that a discovery mechanism can be implemented in different ways. It could be completely distributed - where each peer utilizes discovery to find peers and resources with zero dependencies on servers; distributed with supernodes - where each peer can do its own discovery but could rely on supernodes; use a proxy - where a peer depends on a proxy to do the discovery; centralized - where a peer depends on a central server to discover peers and resources, and store this information.

Peer-to-peer systems have very particular characteristics when it comes to connectivity, availability, and scalability (as described in the use cases and in previous sections of this document), and some of the derived discovery requirements fall in these areas.

6.1 Connectivity

In Peer-to-Peer systems, network partitions or partial disconnects are frequent, and this leads to the existence of islands of peers that are isolated for a period of time. In some cases, disconnects are sometimes short, and some other times long, sometimes unplanned, and sometimes planned. This type of connectivity imposes specific requirements in terms of discovery, given that peers are not guaranteed to have access to specific servers (i.e. name servers, directories, etc.) at all times.

Note: this has nothing to do with server's availability.

6.a	Discovery service should function properly in a disconnected network.

6.2 Availability

Lack of availability, sometimes due to the same connectivity characteristics described above, doesn't necessarily mean that a peer crashed or failed. Sometimes it means that a connection path does not exist; some other times means that the peer has gone into maintenance mode and that general access is restricted; and some other times that the peer is overloaded and cannot be available until some resources are freed up. Regardless if a particular peer is completely available, available with restrictions, or not available at all, it is necessary to being able to discover the unavailability reason and status. This availability information will need to be considered by other systems and peers, before making assumptions about the status of a particular service that is running on a peer.

6.b	Discovery service should be able to determine the status and level of availability of peers and resources.

6.3 Scalability

In the scalability area, given that one characteristic of a Peer-to-Peer system is its potential to scale to millions of peers, it is necessary to have an efficient distributed discovery mechanism that scales. Refer to requirement 4.a.1.

6.4 Initial peer discovery

Discovering a peer is the most basic function in a Peer-to-Peer network. Without this function a Peer-to-Peer network cannot be established. This initial discovery or bootstrap process could be accomplished in many different ways, and could go from being completely static to being completely dynamic. For instance, discovery could be accomplished by having a local list of all the peers in a file, by querying a central directory, or by utilizing a broadcast mechanism.

6.c	Dynamic peer discovery, without utilizing a central server or static mechanism, should be possible.

6.5 Discovering other resources

Once several peers have been discovered and a Peer-to-Peer network exists, it is necessary to have a discovery or search service that allows a peer to discover resources and services that others peers provide and that are willing to share. Given that in a Peer-to-Peer network not every visible peer is directly accessible (i.e. a virtual connection is established through a rely or proxy), it is necessary that the discovery mechanism be able to discover resources on peers to which a direct network connection cannot be established.

6.d	Discovering resources on the network should be possible without being restricted by network topologies, appliances and firewalls.

6.6 Discovering Virtual Organizations

A virtual organization as defined by OGSA maps closely to a peer group. Peer groups will be created and destroyed dynamically, and they could live for a long time or for a short time. Peers could join and leave peer groups often. The concept of peer group or VO is an abstract concept, and it is not necessarily associated with a peer, or with a resources. This brings the issue as to how can it be discovered, given that peer that created it, or the resource where it was initially created might not be up, or may not even exist any more. To cover this case then, it is necessary for the discovery mechanism to be able to discover peer groups or VOs or any other “abstract resource” (if we could define it like that).

6.e	Discovering Virtual Organization (i.e. Peer groups) or other abstract resources should be possible.

6.7 Discovering other peer information

Peers might have specific information that other peers or services might be interested in obtaining or monitor. This could be some kind of peer meta-data, like peer characteristics, version, services, etc. It is not clear yet if a peer can be considered a resource, or if it could be represented as a Grid Service.

6.f	Discovering other peer information should be possible.

7. Security

Peer to peer systems also bring a set of unique security requirements that stem for the fact that Peer-to-Peer systems must deal with looser notions of trust than server-based grid environments. In the server environments, there is an assumption of trust in the actions of the administrators and between the administrative domains. It is assumed that the host certificates are valid and secure; that the users have been validated before user certificates are issued; and that the software functions as advertised. The challenge for building Peer-to-Peer systems is that none of these assumptions hold.

7.1 Trust in identities

In server-based environments, users are assigned identities that are tied to their employment at the organization. The validity of the accounts is (usually) determined by direct interaction and an effort is made to ensure that exactly one account is created for exactly one user. Trust in this setting is given to users because accountability is provided by the system. Malicious users can be traced to their person and can affect their access, their employment, or in some cases even their liberty.

When the system scales beyond the capability to personally verify a user, such as a web-based community, alternative mechanisms for allocating trust have been developed. As an example, consider Ebay. A single user may have multiple accounts and the existence of an account does *not* immediately convey any trust. Instead, *community-based trust* has developed that allows every other user in the system to rate the trustworthiness of every other individual in the system. While a single malicious user may impugn a trustworthy person, the law of large numbers ensures that the averages will bear out the trust. Using this mechanism, trust is earned based on the number of transactions and the quality of the results of the transactions. There is some accountability in these systems (credit card verification), however, in many other communities, the only punishment for malicious users is expulsion from the community. Even this is not guaranteed since the user may just create a new identity.

In Peer-to-Peer environments it may be necessary that community based trust does not depend on a centralized server to manage peers' reputation. It is also important that the absence of a reputation system does not prevent peers from interacting with each other, if trust is not required. However, when trust is required, and it is based on peer's reputation, it is necessary that the reputation system be available all the time, thus making an explicit requirement for reputation being part of the Peer-to-Peer infrastructure.

A reputation system consists mainly of three parts (i) the peer that performs an action and gets rated (2.a.1), (ii) the peers that use some other peer's rating information to decide if they would like to interact with such a peer or not (2.a.2), and (iii) the infrastructure or network used to distribute peer's reputation.

The reputation model depends heavily the infrastructure that is used to distribute the reputation information through the network. If the reputation model is embedded in the communication protocols used to connect peers, then reputation will be part of the infrastructure and easier to include and use in Peer-to-Peer networks (2.a.3).

It is important to distinguish the type of peers that is being rated because; we could be rating devices, applications, or users. For example, we need to distinguish for a particular peer which starts flooding the network, if it is because the device has gone wrong, the application is intended to do that or the user is doing an attack.

If a reputation model for trust is used, it is necessary to have a classification of peers/users and distinguish for what type of behavior the reputation is being built.

Behavior for which peer trust can be built:
- Peer's network usage
- Peer's bandwidth
- Peer's responsiveness
- Peer's quality of responses
- Peer's overall presence
- Peer's reputation as perceived from other peers
- Peer's reputation based on existence
- more...

7.a	Community-based trust
7.a.1	Allow users to rate other users' trust as a function of actions.
7.a.2	Allow users to query the aggregate level of trust as a function of actions.
7.a.3	Support for reputation networks or other type of non-centralized reputation infrastructure.
7.a.4	A peer can be bound for specific behaviors to be tracked and rated, depending on its nature and classification.

Establishment of identities is another area where there may be some unique requirements, although some of these issues are coming to a head even in existing server-based grids. The basic issue is centralized versus decentralized creation of user identities. Centralized mechanisms don't scale as the number of organizations increases and requires the user to manage multiple identities (one for every grid). For example, some grid system includes a centralized certificate authority that provides user certificates to each user of the grid. So in addition to the local organization username and password, the user must maintain the grid username and password. The more grid systems the user is associated with, the more identities he or she must manage. A decentralized approach requires each individual organization that participates in the grid to issue its own user credentials (for example, by running its own certificate authority). The grid resources then are configured to accept certificates signed by each of the participating certificate authorities. This decentralized approach has the advantage of allowing each organization to independently assign user identities as they see fit and is simpler for the end-user. As systems scale in the number of organizations, decentralized identity establishment will become increasingly important.

7.b	Decentralized identity establishment

A third issue related to user identities is anonymity. While access to grid resources requires user verification and authorization, there are some applications where it is important that the users are anonymous. As an example, consider a local government grid and the application of voting. It is key that each user only is able to vote once and only if they are authorized to do so. But it is equally important that the application cannot in any way determine which users issued which votes.

7.c	User identity anonymity
7.c.1	A user should be able to shield his/her identity from the service, if desired, when using the service.
7.c.2	An anonymous peer should be able to use specific services by being member of an identified and trusted peer-group (including “no-group”).

7.2 Trust in resources

The basic notions that trust is a binary property – either the entitiy is trusted or it is not – must be examined carefully in the grid context. As grid systems scale in the number of organizations, it becomes harder to validate the level of trust that users should ascribe to resources. With a few organizations, trust is relatively simple: an agreement is made between the career system administrators at the different organizations as to the security policies that each entity will abide by. The policies are specified in writing and the agreements are legally binding contracts. Each new organization that joins the grid must agree to and abide by these security policies.

In a Peer-to-Peer system where each peer could potentially be in its own administrative domain, such an approach to acquiring trust is a high hurdle to pass. The number of organizations could well be in the thousands or tens of thousands. Furthermore, detecting breaches of the contract (such as malicious users) and enforcing the terms of the contract is unrealistic.

In addition to community-based trust (discussed above), the system should be able to assign different levels of trust to different resources based on the characteristics of the user and resource. Using a distributed computing example, the system may have a high degree of trust in professionally maintained server-class machines running in the same organization as the user; a medium level of trust in machines running at a different organization; and very-little trust in machines running on Internet-based grids. In each case, the system may provide different services: extensive use of virtual machine-based technology, encryption and redundancy for the Internet-based machines; data encryption and validation for machines at different organizations; and no special considerations for server machines running in the same organization.

7.d	Trust based on resource characteristics
7.d.1	Ability to run application in protected virtual environment (maintain resource integrity)
7.d.2	Ability to provide data integrity and confidentiality (encryption and data validation)
7.d.3	Ability to provide code integrity and confidentiality (using encryption, virtual environments and redundancy)

7.3 Trust in Data

An issue that is particularly important for Peer-to-Peer file systems is the issue of copyright identification: the data that is shared between the peers is in some cases copyrighted information and must be detected and copyright policies enforced. Digital rights management (DRM) is one mechanism to control the rights to data, but in some cases, active monitoring and identification of copyrighted material is needed. For example, corporations may have a policy to scan and limit the documents attached to email sent outside of the corporation or internet-based Peer-to-Peer systems may decide not to allow sharing of commercial copyrighted music or videos.

7.e	Support for DRM systems to protect data
7.f	Support for active monitoring and identification of copyrighted material

8. Resilience to Variability in Resource Availability

Resources in a Peer-to-Peer context differ dramatically from server-class resources in terms of their availability – that is to say, desktop resources fail more often and with longer durations than their server counterparts. Part of the explanation is that server-class resources are typically "higher grade" equipment with hardware redundancy built in to avoid downtime. In addition, server-class resources typically have professional administrators who monitor the resources to ensure high availability. Peer-to-peer resources, on the other hand, are composed of lower-end desktop PC's and laptops and are often not professionally managed or updated. And where managed, there may be policies that actually reduce availability: some corporations require desktop PC's to be turned off at night to save electricity and limit security risks.

In addition to simple hardware failures, resources may be *un*available for other reasons: the resource under the control of the desktop user may not wish to provide service or wish to provide a reduced level of service for a period of time. To illustrate this, consider two applications: instant messaging and distributed computing. For the first application, the resource (the user) may be present but in a "busy" state and not accepting incoming messages. In the second application, a resource may offer a reduced level of service during normal business hours.

P2P applications deal with these issues using a variety of mechanism including statistical mechanics (law of large numbers), redundancy and validation, and active feedback and optimization.

8.1 Offline Resource

The server-grid environment assumes that resources are online a majority of the time and that failures and shutdowns are done in controlled circumstances and are performed relatively infrequently. In a Peer-to-Peer setting, offline resources will be quite common as desktop machines are turned off, laptops put to sleep, and other devices become disconnected from the network. In [7], the authors examined machine availability of roughly 51,000 desktop machines and found a 5% daily variation and 10% weekly variation; that is, within the span of a day (or week), 5% (or 10%) of the machines could change their availability status. Entropia [10] found a similar variability, 4% of jobs failed due to machine reboots.

As systems scale up network connectivity also reduces availability. For example, as laptop usage in corporate environments increases, the grid systems must intelligently deal with network disconnections. Some systems deal with network disconnections in the same way as machine shutdowns, but the semantics are quite different. For PC Grids, disconnected machines can still perform significant work and relay the results when reconnected. Hence, it is useful to distinguish between machine failures and network failures (although it may be difficult to do so).

It is also useful to differentiate between a network failure that affects a single machine and network failures that affect a large number of resources. The latter case is exemplified by the failure of a wide-area router or switch cutting off connectivity to entire subnet or domain. In this case, connectivity among the resources in the domain is still functioning but connectivity across domains is not. This has been the basis of intelligent web caching and replication strategies.

8.a	Support resource unavailability
8.a.1	Deal with situations where resources are turned off or rebooted (soft)
8.a.2	Resource crash (hard)
8.a.3	Deal with situations where resource gets disconnected
8.a.4	Deal with situations where resource present, but not accepting desired level of service,

8.2 Byzantine Failures (including malicious users)

Resources may be inaccessible due to malicious users. A common example of this is denial of service attacks that render services inaccessible. This is a difficult area and very related to security.

8.b	Detection and resilience to DoS attacks must be provided

8.3 System Corruption or Failure

The third cause of unavailability of resources is that the grid system itself is unavailable or corrupted. Included in this is the case where metadata regarding the resources (file or service) becomes corrupted. In some cases, this is done purposely by individuals and groups for DoS.

8.c	System Corruption and Failures
8.c.1	Data integrity checks for metadata must be implemented
8.c.2	No single point of failure (e.g. Resource discovery) present in an architecture

9. Location Awareness

Location Awareness is the capability to leverage proximity, absolute, relative or contextual, from within the application. There are a variety of reasons why location information could be important for the application, from providing location-based services to system-level optimizations. The former is an important capability in pervasive, ubiquitous computing systems. In addition, mobile phones have recently been required to provide geographic location information for emergency 911 calls, and mobile systems providers are already leveraging this capability to provide prototype location-based services to their users. Examples of location-aware system optimizations are also abundant: consider the Peer-to-Peer content deliver network described in the use cases above. In this application, large files are retrieved from peers that are located "close-by" in terms of network distance. Grid systems built over commodity networks with applications that use large files must also address this problem.

We have identified three different types of "location" that can be leveraged by applications. Absolute location defines the location of a peer or resource in absolute terms, perhaps latitude and longitude, or with an appropriate network location identifier. Relative location provides distance information between two peers or resources. While it is derivable from the absolute location, it may in fact be easier to provide than absolute location and just as effective for some applications. Contextual location is somewhat less clear. Consider the file sharing application use case described above. If a group is created that is defined as "relatives of mine", then the peers in the group are all related to me. This is an example of contextual location. While the members may in fact be geographically dispersed and in different networks, the membership in the group has some location-like properties.

9.a	Absolute Location
9.a.1	Given a peer, what is its geographic location?
9.a.2	Given a pair of peers, what is their geographic distance?

9.b	Network Location
9.b.1	Given a peer, what is its network location?
9.b.2	Given a pair of peers, what is their network distance?

9.c	Contextual Location
9.c.1	Given a peer, what is its contextual location?
9.c.2	Given a pair of peers, what is their distance?

9.d	Support location-based Queries
9.c.1	Find all resources within specified distance.

10. Group Support

Groups are an essential mechanism to collect and aggregate a set of peer resources or users with common characteristics together. The common characteristics may relate to shared interests of users or specific system configuration of machines. A group must have an identity and must be able to be created and destroyed. Peer resources must be able to discover the groups once created and a mechanism to join a group must be provided.

10.a	A user or resource should be able to create new group
10.a.2	If desired, new groups should be discoverable
10.a.3	Identity of a group is unique

What about deleting a group?

10.b	A user or resource should be able to join an existing group
10.b.1	Joining a group may require explicit permission from owner, or implicit approval through access control mechanisms
10.b.2	A member of a group should be able to query for the group membership
10.b.c	A member of a group may/should be able to vote on group decisions (if appropriate)

10.c	Misc
10.c.1	Delegation of ownership to others
10.c.2	No orphaned group should exist (an orphaned group is one that has no owner)
10.c.3	Support network partitioning

11. Conclusions

This document attempts to provide additional requirements for developing the Open Grid Services Architecture that relate to “Peer-to-Peer” technology. We have provided explanatory use cases for why Peer-to-Peer architectures should be considered as OGSA is developed and have provided a number of specific requirements that support such architectures and applications. The requirements are grouped into six areas:

- scalability
- connectivity
- discovery
- security
- resource availability
- location awareness
- group support

These requirements were developed through the contributions of many people attending the OGSAP2P research group meetings at the Global Grid Forum and the associated group teleconferences.

12. Security Considerations

Although there may be security considerations related to particular Peer-to-Peer applications, this document does not address specific security considerations.

Author Information

The primary authors for this document included:

Karan Bhatia
San Diego Supercomputer Center
UCSD
USA
karan@sdsc.edu

Per Brand
SICS
Sweden
perbrand@sics.se

Sergio Mendiola
Oracle Corporation
USA
Sergio.mendiola@oracle.com

Alex Mallet
Microsoft Corporation
USA
amallet@winse.microsoft.com

Karlo Berket
Lawrence Berkeley National Laboratory
USA
kberket@lbl.gov

Intellectual Property Statement

such proprietary rights by implementers or users of this specification can be obtained from the GGF Secretariat.

The GGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the GGF Executive Director.

Full Copyright Notice

References

1. *Condor: High Throughput Computing*, 2003, The Condor Team.
2. *DataSynapse Home Page*, 2003, DataSynapse.
3. *The Napster Home Page*, 2003, Napster.
4. *United Devices Home Page*, 2003, United Devices.
5. Barkai, D., *Peer-To-Peer Computing: Technologies for Sharing and Collaborating on the Net*. 2001: Intel Press.
6. Bhagwan, R., S. Savage, and G. Voelker. *Understanding Availability*. in *2nd International Workshop on Peer-to-Peer Systems*. February 20-21 2003. Berkeley, CA.
7. Bolosky, W., J. Douceur, D. Ely, and M. Theimer. *Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs*. in *Proceedings of the international conference on Measurement and modeling of computer systems*2000.
8. Casanova, H., *Distributed computing research issues in Grid computing.* Sigact News, 2002. **33**(3): p. 50-70.
9. Cass, S., *Can't we all just get along?* IEEE Spectrum, 2003. **40**(1): p. 47-50.
10. Chien, A., B. Calder, S. Elbert, and K. Bhatia, *Entropia: Architecture and Performance of an Enterprise Desktop Grid System.* Journal of Parallel Distributed Computing, May 2003 2003. **63**(5): p. 597-610.
11. Foster, I. and C. Kesselman. *Globus: a metacomputing infrastructure toolkit*. in *Sage Periodicals Press. International Journal of Supercomputer Applications, vol.11, no.2, Summer 1997, pp.115-28. USA.*
12. Ledlie, J., J. Shneidman, M. Seltzer, and J. Huth. *Scooped, Again*. in *IPTPS*. February 2003. Berkeley, CA.
13. Oram, A., *Peer-to-peer : harnessing the benefits of a disruptive technology*. 1st ed. 2001, Beijing ; Sebastopol, CA: O'Reilly. xv, 432.
14. Sullivan, W.T., D. Werthimer, S. Bowyer, J. Cobb, D. Gedye, and D. Anderson. *A new major SETI project based on Project Serendip data and 100,000 personal computers*. in *the Fifth Intl. Conf. on Bioastronomy*. 1997 1997. Bologna, Italy.
15. Talia, D. and P. Trunfio, *Towards a Synergy Between P2P and Grids.* IEEE Internet Computing, 2003 2003. **7**(4): p. 94-96.
16. Tuecke, S., K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, and D. Snelling, *Open Grid Services Infrastructure (OGSI) Version 1.0*. June 27, 2003 2003, Global Grid Forum.