IMPLEMENTING GRID USING GLOBUS TOOLKIT AND SECURITY CONCERNS

SHIJITH T
EPAGECS050
Department of Computer Science and Engineering
Govt. Engg. College Sreekrishnapuram
Palakkad, India-679517
E-mail:shijitht@gmail.com

Abstract— **Grid computing is the act of sharing task over multiple computers. In it's 3-layer architecture, the middle layer is a third party software which monitors the resources and services the requests of the host. Globus toolkit is such a software.**

As the level of sharing is high in grid networks, security becomes a main issue. Open Grid Form(OGF) is an organization to develop grid standards. Open Grid Security Infrastructure(OGSI) by OGF defines mechanisms for creating, managing, and exchanging information among entities called Grid services. Globus GSI follows OGSI. Here I focus on details, providing a full specification of Globus GSI and it's implementation.

I. INTRODUCTION

Grid computing has emerged as an important new field, distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation. The term the Grid was coined in the mid1990s to denote a proposed distributed computing infrastructure for advanced science and engineering

The real and specific problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations. The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource- brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a virtual organization (VO).

In a grid network, one computer acts as resource broker(RB). The RB monitors resources and requests in the network. It allocates available resources to the needed.The job is split into many and submitted to the RB by the host grid middle-ware. RB assigns the jobs to other available resources and returns the result.

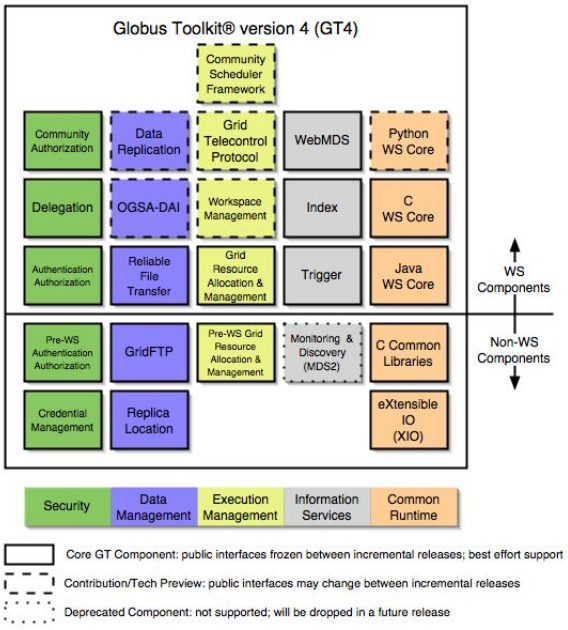
The Globus Toolkit has been designed to use (primarily) existing fabric components, including vendor-supplied protocols and interfaces. However, if a vendor does not provide the necessary Fabric-level behavior, the Globus Toolkit includes the missing functionality. For example, enquiry software is provided for discovering structure and state information for various common resource types, such as computers (e.g., OS version, hardware configuration, load, scheduler queue status), storage systems (e.g., available space), and networks (e.g., current and predicted future load), and for packaging this information in a form that facilitates the implementation of higher-level protocols, specifically at the Resource layer. Resource management, on the other hand, is generally assumed to be the domain of local resource managers. One exception is the General-purpose Architecture for Reservation and Allocation (GARA), which provides a slot manager that can be used to implement advance reservation for resources that do not support this capability. Others have developed enhancements to the Portable Batch System (PBS) and Condor that support advance reservation capabilities.

II. GLOBUS TOOLKIT 4

Globus Toolkit, The Globus project provides open source software toolkit that can be used to build computational grids and grid based applications. It allows sharing of computing power, databases, and other resources securely across corporate, institutional and geographic boundaries without sacrificing local autonomy. The core services, interfaces and protocols in the Globus toolkit allow users to access remote resources seamlessly while simultaneously preserving local control over who can use resources and when. The toolkit includes a lot of components which we can use to program Grid applications.

A. Architecture

The Globus Toolkit 4 is composed of several software components.



As shown in the figure, these components are divided into five categories: Security, Data Management, Execution Management, Information Services, and the Common Runtime. Notice how, despite the fact that GT4 focuses on Web services, the toolkit also includes components which are not implemented on top of Web services. For example, the GridFTP component uses a non-WS protocol which started as an ad hoc Globus protocol, but later became a GGF specification.

B. GT4 Components

- Common Run-time
  The Common Run-time components provide a set of fundamental libraries and tools which are needed to build both WS and non-WS services.
- Security
  Using the Security components, based on the Grid Security Infrastructure (GSI), we can make sure that our communications are secure.
- Data management
  These components will allow us to manage large sets of data in our virtual organization.
- Information services
  The Information Services includes a set of components to discover and monitor resources in a virtual organization.
- Execution management
  Execution Management components deal with the initiation, monitoring, management, scheduling and coordination of executable programs, usually called jobs, in a Grid.

III. GT4 Security

A. What is a secure communication?

Secure communication is not simply any communication where data is encrypted. However, security encompasses much more than simply encrypting and decrypting data.

B. The Three Pillars of a Secure Communication

The three pillars of a secure communication is privacy, integrity, and authentication. Ideally, a secure conversation should feature all three pillars, but this is not always so. Different security scenarios might require different combination of features (e.g. ”only privacy”, ”privacy and integrity, but no authentication”, ”only integrity”, etc.).

- Privacy
  A secure conversation should be private. In other words, only the sender and the receiver should be able to understand the conversation. If someone eavesdrops on the communication, the eavesdropper should be unable to make any sense out of it. This is generally achieved by encryption/decryption algorithms.
  For example, imagine we want to transmit the message ”INVOKE METHOD ADD”, and we want to make sure that, if a third party intercepts that message (e.g. using a network sniffer), they won’t be able to understand that message. We could use a trivial encryption algorithm which simply changes each letter for the next one in the alphabet. The encrypted message would be ”JOWPLFANFUIPEABEE” (let’s suppose ’A’ comes after the whitespace character). Unless the third party knew the encryption algorithm we’re using, the message would sound like complete gibberish. On the other hand, the receiving end would know the decryption algorithm beforehand (change each letter for the previous one in the alphabet) and would therefore be able to understand the message. Of course, this method is trivial, and encryption algorithms nowadays are much more sophisticated.
- Integrity
  A secure communication should ensure the integrity of the transmitted message. This means that the receiving end must be able to know for sure that the message he is receiving is exactly the one that the transmitting end sent him. Take into account that a malicious user could intercept a communication with the intent of modifying its contents, not with the intent of eavesdropping.
  ’Traditional’ encryption algorithms don’t protect against these kind of attacks. For example, consider the simple algorithm we’ve just seen. If a third party used a network sniffer to change the encrypted message to ”JAMJAMJAMJAMJAMJA”, the receiving end would apply the decryption algorithm and think the message is ”I LI LI LI LI LI ”. Although the malicious third party might have no idea what the message contains, he is nonetheless able to modify it (this is relatively easy to do with certain network sniffing tools). This confuses the receiving end, which would think there has been an error in the communication. Public-key encryption algorithms do protect against this kind of attacks (the receiving end has a way of knowing if the message it received is, in fact, the one the transmitting end sent and, therefore, not modified).

- Authentication
  A secure communication should ensure that the parties involved in the communication are who they claim to be. In other words, we should be protected from malicious users who try to impersonate one of the parties in the secure conversation. Again, this is relatively easy to do with some network sniffing tools. However, modern encryption algorithms also protect against this kind of attacks.
  Another important concept in computer security, although not generally considered a 'pillar' of secure communications, is the concept of authorization. Simply put, authorization refers to mechanisms that decide when a user is authorized to perform a certain task. Authorization is related to authentication because we generally need to make sure that a user is who he claims to be (authentication) before we can make a decision on whether he can (or cannot) perform a certain task (authorization).So authentication refers to finding out if someone's identity is authentic (if they really are who they claim to be) and that authorization refers to finding out is someone is authorized to perform a certain task.

IV. Key Concepts

A. Public key cryptography

Public-key algorithms are asymmetric algorithms and, therefore, are based on the use of two different keys, instead of just one. In public-key cryptography, the two keys are called the private key and the public key.
* Private key : This key must be know only by its owner.
* Public key : This key is known to everyone (it is public)
* Relation between both keys: What one key encrypts, the other one decrypts, and vice versa. That means that if you encrypt something with my public key (which you would know, because it's public), I would need my private key to decrypt the message.

B. Certificates and certificate authorities

A digital certificate is a digital document that certifies that a certain public key is owned by a particular user. This document is signed by a third party called the certificate authority (CA).If we trust the CA we can trust the certificate signed by that CA.

C. X.509 certificate format

The format in which digital certificates are encoded: the X.509 certificate format. An X.509 certificate is a plain text file which includes a lot of information in a very specific syntax. The four most important things we can find in an X.509 certificate:

- Subject
  This is the 'name' of the user. It is encoded as a distinguished name.
- Subject's public key
  This includes not only the key itself, but information such as the algorithm used to generate the public key.
- Issuer's Subject
  CA's distinguished name.
- Digital signature
  The certificate includes a digital signature of all the information in the certificate. This digital signature is generated using the CA's private key. To verify the digital signature, we need the CA's public key (which can be found in the CA's certificate).

The certificate is a public document we want to be able to distribute to other users so they can verify our identity, so we don't want to include the private key (which must be known only by the owner of the certificate). When we are in possesion of both a certificate and its associated private key, these two items are generally refered to as the user's credentials.

- Examples of distinguished names
  Names in X.509 certificates are not encoded simply as 'common names', such as "Borja Sotomayor", "Lisa Childers", "Certificate Authority XYZ", or "Systems Administrator". They are encoded as distinguished names, which are a comma-separated list of name-value pairs. For example, the following could be our distinguished names:
  O=University of Chicago, OU=Department of Computer Science, CN=Borja Sotomayor
  O=Argonne National Laboratory, OU=Mathematics and Computer Science Division, CN=Lisa Childers
  A distinguished name can have several different attributes, and the most common are the following:
  O : Organization
  OU : Organizational Unit
  CN : Common Name (generally, the user's name)
  C : Country

V. Grid Security Infrastructure

Security is one of the most important parts of a Grid application. Since a grid implies crossing organizational boundaries, resources are going to be accessed by a lot of different organizations. This poses a lot of challenges. We have to make sure that only certain organizations can access our resources, and that we're 100% sure that those organizations are really who they claim to be. In other words, we have to make sure that everyone in our grid application is properly authenticated.

A. Globus GSI

The Globus Toolkit 4 allows us to overcome the security challenges posed by grid applications through the Grid Security Infrastructure (or GSI). GSI is composed of a set of command-line tools to manage certificates, and a set of Java classes to easily integrate security into our web services. GSI offers programmers the following features,

- Transport-level and message-level security
- Authentication through X.509 digital certificates
- Several authorization schemes
- Credential delegation and single sign-on

- Different levels of security: container, service, and resource
- Transport-level and message-level security
  GSI allows us to enable security at two levels: the transport level or the message level. Transport-level security, then the complete communication (all the information exchanged between the client and the server) would be encrypted. If we use message-level security, then only the content of the SOAP message is encrypted, while the rest of the SOAP message is left unencrypted.
  Both transport-level and message-level security in GSI are based on public-key cryptography and, therefore, can guarantee privacy, integrity, and authentication. However, not all communications need to have those three features all at once. In general, a GSI secure conversation must at least be authenticated. Integrity is usually desirable, but can be disabled. Encryption can also be activated to ensure privacy.
- Authentication
  GSI supports three authentication methods:

- X.509 certificates
  X.509 certificate format is used to provide strong authentication.
- Username and password
  A more rudimentary form of authentication, using usernames and passwords, can also be used. However, when using usernames and password, we will not be able to use features like privacy, integrity, and delegation.
- Anonymous authentication
  We can request that a communication be anonymous, or unauthenticated. Anonymous generally makes sense when we are using more than one security scheme. For example, we can use GSI Secure Conversation (authenticated with X.509 certificates) and anonymous GSI Transport, so that we don't perform an additional (redundant) authentication.
- Authorization

  Although authorization is not one of the 'fundamental pillars' of a secure conversation, it is nonetheless an important part of GSI. Authorization refers to who is authorized to perform a certain task. In a Web services context, we will generally need to know who is authorized to use a certain web service.
  GSI supports authorization in both the server-side and the client-side. Several authorization mechanisms are already included with the toolkit, but we will also be able to implement our own authorization mechanisms.

  – Server-side authorization
    The server has six possible authorization modes. Depending on the authorization mode we choose, the server will decide if it accepts or declines an incoming request.

    * None
      This is the simplest type of authorization. No authorization will be performed.
    * Self
      A client will be allowed to use a service if the client's identity is the same as the service's identity.
    * Gridmap
      A gridmap is a list of 'authorized users' akin to an ACL (Access Control List). When this type of authorization is used, only the users that are listed in the service's gridmap may invoke it.
    * Identity authorization
      A client will be allowed to access a service if the client's identity matches a specified identity. In a sense, this is like having a one-user gridmap (except that identity authorization is configured programmatically, whereas the gridmap is represented as a file in our system).
    * Host authorization
      A client will be allowed to access a service if it presents a host credential that matches a specified hostname. In other words, we will only allow requests coming from one particular host.
    * SAML Callout authorization
      We can delegate the authorization decision to an OGSA Authorization-compliant authorization service. One of the main technologies used in these components is SAML (Security Assertion Markup Language).

  – Client-side authorization

    This allows the client to figure out when it will allow a service to be invoked. This might seem like an odd type of authorization, since authorization is generally seen from the server's perspective. However, in GSI, clients have every right to be picky about the services they can access.

    * None
      No authorization will be performed.
    * Self
      The client will authorize an invocation if the service's identity is the same as the client.
    * Identity authorization
      As described above, the client will only allow requests to be sent to services with a specified identity.
    * Host
      The client will authorize an invocation if the service has a host credential. Furthermore, the client must be able to resolve the address of the host to the hostname specified in the host credential. Note that this is different from server-side host authorization, where we check if the hostname in the credential is equal to a host specified by us.

  – Custom authorization

GSI provides an infrastructure to easily plug in our own authorization mechanisms. For example, our organization might be using a legacy authorization service that can't work out-of-the-box with the authorization methods provided by the toolkit. In this case, we can create a new authorization method that will allow GSI to make authorization decisions based on our organization's legacy service.

- Delegation and single sign-on (proxy certificates)

  Credential delegation and single sign-on are one of the most interesting features of GSI, and are possible due to something called proxy certificates.
  Proxy certificates: Allowing someone to act unconditionally on your behalf is a risky affair. Sure, you might trust them now, for the particular task you want to do, but someone from Organization B might use the proxy certificate in the future to carry out some mischievous deeds on your behalf. Therefore, the lifetime of the certificate is usually very limited (for example, to 12 hours). This means that, if the proxy certificate is compromised, the attacker won't be able to make much use of it. Furthermore, proxy certificates extend ordinary X.509 certificates with extra security features to limit their functionality even more (for example, by specifying that a proxy certificate can only be used for certain tasks). A proxy certificate allows a user to act on another user's behalf. This is more properly called credential delegation, since proxy certificates allow a user to effectively delegate a set of credentials (the user's identity) to another user. This solves the problem originally posed, since B could use a proxy certificate (signed by A, of course) to prove that it is acting on A's behalf. Organization C would then accept B's request.
  By using proxy certificates we also get another desirable feature: single sign-on. Without proxy certificates, Organization A would have to authenticate itself with all the organizations that receive requests 'on behalf of A'. In practice, this mean that the user in Organization A with permission to read the private key would have to access the key each time a mutual authentication is needed. Since private keys are usually protected by a password, this means that the user would have to sign on (provide the password) to access the key and perform authentication. Using proxy certificates, the user only has to sign in once to create the proxy certificate. The proxy certificate is then used for all subsequent authentications.
- Container, service, and resource security

  Finally, it should be noted that many of the features described can be specified at three levels: container, service, and resource level. Of special interest is the fact that we can configure security at the resource level. For example, we can set different authorization mechanisms for a service and its resources, so that stateless operations can be performed without authorization, but stateful operations do require an authorized user.

VI. Setting up the machine

A. Pre-requisites

- Fedora release 11
- OpenSSL v 0.9.8
- Sun-jdk v 1.5
- Apache-ant v 1.6.5
- GCC v 4.4.1
- GCC-C++
- Perl-XML-Parser
- Perl-ExtUtils
- Xinetd

B. Building the Toolkit

Created a user named globus, and start the installation as globus. Setup ANT_HOME and JAVA_HOME.Create a .bashrc like this

```
JAVA_HOME=/usr/local/jdk1.5.0_20
ANT_HOME=/usr/local/apache-ant-1.7.1
GLOBUS_LOCATION=/home/globus/globus-4.2.1
PATH=$JAVA_HOME/bin:$PATH:$ANT_HOME/bin:$GLOBUS_LOCATI
LD_LIBRARY_PATH=$GLOBUS_LOCATION/lib:$LD_LIBRARY_PATH
GRID_SECURITY_DIR=/etc/grid-security
GRIDMAP=/etc/grid-security/gridmapfile
GLOBUS_HOSTNAME=‘/bin/hostname‘
CLASSPATH=$JAVA_HOME/lib/tools.jar:$CLASSPATH
export PATH JAVA_HOME CLASSPATH GLOBUS_LOCATION GRID_SECURITY_DIR GRIDMAP GLOBUS_HOSTNAME ANT_HOME LD_LIBRARY_PATH
```

As globus do,

```
$ tar -xvzf gt4.2.1-all-source-installer.tar.gz
$ cd gt4.2.1-all-source-installer
$ ./configure –prefix=/home/globus/globus-4.2.1/
```

This creates a make file.

```
$ make
$ make install
..Done
```

C. Setting up security on your machine

- SimpleCA
  All of the work we're going to do now requires that we be authenticated and authorized. We use certificates for this purpose. The Distinguished Name (DN) of a certificate will serve as our authenticated identity. That identity will then be authorized. In this the authorization will happen in a file lookup.
  We will need identites for both the services and users. For the services, we will use an identity that is equal to their hostname. For the users, we'll use their full name. To create the certificates, we're going to use the

SimpleCA that is distributed with the toolkit. Here's how we set it up.
As root do,
cd /home/globus/gt4.2.1-all-source-installer
mkdir /etc/grid-security
perl gt-server-ca.pl -y
This is to setup simpleCA and the host certificate.
Now we copy that signed certificate into /etc
mv $GLOBUS_LOCATION/etc/host*.pem /etc/grid-security/
We'll make the containercerts owned by globus:
cd /etc/grid-security
cp hostcert.pem containercert.pem
cp hostkey.pem containerkey.pem
chown globus:globus container*.pem

- Creating a MyProxy server
  We are going to create a MyProxy server. This will be used to store our user's certificates. Recall that so far we have made a host certificate, but we don't have any certificates for end users yet.
  # export GLOBUS_LOCATION=/home/globus/globus-4.2.1/
  # cp $GLOBUS_LOCATION/share/myproxy/myproxy-server.config /etc
  # vim /etc/myproxy-server.config
  # diff /etc/myproxy-server.config $GLOBUS_LOCATION/share/myproxy/myproxy-server.config
  15,21c15,21
  <accepted_credentials "*"
  <authorized_retrievers "*"
  <default_retrievers "*"
  <authorized_renewers "*"
  <default_renewers "none"
  <authorized_key_retrievers "*"
  <default_key_retrievers "none"
  —
  >accepted_credentials "*"
  >authorized_retrievers "*"
  >default_retrievers "*"
  >authorized_renewers "*"
  >default_renewers "none"
  >authorized_key_retrievers "*"
  >default_key_retrievers "none"
  # cat $GLOBUS_LOCATION/share/myproxy/etc.services.modifications >>/etc/services
  # tail /etc/services
  myproxy-server 7512/tcp # Myproxy server
  # cp $GLOBUS_LOCATION/share/myproxy/etc.xinetd.myproxy /etc/xinetd.d/myproxy
  # vim /etc/xinetd.d/myproxy
  # cat /etc/xinetd.d/myproxy
  service myproxy-server
  {
  socket_type = stream
  protocol = tcp
  wait = no
  user = root
  server = /home/globus/globus-4.2.1/sbin/myproxy-server
  env = GLOBUS_LOCATION=/home/globus/globus-4.2.1
  LD_LIBRARY_PATH=/home/globus/globus-4.2.1/lib
  disable = no
  }
  # /etc/init.d/xinetd reload

- Creating user certificate
  Now that myproxy is setup, we'll get a usercert for bacon. The globus user will add a new credential into myproxy. I have to specify a full name and a login name. I'll be using "Charles Bacon" and "bacon" for my user. I have to supply two different passwords. The first password is going to be the bacon user's password. The second password has to be my SimpleCA password from when I ran gt-server-ca.pl.
  # myproxy-admin-adduser -c "Charles Bacon" -l bacon
  Create a grid-mapfile as root for authorization. You can copy and paste the /O=Grid/OU=... subject name from the output above.
  # vim /etc/grid-security/grid-mapfile
  "/O=Grid/OU=GlobusTest/OU=simpleCA-elephant.mcs.anl.gov/OU=mcs.anl.gov/CN=Charles Bacon" bacon

- Set up GridFTP
  Now that we have our host and user credentials in place, we can start a service.
  # vim /etc/xinetd.d/gridftp
  service gsiftp
  {
  instances = 100
  socket_type = stream
  wait = no
  user = root
  env += GLOBUS_LOCATION=/home/globus/globus-4.2.1
  env += LD_LIBRARY_PATH=/home/globus/globus-4.2.1/lib
  server = /home/globus/globus-4.2.1/sbin/globus-gridftp-server
  server_args = -i
  log_on_success += DURATION
  disable = no
  }
  # vim /etc/services
  myproxy-server 7512/tcp # Myproxy server
  gsiftp 2811/tcp
  # /etc/init.d/xinetd reload
  Now the gridftp server is waiting for a request, so we'll run a client and transfer a file.
  $ myproxy-logon -s elephant
  Enter MyProxy pass phrase: ******

A credential has been received for user bacon in /tmp/x509up_u1817.
$ globus-url-copy gsiftp://elephant.mcs.anl.gov/etc/group file:///tmp/bacon.test.copy

- Starting the webservices container
  Now we'll setup an /etc/init.d entry for the webservices container.
  # cp $GLOBUS_LOCATION/etc/init.d/globus-ws-java-container /etc/init.d
  $ /etc/init.d/globus-ws-java-container start
  At this point, we can use one of the sample clients/services to interact with the container.
  As bacon do,
  $ globus-check-remote-environment -s https://localhost:8443
  Remote Endpoint Version Information Axis Version on remote endpoint https://localhost:8443:
  to check remote endpoint.This shows Axis and Java WS Core version.

- Configuring Reliable File Transfer

  We will use the globus-crft command to start a reliable file transfer. It takes an input file whose syntax is one pair of URLs per line. It will use RFT to manage the transfer of all the URLs in the transfer file. For this example, we'll just move a single file.
  As bacon,
  $ cat transfer
  gsiftp://elephant.mcs.anl.gov/etc/group
  gsiftp://elephant.mcs.anl.gov/tmp/asdf
  $ globus-crft -ez -f transfer
  Communicating with delegation service.
  Creating the RFT service.
  Starting the RFT service.
  Waiting for the RFT transfers to complete.
  Transfered 1 of 1 — Status: Done

- Setting up Grid Resourse Allocation Manager(GRAM) 4
  First we have to setup sudo so the globus user can start jobs as a different user.
  As root,
  # visudo
  Runas_Alias GLOBUSUSERS = ALL, !root;
  globus ALL=(GLOBUSUSERS) NOPASSWD: /home/globus/globus-4.2.1/libexec/globus-gridmap-and-execute -g /etc/grid-security/grid-mapfile /home/globus/globus-4.2.1/libexec/globus-job-manager-script.pl *
  globus ALL=(GLOBUSUSERS) NOPASSWD: /home/globus/globus-4.2.1/libexec/globus-gridmap-and-execute -g /etc/grid-security/grid-mapfile /home/globus/globus-4.2.1/libexec/globus-gram-local-proxy-tool *
  As bacon,
  $ globusrun-ws -submit -c /bin/true
  $ globusrun-ws -submit -c /bin/false
  Now we've got a working GRAM installation.

VII. Conclusion and Future Scope

A. Conclusion

I have given a brief explanation of grid computing.I have discussed in detail Globus GSI and it's implementation. The use of GNU/Linux as a platform and the opensource project Globus toolkit for implementation made it completely free of cost. Since all components of the toolkit follows OGF standard, it is highly reliable.Using globus GSI we can create a secret, tamper-proof, delegatable communication between software in a grid computing environment.

B. Future Scope

A number of scientific and commercial applications have started harnessing Grids. It can be observed that while there has been a lot of development for Grid technologies for eScience, there is still more to be achieved in terms of Grids providing computing utilities in the same manner as power utilities supply electric power. Ultimately, this would require development of richer services and applications on top of already existing ones so that Grid computing can move beyond scientific applications and into mainstream IT infrastructure.

References

[1] The Globus project : *www.globus.org*, Dec 2009.
[2] *Cryptography and Network Security* by William Stallings,Fourth edition,Perntice Hall, 2008
[3] Czajkowski Fitzgerald Foster and Kesselman. *Grid Information Services for Distributed Resource Sharing*, 2001.
[4] Berman, Ian Foster and Kesselman, *The Grid:Blueprint for a New Computing Infrastructure*, 1999.
[5] Butler, Engert, Ian Foster, Kesselman, Tuecke, Volmer and Welch. *Design and Deployment of a National-Scale Authentication Infrastructure.*, 2000.
[6] Ian Foster, Steven Tuecke, Carl Kesselman. *The Anatomy of the Grid : Enabling Scalable Virtual Organizations*, 2001
[7] *http://grid.ncsa.illinois.edu/*, Dec 2009.