Globus Grid and Firewalls: Issues and Solutions in a Utility Data Center Environment[1]

Sven Graupner, Carsten Reimann
Internet Systems and Storage Laboratory
HP Laboratories Palo Alto
HPL-2002-278
October 2nd , 2002*

E-mail: {sven.graupner, carsten.reimann}@hp.com

grid computing, firewall, utility data centers, resource management, security, resource virtualization

The "Grid" has become the next incarnation of the utility computing vision: seamless access to data and services anytime, anywhere. Sharing a utility infrastructure requires high standards of security. Grid platforms such as the Globus Toolkit, the Grid Engine, Platform or the forthcoming Open Grid Services Architecture (OGSA) include various security mechanisms from encrypted communication, user authentication and authorization to hosting environments for running Grid applications.

The paper reviews the status of the Globus Grid Toolkit (version 2.0) from a perspective of providing Grid services from behind corporate firewalls to Grid users outside the firewall. Issues are discussed and solutions are presented that have been investigated in a case study running Globus in a Utility Data Center and connecting it to Globus at an outside University crossing a firewall.

The second part of the paper discusses the OGSA notion of a hosting environment providing protected and secure execution of Grid applications and how resource virtualization capabilities of the Utility Data Center can be utilized for this purpose.

* Internal Accession Date Only Approved for External Publication
[1] in cooperation with Chemnitz University of Technology, Germany

Globus Grid and Firewalls: Issues and Solutions in a Utility Data Center Environment[1]

Sven Graupner, Carsten Reimann

Hewlett-Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA 94304, USA

{sven.graupner, carsten.reimann}@hp.com

Abstract

The "Grid" has become the next incarnation of the utility computing vision: seamless access to data and services anytime, anywhere. Sharing a utility infrastructure requires high standards of security. Grid platforms such as the Globus Toolkit, the Grid Engine, Platform or the forthcoming Open Grid Services Architecture (OGSA) include various security mechanisms from encrypted communication, user authentication and authorization to hosting environments for running Grid applications.

The paper reviews the status of the Globus Grid Toolkit (version 2.0) from a perspective of providing Grid services from behind corporate firewalls to Grid users outside the firewall. Issues are discussed and solutions are presented that have been investigated in a case study running Globus in a Utility Data Center and connecting it to Globus at an outside University crossing a firewall.

The second part of the paper discusses the OGSA notion of a hosting environment providing protected and secure execution of Grid applications and how resource virtualization capabilities of the Utility Data Center can be utilized for this purpose.

1 Introduction

The Globus Project [1] is developing fundamental technologies needed to build computational Grids. Grids are persistent environments that enable software applications to integrate instruments, displays, computational and information resources that are managed by diverse organizations in widespread locations. The Globus Toolkit ™ [2] provides a set of tools for building Grid applications. Alternative toolkits include Platform [3] and GridEngine [4]. The Open Grid Services Architecture (OGSA) is the forthcoming specification of the Grid [5].

Globus also defines a Grid Security Infrastructure (GSI) [6], and also OGSA will have a strong emphasis on security. The Globus Toolkit uses the Grid Security Infrastructure (GSI) for enabling secure authentication and communication over an open network. GSI provides security services such as mutual authentication and single sign-on.

The primary motivations behind the GSI are:

- the need for secure communication (authenticated and perhaps confidential) between elements of a computational Grid,
- the need to support security across organizational boundaries, thus prohibiting a centrally-managed security system, and

[1] in cooperation with Chemnitz University of Technology, Germany.

- the need to support "single sign-on" for users of the Grid, including delegation of credentials for computations that involve multiple resources and/or sites.

GSI is based on public key encryption, X.509 certificates, and the Secure Sockets Layer (SSL) communication protocol. Extensions to these standards have been added for single sign-on and delegation. The Globus Toolkit's implementation of the GSI adheres to the Generic Security Service API (GSS-API), which is a standard API for security systems promoted by the Internet Engineering Task Force (IETF).

For offering Grid services behind firewalls, several issues occur:

- How to pass application’s and Globus Toolkit communication through firewalls?
- How to provide secure access to Grid resources behind firewalls?
- How to ensure that resources made accessible to outside Grid users do not allow them to access other resources in a protected environment (sandboxing)?

Addressing the first problem includes all known means for firewall traversal such as using the SOCKS protocol [7], HTTP- or secure shell (SSH) tunneling. The problem here is that all these methods are not transparent to applications or tools using them. Section 4 explains a method how the environment in which applications and Globus Toolkit components operate can be modified transparently such that no modifications in application’s or toolkit’s components are necessary.

The second problem is covered by GSI and the components for authentications and authorization. Users, however, need to be given accounts on Grid resources. Users need to be registered in a local system before they can use resources or services of that system.

The third problem is not easily to address. Typically, resources (such as machines) behind firewalls are part of the internal, protected network. Users given access to those resources can thus access other resources. Preventing users from using other resources than the granted ones requires isolating those resources. Isolation (sandboxing) can be achieved in several ways. Static isolation of machines from the corporate network is one solution (by placing resources in special, de-militarized zones (DMZ)). Then machines cannot be reached from the internal network. Putting machines into Virtual Networks is another solution requiring configuring routers accordingly. Configuring routers is complex and not applicable on a dynamic per-application basis if it is not automated.

Section 6 shows how individual machines can be placed in a programmable virtualized network environment for running multiple Grid applications isolated from each other and isolated from the internal network using capabilities of the Utility Data Center [8].

2 The Globus Toolkit

The Globus Toolkit ™ (we refer to version 2.0 here) provides a set of tools that help to build computational grid applications. It offers a standardized interface to different local resource managing services and supports security in communication and authentication.

2.1 Resource Management

Resource management is one of the central tasks of the Grid and of the Globus Toolkit. The Globus Resource Allocation Manager (GRAM) [9] performs this function located above local resource allocation services. The GRAM has a standardized interface to the underlying local resource management tools (LRMT) such as PBS [10] or Condor [11] forming the lowest level of the Globus resource management architecture. GRAM allows running jobs on a Grid system, and it also provides an API for submitting, monitoring, and terminating computational tasks, also referred to as jobs.

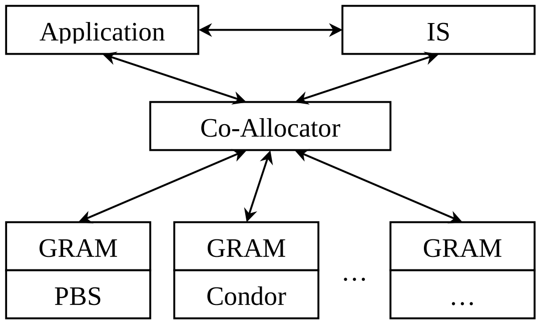


Figure 1: GRAM on top of various LRMT.

The Resource Specification Language (RSL) [10] is used for describing resource needs for executing a job. RSL is also used for exchanging information between components in the Globus resource management. RSL provides the general interchange language within Globus for describing resources. The various components of the Globus resource management architecture manipulate RSL strings in order to perform management functions in cooperation with other components in the system.

Globus may provide a coallocator service, which coordinates a single request that may span multiple GRAMs. One coallocator is the Dynamically-Updated Request Online Coallocator (DUROC) [13] (coallocation is not considered in this report).

2.2 Globus Resource Information Service

The Globus Metacomputing Directory Service (MDS) [14] provides the tools for building an information infrastructure about resources, where resources are provided and how they can be used in a computational grid.

MDS uses LDAP for storing and querying information from a variety of system components. Using LDAP also allows constructing a uniform namespace for resource information across a grid system that may involve many organizations.

The Grid Resource Information Service (GRIS) provides uniform means for querying resources in a computational grid based on their current configuration, capabilities, and status. Such resources include, but are not limited to computation nodes, data storage systems, scientific instruments, network links, or databases.

2.3 Globus Security

The Globus Toolkit uses the Grid Security Infrastructure (GSI) for secure authentication and communication over an open network

The central concept in GSI authentication is the certificate. Each user or service in the Grid is assumed to be identified by a certificate containing all information needed for identification and authentication of a user or a service (users and services are called subjects). Certificates are issued by trusted Certificate Authorities (CA) that are part of the Public Key Infrastructure (PKI) defined by X.509 [15].

A GSI certificate includes four primary parts of information:

- a subject’s name, which identifies the person or service represented by that certificate,
- the public key belonging to the subject,
- the identity of a Certificate Authority (CA) that has signed the certificate to certify that the public key and the identity both belong to the subject, and
- the digital signature of the named CA.

Note that a third party (a CA) is used to certify the association between the public key and the subject represented in the certificate. In order to trust the certificate and its contents, the CA's certificate must be trusted. The association between the CA and its certificate must be established beforehand via some non-cryptographic means. GSI certificates are encoded in the X.509 certificate format, a standard data format for certificates established by the IETF. These certificates can be shared with other public key-based software.

The GSI uses the Secure Sockets Layer (SSL) for its mutual authentication protocol and for encrypted communication. By default, GSI does not establish confidential (encrypted) communication between parties. Once mutual authentication is performed, GSI steps out of the way such that direct communication can occur without the overhead of constant encryption and decryption. The GSI can also easily be used for securely sharing private keys for symmetric encryption.

A related security aspect is communication integrity. Integrity means that an eavesdropper may be able to read communication between two parties but is not able to modify the communication in any way. The GSI provides communication integrity by default. (It can be turned off if desired). Ensuring communication integrity introduces some overhead in communication, but not as significant as needed for full encryption.

3 Globus 2.0 and Firewalls

3.1 Problems with Globus and Firewalls

Organizations such as corporations protect their internal networks or intranets with firewalls from outside networks. Connections from the outside are generally blocked except for communication that uses special protocols and TCP/UDP-ports that are configured for traversing the firewall. Connections from inside the firewall are also usually restricted to few protocols such as SOCKS. Other protocols need to be routed through special proxies such as HTTP.

When a job is submitted to a GRAM, the request is sent to a so-called Gatekeeper, a process that provides access to a set of resources (typically machines). The Gatekeeper handles the job submission and creates a job manager process for each job. The job manager starts and monitors the job, communicating state changes back to the user. When the remote job terminates, normally or by failing, the job manager also terminates.

The Gatekeeper process assumes an open TCP-port (2119). There are no problems when the client and the Gatekeeper processes are running behind the same firewall. The communication is straight as shown in Figure 2. When the Gatekeeper and client are on different sides of the firewall, their communication needs to pass the firewall. If the firewall is not configured for this communication, the communication between these two processes will be blocked.

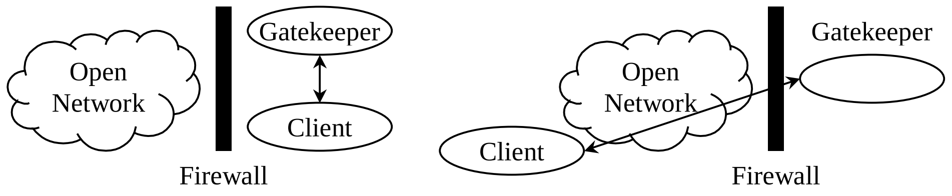


Figure 2: Connecting with the Gatekeeper behind the firewall and through the firewall.

The Grid vision is making resources available across different organizations implying that Gatekeeper and client may also be behind two different firewalls as shown in Figure 3.

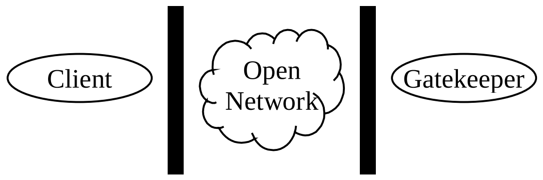


Figure 3: Scenario with two firewalls.

Grid systems also rely on exchanging resource availability and status information through MDS, the Metacomputing Directory Service. Figure 4 shows all processes assumed for a Globus system and the communication links occurring between them through the firewall. For receiving the terminal output, the globusrun command creates a GASS (Global Access to Secondary Storage) server, which receives the output of the job from the job manager at the other host.

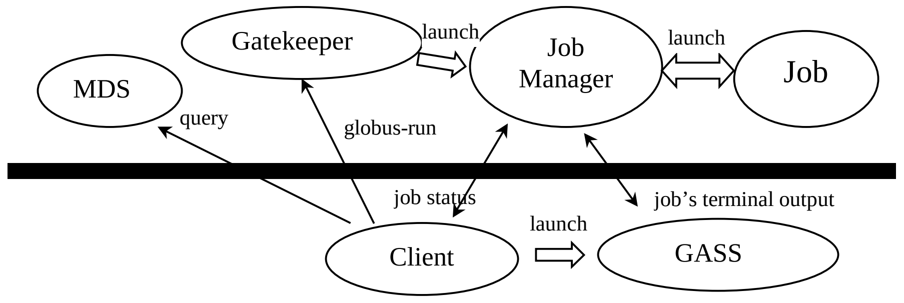


Figure 4: Globus Toolkit processes communicating with a client on two sides of a firewall.

First, we look at the Gatekeeper process. After a job request reaches the Gatekeeper, it will create a job manager process that starts and monitors the job. If the globus-job-run command is used the client will block until the job manager contact the client that the job is finished. If the globus-job-submit command is used, the Gatekeeper returns a contact string which refers to the job manager process and one can communicate directly with the job manager to see, whether the job is done or not. Thus, we need to make the connection available from client to Gatekeeper, the connection from client to job manager and the connection from job manager back to the client.

Second we look at the MDS process. It is the service process, which delivers the resource information about the Globus system it belongs to. As described above MDS uses LDAP for information transfer and is usually listening on TCP port 2135. A query to the MDS process is always initiated by a client process, thus only the communication between the clients to the MDS process needs to be made available.

3.2 Requirements

Given the goals introduced above, following requirements can be summarized:

- allow using Grid services behind corporate firewalls (inside) from outside in a secure fashion, and vice versa, allow using outside Grid services from inside,
- without changing the firewall configuration, and
- without changing any of the Grid components (by building a firewall passing arrangement around Grid components).

3.3 Globus Toolkit Recommendation for crossing Firewalls

The documentation of the Globus Toolkit regarding firewalls [16] describes what needs to be done for making the Gatekeeper available to the outside world of the firewall. The only proposed option is opening the Gatekeeper's port on the firewall and a range of further ports for clients communicating back.

Globus Gatekeeper	MDS	GASS (backward communication)
2119	2135	Globus selects 2 ports per job from GLOBUS_TCP_PORT_RANGE

Table 1: Globus Toolkit communication ports required for connections through the firewall.

In corporations, the network security administration will typically not allow opening ports. Thus, another way of passing through the firewall without changing its configuration is needed under the restriction that components of the Globus Toolkit should not be changed either. Changing Globus Toolkit components is required, for instance, when the SOCKS protocol is used for passing through the firewall.

The Globus recommendation is opening all ports shown in the table on the firewall, an assumption that can hardly be met in corporate environments with strict firewall policies.

3.4 Using SOCKS for crossing Firewalls

For using SOCKS, components must be aware that they need to communicate with the SOCKS proxy server instead of straight with the other partner and need to be modified in order to do so.

When a TCP-based client wishes to establish a connection to an object that is reachable only via a firewall (such determination is left up to the implementation), it must open a TCP connection to the appropriate SOCKS port on the SOCKS server system. The SOCKS service is conventionally located on TCP port 1080. If the connection request succeeds, the client enters a negotiation for the authentication method to be used, authenticates with the chosen method, and then sends a relay request. The SOCKS server evaluates the request, and either establishes the appropriate connection or denies it. The problem with SOCKS is that only connections from inside of the firewall to outside of the firewall are possible, but we need the other direction too.

3.5 Offering Grid Services from Demilitarized Zones (DMZ)

Another solution is setting up the system under the corporation's control, but network-wise in front of the firewall. There are no problems with getting access to this system from outside, but this system will not be protected by the firewall and it might be unsafe. The main problem with this solution is that Grid resources cannot easily be reached from inside the organization since, network-wise, resources are outside the firewall.

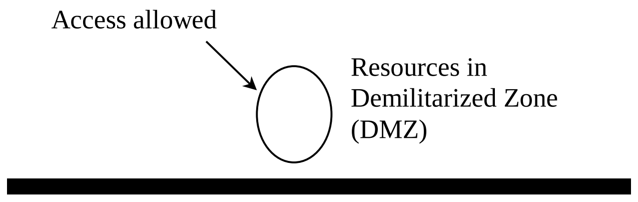


Figure 5: Grid resources offered in demilitarized zone (DMZ).

3.6 HTTP Tunneling

HTTP tunneling is common practice for passing the firewall by using web-proxies. A web-proxy is used when a direct communication to the destination process (HTTP server) is impossible. Processes communicate with the proxy process, and the proxy process forwards communication to the destination passing through the firewall.

This showed the one direction passing through the firewall from an HTTP client inside the firewall to an HTTP server outside. If the HTTP server process is also located behind a firewall, communication is not permitted since web proxies allow communication only in the inside-to-outside direction. This is sufficient for web clients (browsers) accessing web servers in the open Internet. It is insufficient for bidirectional communication through HTTP tunneling. In the case, a web redirecting proxy is typically used (shown in Figure 6) that is outside both firewalls acting as a re-director for incoming HTTP requests from either side. One side must poll the redirector proxy for requests that may have arrived from the other side. Those requests are then passed through the destination firewall as HTTP responses.

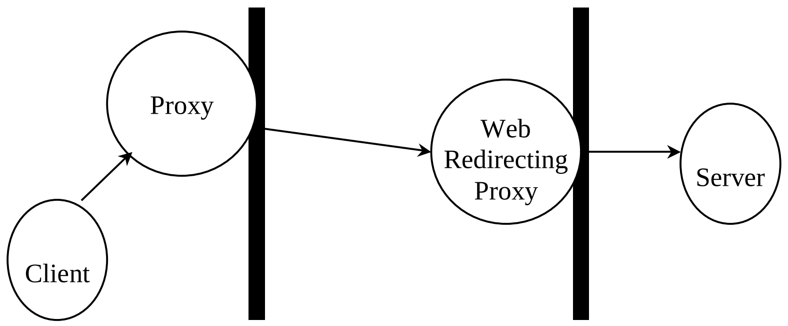


Figure 6: Using HTTP-tunneling for passing through the firewall.

3.7 SMTP Tunneling

A more exotic approach of passing through firewalls is using email protocols such as SMTP (Simple Mail Transfer Protocol). These protocols are typically routed through firewalls. Communicating partners would need to be given email addresses, and email routers would need to be set up forwarding incoming email to communicating partners. This solution might be slow, because communication needs to be translated into email-messages for which no delivery guarantees can be assumed. However, it is a possibility.

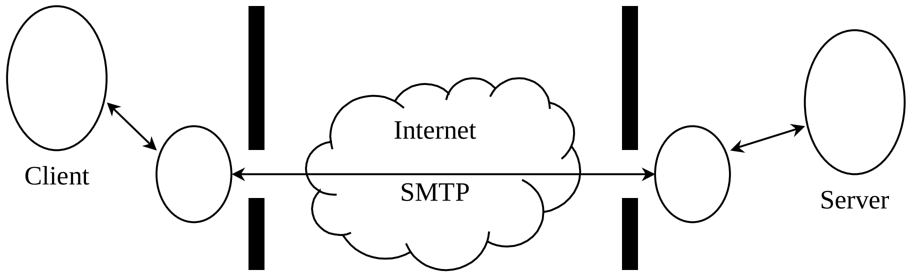


Figure 7: Using a SMTP-bridge for passing through the firewall.

4 The Solution: SSH-Tunneling

The basic idea behind port tunneling is to open an encrypted connection to a remote host and forward all local connections to certain ports to referred ports at the remote host (or vise versa). To enable SSH-tunneling, a SSH-connection to a special host must be initiated. Thus, a host is required at the remote side that allows connections to TCP port 22 coming in and also has a SSH-server process. After a connection has been opened, a command shell will be available for use as normal shell on a computer system (remote shell). If SSH-tunneling is enabled, the SSH process listens on a specified TCP port (connect port) of the client machine. All connections to this port are then forwarded to a specified port TCP port (destination port) on a remote machine. Thus, connecting to a local port opens a connection to a remote host (vise versa is also possible).

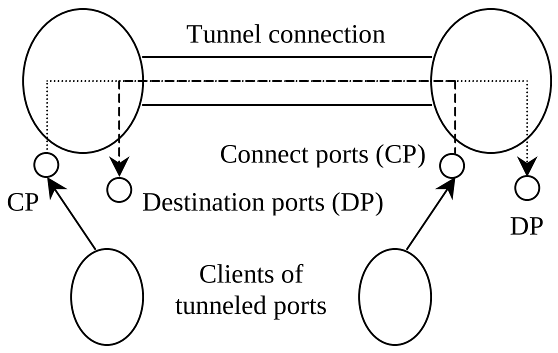


Figure 8: Port tunneling through SSH.

This solution has the advantages that it is easy to realize and no changes are needed in the firewall configuration. Changes in the Globus Toolkit are also not needed making it a reasonable way of passing through the firewall as shown in the experiments section in this report. There are two possibilities for using SSH tunneling:

First, one organization has a firewall-free host to communicate with for establishing the initial SSH connection and for later port tunneling. A communication port from the other organization can directly be tunneled to this host. This is shown in Figure 9.

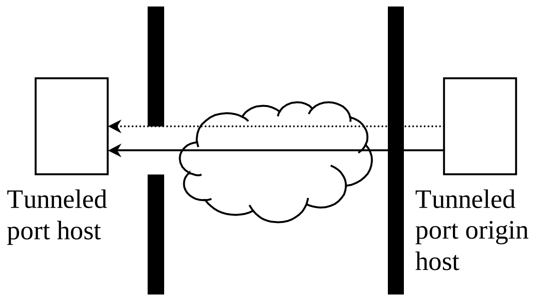


Figure 9: One firewall-free host is available.

Second, both organizations are behind firewalls, and only connections from inside to outside of an organization are available. Here, a third-party host is needed in the middle similarly the HTTP redirector proxy shown in Figure 6, which is accessible from both organizations. Both organization need to connect to this bridge host to open a tunnel. This is shown in Figure 10.

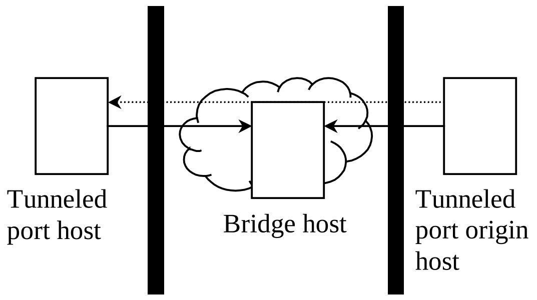


Figure 10: Two firewalls and SSH bridge host in the middle.

Now it is possible to connect to a port on a local host that is tunneled to a remote host of the other organization. Thus, we can access the Gatekeeper of an organization behind a firewall from outside by tunneling its ports to the outside party.

5 Experiments: Connecting two Clusters through a Firewall

With a partner university in Germany, Chemnitz University of Technology, it had been planned to connect a cluster system installed in the Utility Data Center at HP Labs in Palo Alto with a cluster system located at this university using the Globus Toolkit 2.0 software. The goal of the experiments was to show that both systems can be connected under the requirements listed in section 3.2. It should be possible to submit jobs from Palo Alto to Chemnitz University and vise versa. The Globus host at the University site was in front of the firewall. The Globus host is the machine where the central Gatekeeper process for this cluster runs. Thus, the first alternative (see Figure 9) could be used.

For submitting jobs from the Chemnitz University to Palo Alto, the Gatekeeper port (TCP/2119) had to be tunneled using SSH (see section 4) to the Globus host in Chemnitz: The hostname of the Globus host in Palo Alto was “udc-server1.hpl.hp.com” and the hostname of the Globus host at the university site was “john.informatik.tu-chemnitz.de”. In Palo Alto, a SOCKS proxy server was used in order to open a SSH connection in the outside direction to the university site. For opening a SSH tunnel connection, the following command can be used:

```
ssh –l <user> -R <tunnel port>:<origin hostname>:<origin port> <tunnel host>
```

In the specific case:

```
ssh –l cre –R 7001:udc-server1.hpl.hp.com:2119 john.informatik.tu-chemnitz.de
```

Now we can open a TCP connection on “john.informatik.tu-chemnitz.de” to “localhost:7001” on that machine (e.g., telnet localhost 7001). Since this port has previously been tunneled to the Palo Alto site, communication will actually occur with “udc-server1.hpl.hp.com:2119” mediated through the established tunnel.

This enabled submitting jobs from Chemnitz University to Palo Alto by using the globusrun command:

```
globusrun –r "localhost:7001:/O=Grid/…/CN=…" –b –f <rsl-file>
```

However, it is still not possible receiving any output from the job in the user’s terminal yet, and it is also not possible yet connecting to the job manager that controls and monitors the job. The situation is shown in Figure 11. Since the job manager is receiving all communication from the actual job, the job manager’s console (stdin,stdout,stderr) would need to be mirrored back to the client through the tunnel. The following shows how this is achieved.

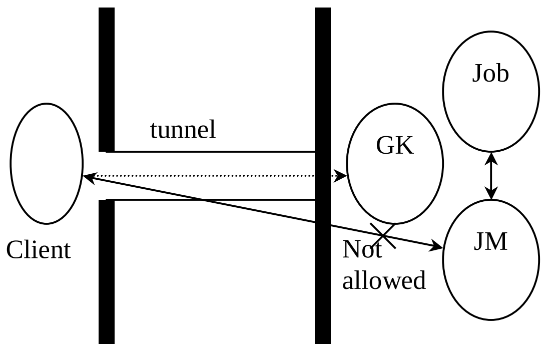


Figure 11: Tunneling the Gatekeeper port.

The following command can be used for submitting a job through the tunnel (from Chemnitz to Palo Alto, "/O=Grid/O=Globus/CN=hpl1-019-12-eth0.hpl.hp.com" is the subject string of the udc-server1):

```
globusrun –r "localhost:7001:/O=Grid/O=Globus/CN=hpl1-019-12-eth0.hpl.hp.com"
          -b –f <file>
```

After this command, the URL of the job manager is returned (e.g., https://hpl1-019-12-eth0.hpl.hp.com:53162/15609/1030555624/). This URL is typically used by the client for retrieving status information about the job from the job manager that is behind the firewall from the client's perspective. Its port needs to be tunneled as well. A problem is that the job manager's port is dynamically chosen. To address this, the range of used ports needs to be controlled by setting the GLOBUS_TCP_PORT_RANGE environment variable (see Globus Toolkit documentation [2]). Then, a range of tunnels needs to be opened from the client's host to the host with the job manager in the other organization. The job manager's URLs need to be translated into the corresponding URLs on the client's machine such as

```
https://hpl1-019-12-eth0.hpl.hp.com:53162/15609/1030555624/
```

needs to be translated into

```
https://localhost:8000/15609/1030555624/.
```

For example, the port range 9000-9020 can be used for tunneling to client's ports 8000-8020.

In order to receive the terminal outputs of the job, additional steps need to be taken. The job manager attempts to contact the client by delivering the output data after the job is done. This communication also needs to be tunneled.

For receiving the terminal output, the globusrun command creates a GASS (Global Access to Secondary Storage) server, which receives the output of the job. The job manager will not be able to contact this GASS server using the URL provided by the client having the original hostname in it and not the "localhost" needed to connect to the port tunneled to the job manager's machine. The solution is creating a separate GASS on the client's machine listening on a special port (in this case 8003) and tunneling this port over to the job manager's host such that the tunneled GASS port could be used by the job manager for reporting output. In order to avoid the mentioned URL conflict, the job manager is told to use the tunneled port for output delivery explicitly by redirecting its terminal output to the desired GASS port that has been tunneled to the job manager's host. This redirection can be done by passing a RSL description with the command that launches the job manager. Figure 12 shows a RSL example how the job manager is told to redirect the job's terminal output to "localhost:7000". SSH will then tunnel this connection to the actual GASS port on the client host, and output will appear in the terminal window from where the GASS process was started (at the client host).

```
&("rsl_substitution" = ("GLOBUSRUN_GASS_URL" "https://localhost:7000" ) )
("stderr" = $("GLOBUSRUN_GASS_URL") # "/dev/stderr" )
("stdout" = $("GLOBUSRUN_GASS_URL") # "/dev/stdout" )
```

Figure 12: RSL example for redirecting a remote job manager's output to a local GASS.

A similar situation occurs with MDS when clients want to query the MDS of a remote system. MDS uses TCP ports that also can be tunneled from a remote host to the client's host. Queries to the MDS process can be made using the grid-info-search command:

```
grid-info-search –h localhost –p <tunnel-port> -x <query>
```

The following figures show all tunneled connections needed for a symmetric connection between Palo Alto and to Chemnitz University with both sides having the same conditions for accessing the Globus system of either side. The bridge ports are not needed if the SSH-client host is the same as the Globus host (usually the terminal runs on a different machine than the Globus host machine running the Gatekeeper).

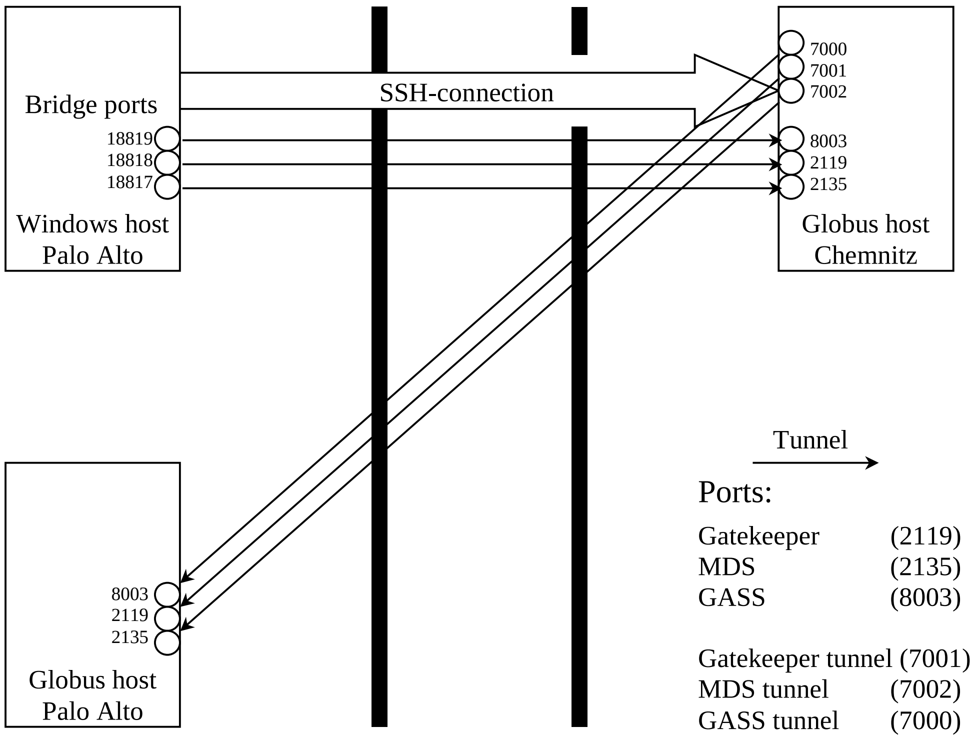


Figure 13: Step 1: establish SSH tunnels from the terminal host.

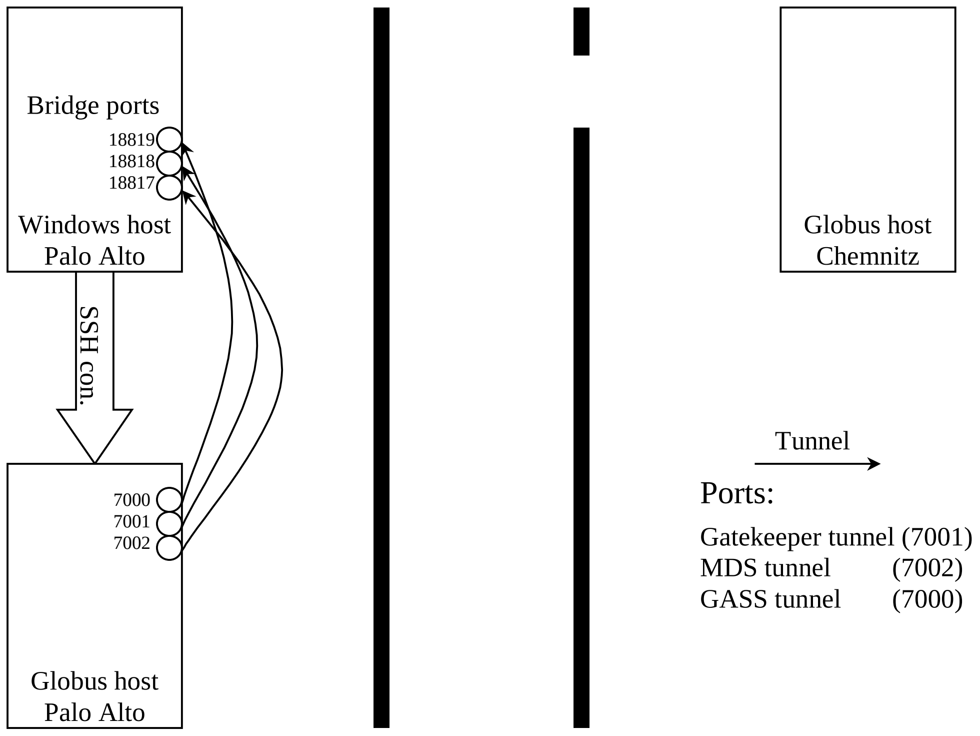


Figure 14: Step 2: establishing tunnels between the Globus host and the terminal host and logging into the Globus host for issuing Globus commands.

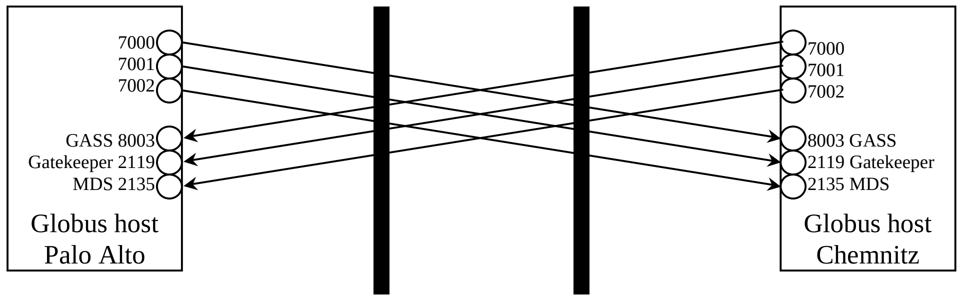


Figure 15: Effective tunnels.

The description of how these tunnels are used is shown in the Appendix.

Limitations experienced in the experimental setting

Because of problems with the Linux runsocks script, we had no possibility to tunnel connections in the other direction (for submitting jobs from Palo Alto to Chemnitz) as desired for providing symmetric conditions for each party. The Windows SSH-client [17] was used instead for opening the initial connection and tunneling ports that did not show the effect that occurred with Linux runsocks.

Another limitation was that for querying MDS only anonymous authentication (unauthorized, public) queries worked. All attempts for authorized queries resulted in an error message “ldap_sasl_interactive_bind_s: Unknown authentication method” in the experimental setting.

If the clock on both hosts is not synchronized well, there might be an authentication problem, when a job is submitted. The problem is that the certificate may not be valid yet since the Gatekeeper’s clock is behind the client’s clock. After waiting until the Gatekeeper’s clock has reached the time of the client’s host, the job submission succeeded. This is a general problem of coordinating distributed clocks, and Globus makes the assumption in its authentication system that time is globally synchronized.

6 Utility Data Center Security Capabilities

Of the problems with Grid services behind firewalls is that users accessing granted resources (machines) potentially could obtain unauthorized access to other machines of the network behind the firewall. In order to prevent this situation, granted machines need to be network-wise isolated by placing them in a private network zone. Boundaries of such a zone are defined by the set of machines granted to a user. Access to the outside world is achieved by routers that are not under the control of the user and are configured when the private network zone is established. With machines being in such a private network zone, users would not be able to access any other machine than the granted ones. Private network zones can be established in multiple ways. This section shows how machines can be offered as Grid resources to Grids applications inside or outside the firewall by placing them in a programmable virtualized network environment provided by the Utility Data Center [8].

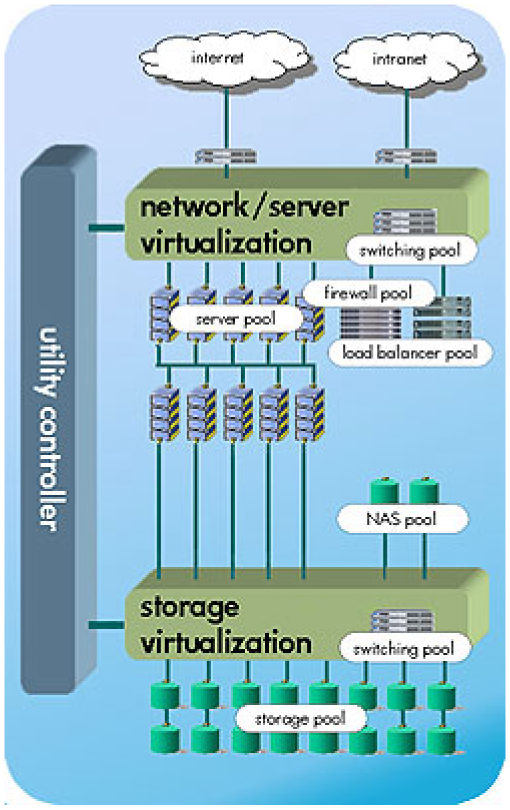
Machines and other resources can be assigned to users from outside the firewall with simultaneously configuring them in one (or multiple) private network zones using the virtualized network environment the Utility Data Center provides. User than are prevented from accessing any other than the granted resources. Programmable routers are under the control of the Utility Data Center. Routers also establish network connections of the user’s private network zone to the outside.

6.1 Resource Virtualization

The basic mechanism the Utility Data Center provides is resource virtualization including storage and network resources. The original motivation for developing a Utility Data Center platform was that deployment and operational costs dominate the balance sheets of enterprise IT customers. Platforms and management solutions are emerging reducing service deployment times and operational costs. Those platforms support the deployment of services (installation and configuration of software and data), the virtual wiring of machines into application environments independently of the physical wiring in a data center. They allow programmatic rearrangements of applications between machines, the dynamic sizing of service capacities, and the isolation of hosting environments located in the same or different data centers. Private network zones with granted resources being part of them thus could span multiple data centers.

The major characteristic of a Utility Data Center is resource virtualization. The storage virtualization fabric with the storage area network attaches storage elements (disks) to processing elements (machines). The network fabric connects processing elements in a private virtual LAN.

Two types of resources are virtualized:



- network resources: by permitting the programmable rewiring of machines and devices to create a virtual LAN network. Virtual wiring is achieved by programming network switches connecting machines and programmatically connecting or removing machines to or from virtual networks, and
- storage resources: by maintaining all persistent states of applications, file systems, bootable operating system images, application software, etc. as whole disk images in separate storage unites detached from machines. Given the programmability of the storage fabric, storage images can then be attached to machines through a storage area network (SAN) in a programmable fashion making them appear on SCSI interfaces of machines from where machines obtain boot images and further data. Machines are not aware about remoteness of storage.

Figure 16: Utility Data Center (UDC) [8] with the two main components: the fabric for network and the fabric for storage virtualization.

6.2 FML – The Farm Markup Language

The Utility Data Center provides a high-level control interfaces for controlling and programming the LAN or SAN switches by the Farm Markup Language (FML) [18]. FML is capable expressing resources and their “wiring” of an entire application environment using XML syntax. A Farm is a collection of resources needed to run an application together with the “wiring” relationships of those resources. A Farm thus is a representation of an entire application environment, containing and encapsulated from the outside by one or more virtual private LANs.

Figure 17 shows an example of an application environment that consists of a firewall (FW) in front of a scalable web server farm (WS_i). Scalable means that the web servers can be scaled up and down in terms of numbers of machines running a web server instance. For example, the FML code fragment shown in Figure 18 determines the minimal number of machines for running web server instances with 4 (min-servers) and the maximal number with 20 (max-servers). The farm will be initiated with 10 servers running web servers (init-servers).

Two machines in two roles of one running a database and the other a fileserver are shown in a subnet behind the web server tier. The firewall is configured to connect two other machines are in a demilitarized zone (DMZ).

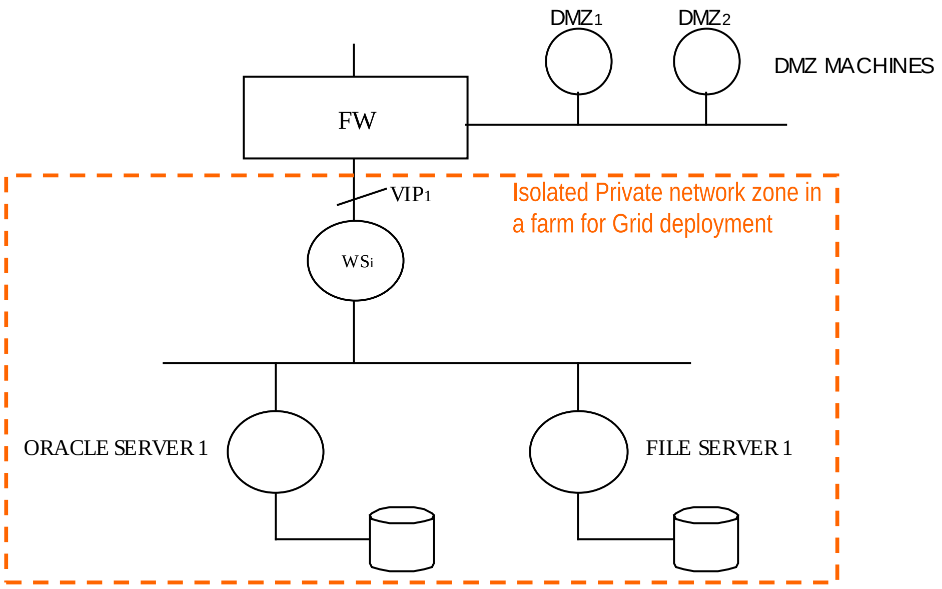


Figure 17: Example of a UDC farm with two virtualized subnets [18].

```
<farm name=”My-3-Tier-Farm”, fmlversion=”1.1”>
<subnet  id=”subnet1”  name=”outer”  ip=”external”  vlan=”outer-vlan”>
</subnet>
<subnet   id=”subnet2”    name=”inner”    ip=”internal”    vlan=”vlan1”>
</subnet>
<subnet   id=”subnet3”    name=”data”    ip=”internal”    vlan=”vlan1”>
</subnet>

<lb id=”lb1” name=”lb1” type=”lb”>
                <interface name=”eth0” subnet=”subnet1” />
                <interface name=”eth1” subnet=”subnet2” />
                <policy> round-robin </policy>
                <vip name=”vip0” subnet=”outer”>
                                <bind id=”bind12” name=”tier1:eth0”
             virtual-port=”8081”
                                 real-port=”8080” />
                </vip>
</lb>

<tier id=”tier1” name=”WebTier”>
                <interface name=”eth0” subnet=”inner” />
                <interface name=”eth1” subnet=”data” />
                <role> role1 </role>
                <min-servers> 4 </min-servers>
                <max-servers> 20 </max-servers>
                <init-servers> 10 </init-servers>
</tier>

<tier id=”tier2” name=”AppTier”>
                <interface name=”eth0” subnet=”subnet2” />
                <interface name=”eth1” subnet=”subnet3” />

                <min-servers> 2 </min-servers>
                <max-servers> 5 </max-servers>
                <init-servers> 3 </init-servers>
                <role> role3 </role3>
</tier>

<tier id=”tier3” name=”DBTier”>
                <interface name=”eth0” subnet=”subnet2” />
                <interface name=”eth1” subnet=”subnet3” />
                <min-servers> 1 </min-servers>
                <max-servers> 1 </max-servers>
                <init-servers> 2 </init-servers>
                <role> role2 </role>
</tier>

</farm>
```

Figure 18: Example FML fragment for the scene shown above [18].

7 Summary

Grid applications are expected to span a multiplicity of underlying hosting environments such as data centers being under the control of several organizations. Those environments are typically protected by firewalls. Hosting environments for Grid applications have to cross firewalls in the underlying infrastructure yet not compromising security. Security requirements for those scenarios are only partially addressed by current Grid toolkits and frameworks. We analyzed one such an environment, the Globus Toolkit (2.0) and identified ways how to enable crossing firewalls with this toolkit. This has been successfully verified in an experimental setup of connecting resources behind the firewall with outside resources. Secondly, we show how network virtualization capabilities of the Utility Data Center can be used for establishing programmable protected hosting environments for applications as a next step after firewall traversal. Protected hosting environments can eventually span resources in multiple underlying data centers.

References

[1] The Globus Project, http://www.globus.org/.

[2] The Globus Toolkit, http://www.globus.org/toolkit.

[3] Platform Inc., http://www.platform.com.

[4] Sun Microsystems, The Sun Grid Engine, http://wwws.sun.com/gridware.

[5] Foster, I., Kesselman, C., Nick, J.M., Tuecke, S., The Physiology of the Grid – An Open Grid Services Architecture for Distributed Systems Integration, DRAFT, http://www.globus.org/research/papers/ogsa.pdf, May 2002.

[6] Grid Security Infrastructure (GSI), http://www.globus.org/security/overview.html.

[7] SOCKS Protocol Version 5, http://www.ietf.org/rfc/rfc1928.txt.

[8] Hewlett-Packard, Utility Data Center, http://www.hp.com/go/hpudc, November 2001.

[9] The Globus Resource Allocation Manager (GRAM), http://www.globus.org/gram.

[10] Open PBS, http://www.openpbs.com.

[11] The Condor Project, http://www.cs.wisc.edu/condor/.

[12] RSL specification, http://www.globus.org/gram/rsl_spec1.html.

[13] The Dynamically-Updated Request Online Coallocator (DUROC), http://www.globus.org/duroc/.

[14] The Monitoring and Discovery Service (MDS), http://www.globus.org/mds/.

[15] Internet X.509 Public Key Infrastructure, http://www.ietf.org/rfc/rfc2587.txt.

[16] Using Globus/GSI with a firewall, http://www.globus.org/security/v1.1/firewalls.html.

[17] Windows SSH client, http://www.ssh.com.

[18] Farm Markup Language (FML) Specification, April 2001.

[19] Graupner, S., Kotov, V., Trinks, H.: Resource-Sharing and Service Deployment in Virtual Data Centers, IEEE Workshop on Resource Sharing in Massively Distributed Systems (ICDCS-2002), July 2, 2002, Vienna, Austria.

[20] Andrzejak, A., Graupner, S., Kotov, V., Trinks, H.: Self-Organizing Control in Planetary-Scale Computing, IEEE International Symposium on Cluster Computing and the Grid (CCGrid), May 21-24, 2002, Berlin.

[21] Kotov, V.: On Virtual Data Centers and Their Operating Environments, HP Labs Technical Report , HPL-2001-44, March 2001.

Appendix

Example Configuration for Connecting Palo Alto to Chemnitz University

Prepare Initial Connection

To create a tunnel that each side can submit jobs to the other side at the same time we need two connections by using the SSH-client for Windows:

Connection 1 (see Figure 13):

Hostname: **`john.informatik.tu-chemnitz.de`** (through firewall)
Outgoing tunnels:

Name	Listen port	Destination host	Destination port
TUC GASS	**18819**	**localhost**	**8003**
TUC Gatekeeper	**18818**	**localhost**	**2119**
TUC MDS	**18817**	**localhost**	**2135**

Incoming tunnels:

Name	Listen port	Destination host	Destination port
HPL GASS	**7000**	**udc-server1**	**8003**
HPL Gatekeeper	**7001**	**udc-server1**	**2119**
HPL MDS	**7002**	**udc-server1**	**2135**

Connection 2 (see Figure 14):

Hostname: **`udc-server1.hpl.hp.com`**
Incoming tunnels:

Name	Listen port	Destination host	Destination port
TUC GASS	**7000**	**localhost**	**18819**
TUC Gatekeeper	**7001**	**localhost**	**18818**
TUC MDS	**7002**	**localhost**	**18817**

Job Submission

The submission of a job is for each side the same. They need to start a GASS-server on port 8003:

```
globus-gass-server –p 8003
```

For receiving the output they need to redirect the standard output to localhost:7000 by using RSL:

```
&("stdout" = "https://localhost:7000/dev/stdout" )
("stderr" = "https://localhost:7000/dev/stderr" )
```

The Gatekeeper port is available on localhost:7001, thus the resource managers address is:

```
"localhost:7001:<subject string of the other side>"
```

Example of job-submission:

```
globusrun –r "localhost:7001:/O=Grid/…/CN=…" –b –f <rsl-file>
```

MDS query

Making a query is also the same for each side:

```
grid-info-search –h localhost –p 7002 –x <query>
```

Example query for getting job queue information:

```
grid-info-search –h localhost –p 7002 –x objectClass=MdsGramJobQueue
```