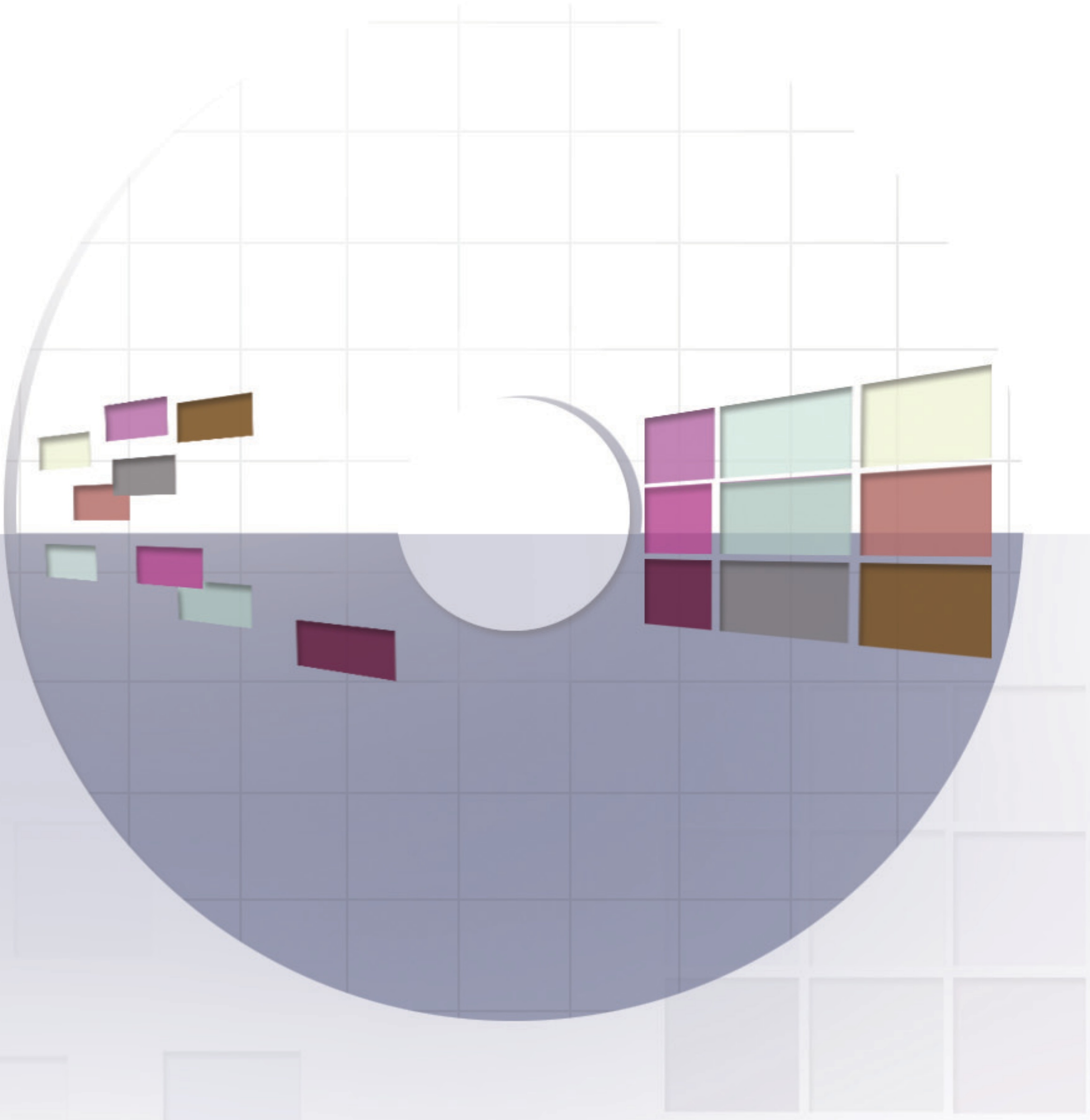
Remote Job Submission Design and Security Infrastructure



PAWEL KREPSZTUL
pkrepsztul@yahoo.com

Electrical and Computer Engineering Department
Fairfield University
Fairfield CT 06430

Project Initium

Remote Job Submission Design and Security Infrastructure

PAWEL M. KREPSZTUL
pkrepsztul@yahoo.com

A Thesis submitted to the Graduate Faculty of Fairfield University
in partial fulfillment of the requirements for the degree of
a Master of Science in the Electrical and Computer Engineering program

Advisor
Professor Douglas A. Lyon, Ph.D.

Electrical and Computer Engineering Department
Fairfield University
Fairfield CT 06430

TABLE OF CONTENTS

ABSTRACT

This paper presents a framework for grid computing that makes use of idle computers on a network. The goal is to provide the middleware needed to deploy jobs to non-geographically co-located clusters with decentralized look-up severs. We have named our framework the Initium Remote Job Submission (RJS) system. RJS opens the door to SETI-style grid computing on a heterogeneous network for Java programmers. Initium generates a jar file that is signed by a trusted certificate authority CA [Lyon]. The jar is run by a Computation Server (CS), which is an idle, remote computer running the Initium Computer Server Software. A Web Server (WS) has a Java Network Launch Protocol (JNLP) file that makes reference to a signed jar file on a web server. The signed jar file contains a job for the computation server. This jar file is called the computation jar. The computation jar is executed on the CS and the answer is sent back to the server using RMI over SSL (RMI/SSL). A look-up server (LUS) is used to register computation servers as they come on line. When a computation server is started, it registers with a LUS. The LUS then updates its list of computation servers. Look-up servers can be started using Java Web Start. They can be contacted with multicast protocols.

1. THE DISTRIBUTED COMPUTING SECURITY PROBLEM

We are given an idle computer on the Internet. Our goal is to make use of the idle CPU cycles of the computer. We seek to create a framework that exploits the idle CPU's. We are subject to the constraint that our code must be downloaded over the Internet (that is, a normally insecure channel). Further that the configuration should be automatic. In addition, the code should be verified as originating from a trusted source. Finally, computed answers should be returned over a secure link to a Web Server (WS). The web server is able to hold "jobs" for deployment, and hold computed answers.

1.1 APPROACH

Our approach for authentication and encryption is to use Java Secure Socket Extension (JSSE) to implement a Java technology version of Secure Socked Layer (SSL) protocols. This is integrated into the Java 2 SDK, Standard Edition v 1.4 [Sun 2004].

Our goal is to securely download a list of jobs to the LUS, and securely upload the answer jar to the WS. All LUS - CS commutation is on the LAN behind the firewall. Secure sockets (SSL) allow confidential communication. Most grid systems use SSL for authentication, but by default do not establish encrypted communication in a secure manner [Globus 2]. Our approach requires that we encrypt all WS-LUS communication via a session key. In order to establish the session key, WS and LUS must agree on shared key, without sending any secret data in the clear. To do this, we use the RSA key exchange algorithm as a standard method for SSL key exchange. In RSA key exchange, the WS encrypts a number of random bytes with the LUS's public RSA key and they both use this shared secret to create the session keys. In order to guarantee the integrity of the messages, we use the Message-Digest 5 (MD5) algorithm. The MD5 algorithm is intended for digital signature applications, where a large file must be "compressed" in a secure manner before being encrypted with a private (secret) key under a public-key cryptosystem such as RSA [Rivest]. The CS receives a URL to a jnlp file and downloads the Java Web Start "job" in order to compute it. The owner of the CS must accept the job owner certificate. Once the certificate is trusted, all jobs signed by it will be executed automatically.

RSA algorithms ensure secure communication and MD5 message digest ensures that no one has altered the data in transit.

An alternative approach to security might use the Kerberos network authentication protocol [Kerberos] or SSH protocol [OpenSSH]. However, such tools are vulnerable, as the attackers can gain access to user's password as it is being typed [Basney].

1.2 MOTIVATION

We are interested in grid computing because today's workstations are typically idle 14 or more hours per workday. If we assume that they are also idle on the weekend then they are idle 118 hours out of every 168 hours during the week (i.e., 70% of the time). This results in a huge amount of CPU cycles, storage capacity, and network bandwidth being wasted.

We are motivated to address security issues in grid computing because an insecure network could be exploited by an advisory in order to gain access to data, destroy data, and steal logins or

services (email, CPU cycles). Users will not agree to donate their CPU time without an assurance of security. We also need to make sure that programs that can be run on computers in the grid are safe to run.
Grids need to be protected not only because they are high-value assets representing lots of hardware and software, but also because they often serve a strategic function that's central to success [Myer]. The benefit of using our approach is elimination of tedious and time consuming software installations. Furthermore, this solution is operating system independent and secure.

2. LITERATURE SURVEY

Grid Computing is not new, nor for that matter is grid security. The use of grid computing on a heterogeneous network is also not new. What is new is our use of Java Web Start to distribute jobs on the grid via a screen saver. This opens the door to SETI-style grid computing on a heterogeneous network for Java programmers.

SETI, for example, does not work in Java [SETI]. Additionally, it is a closed system in that others can only contribute CPU cycles (but not programs). Our system differs from SETI in four major ways:

1. RJS is written in Java and so has binary portability,
2. RJS enables others to submit jobs to the grid,
3. RJS automatically configures the CS,
4. RJS is secure.

Globus uses the Grid Security Infrastructure (GSI, Globus Project Toolkit 3.0) to implement grid security [Silva]. GSI provides number of useful services for grids, including mutual authentication and single sign-on. A central concept in GSI authentication is the certificate. Every user and service on the Grid is identified via a certificate, which contains information vital to identifying and authenticating the user or service [Globus2]. Manipulating grid certificates using Globus Toolkit 3.0 (GT3) in Windows environments is awkward. It requires the system administrator or user to install GT3 on Linux systems in order to use command line scripts to generate certificates. Users must move those certificates to their Windows system [Silva 2].

The Globus Toolkit 3.0 consists of APIs that are built into the Java CoG Kit 1.1 that can generate a user certificate or certificate request. It can also sign certificates and create proxies [Globus2].

In many ways our approach is similar to GSI, but we are using pure Java implementation. In our judgment, the GSI technique is very secure; however, it is also cumbersome and costly.
XSOAP and XCAT grid web services use the GSI to provide Public Key Infrastructure [XSOAP]. Also, EU DataGrid project, authentication and delegation project is based on the Globus GSI,

which is an extension of the Public Key Infrastructure [Cornwall]. XSOAP and XCAT grid web services use GSI. As a result, XSOAP and XCAT grid web services suffer from the same advantages and disadvantages of GSI.

The JPARSS (Java Parallel Secure Stream for Grid Computing) package contains a run-time option of security features that enables the subject (a user or a process) to be authenticated by a JPARSS server only once during a time interval using a temporary X.509 certificate signed by the subject whose permanent X.509 certificate may be issued by any trusted certificate authority (CA) [Chen]. The advantage of this solution is one time authentication, but still there is a need for a CA to sign the temporary certificate. Furthermore, it requires password to create X.509 certificate. Hence the JPARSS system is also judged to be secure, but cumbersome.

Grid Portal Development Kit (not supported anymore), builds its security using few simple methods for setting the username, password and designated lifetime of the proxy [GPDK]. In addition to being abandoned, the system is not secure.

JGrid introduced an Authentication Service (AS) that provides short-term certificates for users without their own certificate. In this case, a user can register with the AS, log in a custom way, and obtain private key and certificate. So the AS is a CA (Certificate Authority) of the JGrid, but it provides short-term credentials [JGrid].

JGrid is build based on Jini technology. Jini network technology, which includes JavaSpaces Technology [Flenner] and Jini extensible remote invocation (Jini ERI), is open architecture that enables developers to create network-centric services that are highly adaptive to change [Sun2]. The approach of issuing short-term certificates is similar to JPARSS, thus it shares same advantages and disadvantages.

Condor provides support for strong authentication, encryption, integrity assurance, as well as authorization. Most of these security features are not visible to the user (one who submits jobs). They use configuration macros that are run by the site [Condor]. Since Condor project use GSI for authentication [Condor2], it suffers from the same advantages and disadvantages of GSI.

Gridbus Project has developed a Windows/.NET-based desktop clustering software and Grid job web services to support the integration of both Windows and Unix-class resources for Grid [GRIDBUS]. Unlike RJS, the installation of Gridbus is time intensive. In most of the cases compilation of the code is required. Also, before installation, many prerequisites, such as: IBM TSpace, Globus toolkit 2.4, MySQL or Microsoft .NET Framework 1.1, are required.

3. REMOTE JOB SUBMISSION DESIGN

RJS architecture is based on Java Web Start technology and is operating system independent. Java Web Start provides the power to launch full-featured Java applications with a single click without going through complicated installations procedures [Sun3]. The system is constructed with three major applications: the Web Server, Look-up Server and Computation Server as shown in Figure 3.1-2. All three systems are designed to run on separate computers.

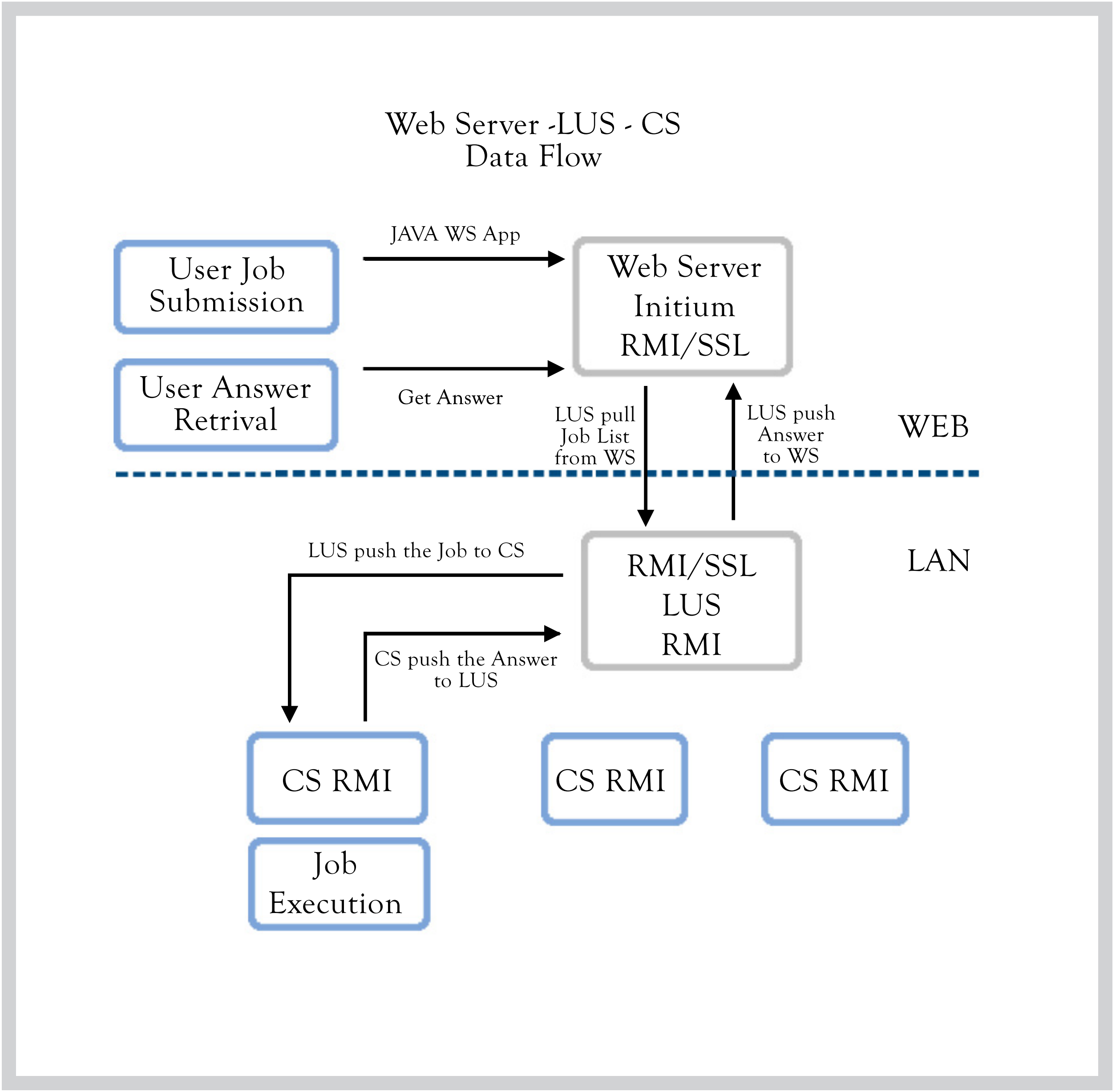


Figure 3.1-2 RJS data flow

3.1 WEB SERVER

The Web Server is designed for jobs and answer holder. Typically, the WS is started on a server with static IP address on WWW (e.g. http://lyon.fairfield.edu/~pawel). Web server must install

Java Runtime Environment (JRE) 1.4 or later version in order to run RJS. The server mime types must be updated to include jnlp extension; otherwise the Java Web Start applications will not be properly mapped and executed. Figure 3.1-1 shows required additions to /etc/mime.types file in Linux based web servers:

```
# MIME type                          Extension
application/x-java-jnlp-file         jnlp
```

Figure 3.1-1 Mime type additions in Linux

Access to Web Server is permitted only to registered users. Users are required to obtain accounts on WS, which include getting ids and passwords. Typically users can connect to WS using Secure Shell (SSH), but this may vary in some Web Servers. SSH is a program used to log into another computer over a network, to execute commands in remote machine and move files from one machine to another. It provides strong authentication and secure communications over insecure channels [SSH].

3.1.1 JOB DEPLOYMENT

Jobs are deployed to WS as Java Web Start (JWS) applications. To create and deploy a job, a user can use the Initium application (freely available at www.docJava.com under Web Start section). The jnlp file needs to be uploaded to user home/jobs directory, and signed jar plus any other resource jars to directories specified in the jnlp file. Figure 3.1.1-1 shows resource sections of jnlp file where some jars are located in libs directory:

```
<resources>
<j2se version="1.4+" />
<jar href="net.rmi.rjs.fc.FractalsJob_1.jar" />
<jar href="libs/jai.jar" download="lazy"/>
<jar href="libs/jmf.jar" kind="lazy"/>
</resources>
```

Figure 3.1.1-1 Resource section of jnlp file

Better yet, resources and native methods can be specific to operating system on which the job is executed. Figure 3.1.1-2 shows how to divide resources per operating system.

```
<resources os="Mac OS X" >
<jar href="libs/rxtx/mac/RXTXcomm.jar" download="eager" />
<nativelib href="libs/rxtx/mac/native.jar" download="eager"/>
</resources>

<resources os="Linux" >
<jar href="libs/rxtx/linux/RXTXcomm.jar" download="lazy" />
<nativelib href="libs/rxtx/linux/native.jar" />
</resources>

<resources os="Windows XP" >
<jar href="libs/windows/RXTXcomm.jar" download="eager"/>
<nativelib href="libs/windows/native.jar" />
</resources>
```

Figure 3.1.1-2 Operating system specific jnlp file resource section

The advantage of specifying OS related resources is the download time, where jars needed to perform the job are limited. The user needs to sign the jar job with his/her certificate, which is subject to verification at the computation server. Web Server starts RMI server registry on port 1099, and creates SSL socket on port 9000.

3.1.2 WEB SERVER RMI/SSL IMPLEMENTATION

In order to create RMI calls over SSL channel, we need to implement two custom socket factories: RMIClientSocketFactory and RMIServerSocketFactory (see Figure 3.1-2). RMIClientSocketFactory has one method createSocket (), which returns an SSLSocket.
RMIServerSocketFactory also has one method createServerSocket (), which returns SSLServerSocket. In each case the SSLSockets are created on fixed port 9000. This is to work around problems with firewall. RMI Server opens up a server socket on port 1099 and listens hire for incoming requests. If a request comes in, another port is used to handle the request and send

back the response. In default implementation, this is done on port 0, which is any available port chosen by the underling OS [Goffings]. In our case Web Servers firewall needs to allow communication on port 9000.

Keystore and truststore resources needed for SSL secure communication are stored in a code as byte arrays and loaded as needed. Keystores are databases of key pairs and certificates that are used to set up SSL authentication. Truststores are keystores that are used to verify the identities of other clients and servers. When a client or server is setting up an SSL session, it will retrieve its certificates and keys from its keystore. When it verifies the identities of other clients or servers, it will retrieve trusted certification authority (CA) certificates from its truststores [onJava]. Figure 3.1.2-1 shows how to create keystores and truststores from byte arrays.

```
ByteArrayInputStream clientKeyStoreInputStream =
        new ByteArrayInputStream(ClientKeyStoreFile.getCKSbytes());
ByteArrayInputStream clientTrustStoreInputStream =
        new ByteArrayInputStream (ClientKeyStoreFile.getCTSbytes());
KeyStore ks = KeyStore.getInstance("JKS");
ks.load(clientKeyStoreInputStream, passphrase);
KeyStore ts = KeyStore.getInstance("JKS");
ts.load(clientTrustStoreInputStream, null);
```

Figure 3.1.2-1 Loading keystores and truststores from byte array

Typically, keystores and truststores are stored in files and are deployed as resources and stored in fixed location. Storing them on a PC without administrative permission can cause a problem. We have addressed this configuration issue by bundling the resources directly in our code.

When WS is started, it waits for incoming LUS communication. Only LUS is allowed to contact the WS and remotely, and over secure channel pull the encrypted list of the jobs.

Current system scans only one directory for the job, but in the future can be easily extended to all users' directories.

3.2 LOOK UP SERVER

The Look up server is designed to run on Local Area Network, with many geographical collocated PCs available as Computation Servers. LUS can directly connect to Web Server and initiate communication as a RMI/SSL Client (see Figure 3.2-4). It also starts RMI File Transfer Server to upload the jobs to CS (see Figure 3.2-4). Since LUS is a typical "man in the middle" it needs to know the WS and CS IP addresses for proper communication.

When LUS is started it first asks the user for WS IP address, as shown in Figure 3.2-1.

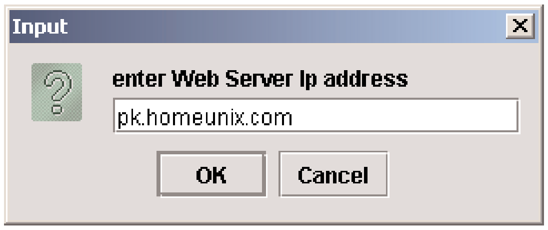
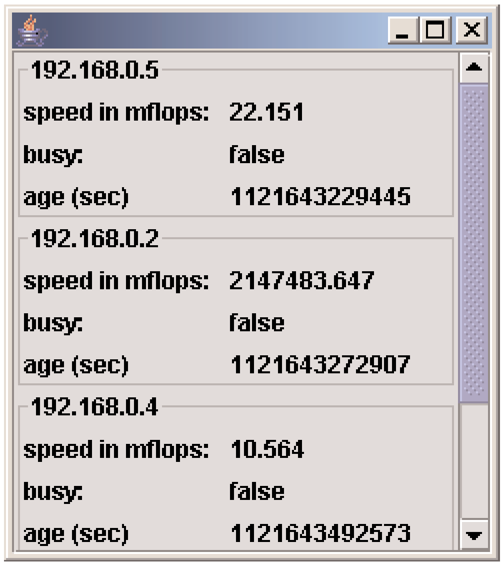


Figure 3.2-1 The Web Server Dialog Box

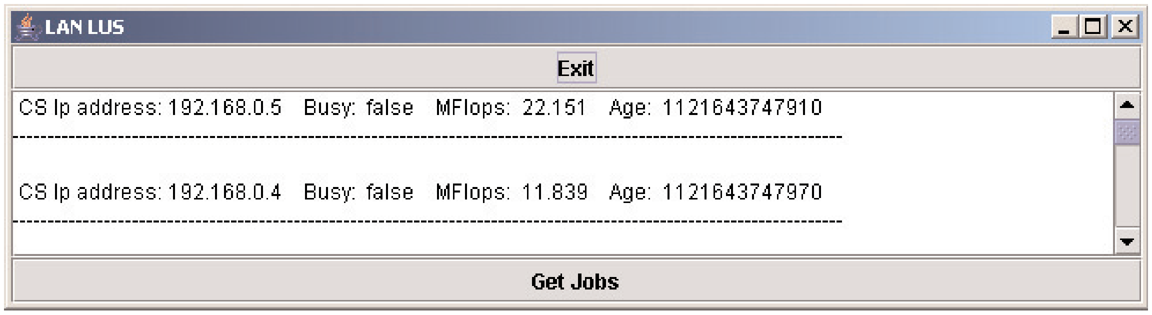
The next step of the LUS is to discover available CS's. Every CS, multicasts its availability. The LUS discovers the CS via the multicast broadcast and registers the following properties of CS: IP address, benchmark speed in Mega Flops, busy state (true or false) and age (time since last update in milliseconds). All CS's and their characteristics are displayed on a LUS panel as shown in the Figure 3.2-2.



```
192.168.0.5
speed in mflops:  22.151
busy:             false
age (sec)         1121643229445
192.168.0.2
speed in mflops:  2147483.647
busy:             false
age (sec)         1121643272907
192.168.0.4
speed in mflops:  10.564
busy:             false
age (sec)         1121643492573
```

Figure 3.2-2 The LUS Panel

Finally, LUS is ready to get a list of available jobs from web server. In the current setup it is done by clicking a "Get jobs" button on LAN LUS control panel, as shown in Figure 3.2-3.



```
CS Ip address: 192.168.0.5   Busy: false   MFlops: 22.151   Age: 1121643747910
------------------------------------------------------------------------------------------------

CS Ip address: 192.168.0.4   Busy: false   MFlops: 11.839   Age: 1121643747970
------------------------------------------------------------------------------------------------
```

Figure 3.2-3 The LUS Control Panel

When the "Get Jobs" button is clicked, RMI/SSL call, which is repeated every 10 seconds, is made to the Web Server requesting a list of available jobs. The list contains JNLP URLs. Once the list is returned to the LUS, it searches for available computation servers that can execute the job/s. In its search, the LUS selects computation servers that are both available and fast. The search repeats every 10 seconds until the job queue is empty. One a CS is identified, the LUS uploads the "job URL" to CS, updates the CS state to busy and removes the job from the queue.

Once the CS completes the job execution it sends the answer jar to LUS. Subsequently, LUS automatically opens an RMI/SSL call to Web Server and uploads the jar to user home/ans directory.

3.3 COMPUTATION SERVER

Computation Server is designed to execute jobs and upload answers to the LUS. When CS is started, it runs a Linpack benchmark that determines its speed. CS then multicasts its availability to LUS and starts an RMI File Transfer Server (see Figure 3.2-4), needed for uploading answers jars back to LUS. Until the LUS sends the job to the CS, the computation server is in a waiting stage. Once the job is received from LUS, the CSAnswerFactory is activated. The factory is responsible for running the WebStartLauncher and uploading answer jars to LUS. Web Start Launcher executes the Java Web Start job on the URL provided by LUS and creates an answer jar in user home/rjs directory. Then the launcher returns the answer jar to CSAnswerFactory, which in turn uses the RMI File Transfer Client to upload the answer to LUS.

An additional feature of CS security is a dialog box that allows CS owners to have control over what runs on their PCs. The dialog box always appears at the time when the job is signed by not trusted certificate. Once the CS owner trusts the Certificate, all other jobs signed by the same certificate will be executed automatically. Otherwise, no jobs will be executed.

Currently, Web Start Launcher needs to know where the Java Web Start executable resides on the system. The path to Java Web Start executable is hard-coded for various operating systems, but in the future needs to change for automatic detection.

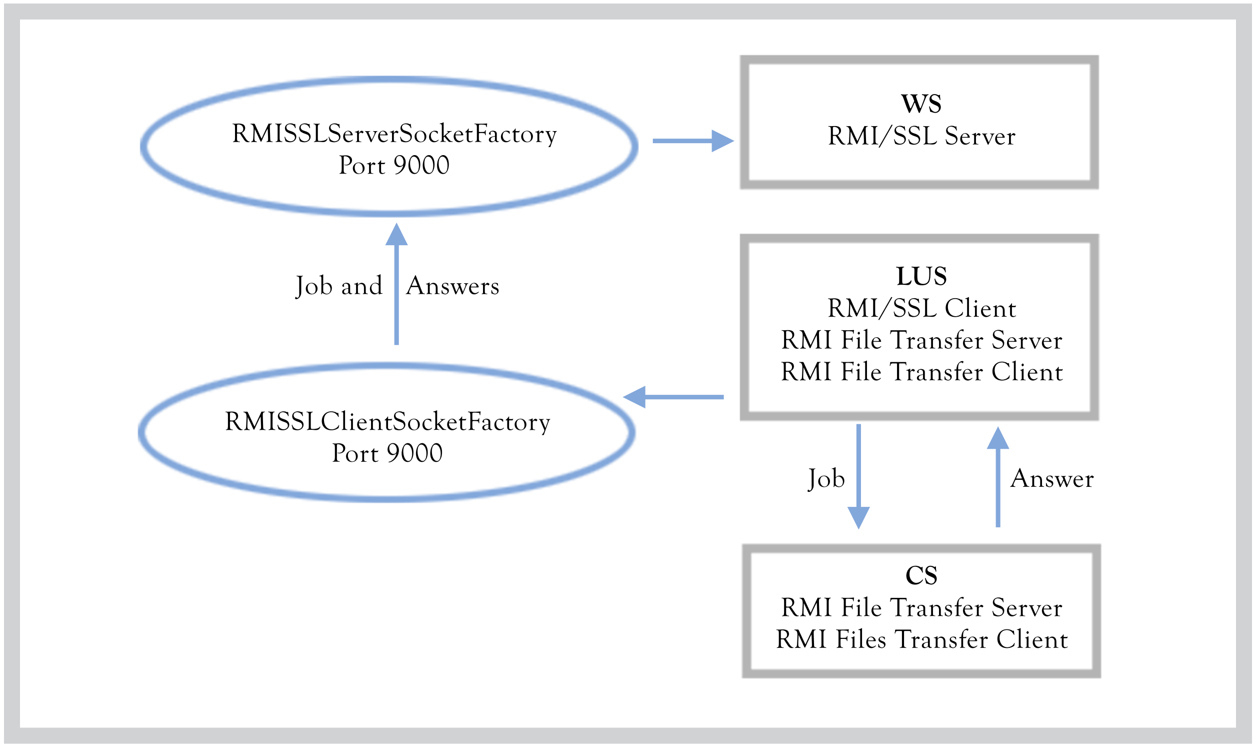


Figure 3.2-4 RJS RMI/SSL and RMI design

3.4 USERS REQUIREMENTS

In order to use RJS grid system, several requirements need to be fulfilled by the user. Foremost, the user is required to obtain an account on web server. Next, the user needs to create jobs directory under user/home. The user must also obtain a certificate of trust. The Initium X.509 Certificate Wizard paper describes the use of the Thawte's "Web of Trust" X.509 certificates for signing and distributing executable Jar resources {Lyon2].

Subsequently, the user needs to deploy jobs to the account. As mentioned in Section 3.1, Initium can be used for automatic deployment. Initium will perform static dependency analysis (SDA) packing only needed classes, generating signed jar, creating jnlp file and uploading it to chosen server over secure channel (SSH). Lastly, the user is responsible for obtaining the answers generated by RJS.

3.5 JOBS REQUIREMENTS

All jobs posted into RJS system must be deployed as Java Web Start applications. Each job must be partitioned into several sub-tasks, and each sub-task needs to be deployed as separate Java Web Start application. Each application must call System.exit (0) when it is done to prevent multiple JVM hanging on the CS PC. The output of the job, referred to herein as the answer, must be in a jar file. All jobs and answer jars must be clearly identified by a unique file name.

4. EXPERIMENTAL RESULTS

Several experiments were run to benchmark RJS system. The first experiment was setup using Linux PC (650 MHz Pentium III) as a Web Server, Mac OS X (G3 350MHz) as CS and Windows XP (1 GHz Pentium III) as LUS. To compute eight jobs Mac OS X took about 490 seconds. Same experiment was repeated replacing Macintosh PS with much faster Windows XP 3.2.GHz Pentium 4 PC as CS. The execution of the same eight jobs took only 180 seconds. The average of those two runs for the eight jobs was 335 seconds. The next experiment was run with two computation servers, slower Macintosh (14.5 mega flops per second) and faster Windows PC (44.3 mega flops per second). As expected, total time of execution decreased to 120 seconds. The job distribution was six to two in favor for the faster Windows PC. This experiment proved that the system is using fastest CS's first and more often than slower ones. As a result, the jobs are executed quicker.

The experiment was repeated using additional, third, CS (Windows XP 900 MHz). The result was a decreased execution time of 40 seconds. In this case, the LUS distributed four jobs to the fastest PCs, three jobs to the next fastest PCs and the last job to the slowest one.
In the next trial, the number of CS's was increased to four, which resulted in savings of 20 seconds. The quickest CS completed three jobs. The next two CS's were equal in speed and completed two jobs each. The slowest one completed only one job.

The above trial was repeated two more times adding one computation server each time. Using five workers the system finished all eight jobs in 50 seconds, which is 10 seconds improvement over the previous test. Having six workers further decreased total time to 42 seconds. Figure 4.1 shows a table with all measurements from the benchmark.

The benchmark clearly shows that using more than one computation server results in quicker job completion. An increase from one to six workers resulted in eight to one improvement in time execution.

Num of CS	Num of jobs	CS OS /CPU speed	Benchmark speed In M Flops	Num of executed jobs per CS	Time (sec) to complete	AVG Time execution per job
1	8	MAC X 350 MHz	11.83	8	490	60
1	8	PC WIN XP 3.2 GHz	42.17	8	180	22.5
1	8	MAC and XP AVG	27	8	335	42
2	8	MAC X 350 MHz	14.5	2	120	15
		PC WIN XP 3.2 GHz	44.3	6		
3	8	MAC X 350 MHz	14.3	1	80	10
		PC WIN XP 900 MHz	34.33	3		
		PC WIN XP 3.2 GHz	41.24	4		
4	8	MAC X 350 MHz	14.61	1	60	7.5
		PC WIN XP 900 MHz	34.33	2		
		PC WIN XP 1 GHz	34.33	2		
		PC WIN XP 3.2 GHz	45.8	3		
5	8	MAC X 350 MHz	14.61	1	50	6.25
		PC WIN XP 900 MHz	34.33	1		
		PC WIN XP 1 GHz	34.33	1		
		PC WIN XP 1 GHz	34.33	1		
		PC WIN XP 2.4 GHz	41.2	2		
6	8	MAC X 350 MHz	14.61	1	42	5.25
		PC WIN XP 900 MHz	34.33	1		
		PC WIN XP 1 GHz	34.33	1		
		PC WIN XP 1 GHz	34.33	1		
		PC WIN XP 2.4 GHz	41.2	2		
		PC WIN XP 3.2 GHz	45.4	2		

Figure 4-1 Experiment log table

Figure 4-2 shows a chart of an execution time versus number of computation servers participating in a grid. For one CS executing all eight jobs, average of Macintosh run (slowest CS) and Windows XP 3.2 GHz (fastest CS) was used. For the rest five runs, data from Figure 4-1 was used. The chart clearly shows that increasing number of computation servers decreases the time of jobs execution.

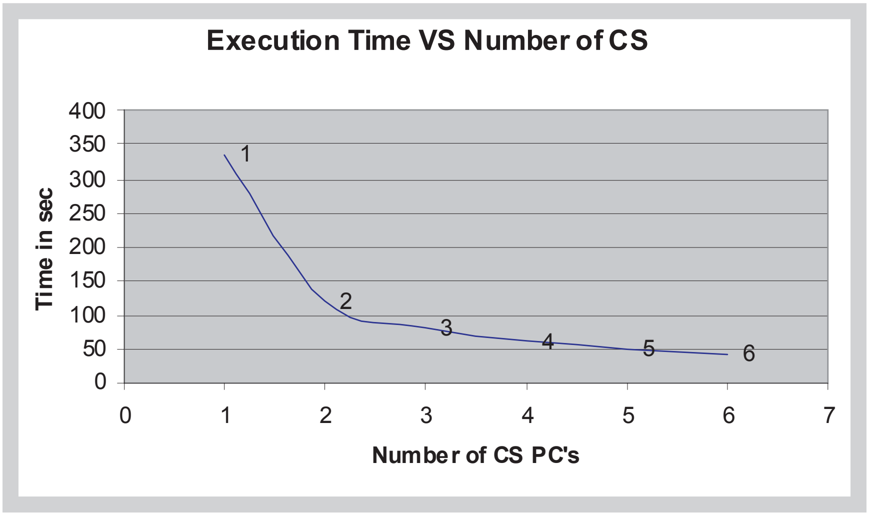


Figure 4-2 Execution Time versus number of participating CS's

Figure 4-3 shows negative correlation between the number of computation servers used and average time per job execution. As the number of computation servers increased, the average job execution time decreased from 42 to 5.25 seconds. Positive correlation was also observed between the number of CS used and average speed in mega flops. As the number of workers increased, the average speed of mega flops increased from 27 to 34.

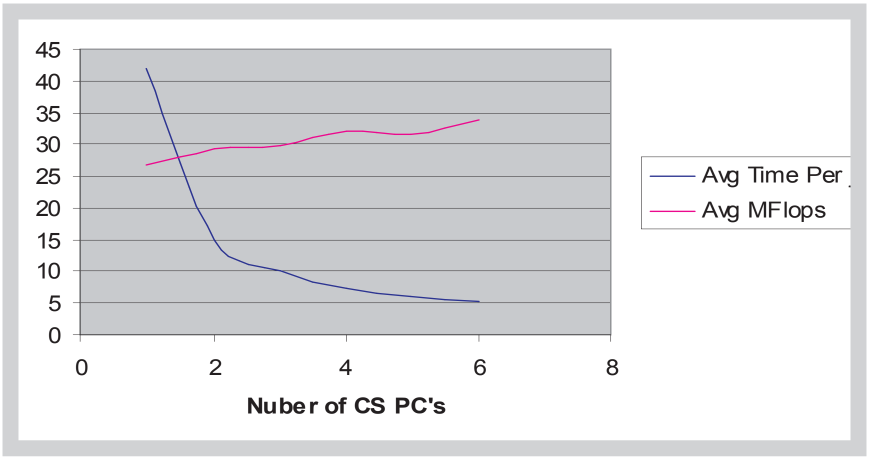


Figure 4-3 AVG Execution Time and AVG Benchmark speed versus number of CS's

For all experiments completed, the RJS system computed one Mandelbrot task subdivided into eight jobs. After each test experiment, the system created eight answers jars, and uploaded them to user answer directory. Figure 4.4 shows visual output from the all eight jobs. Images listed on Figure 4.4 are displayed in the order that they are returned by the CS's and not the correct order for display.

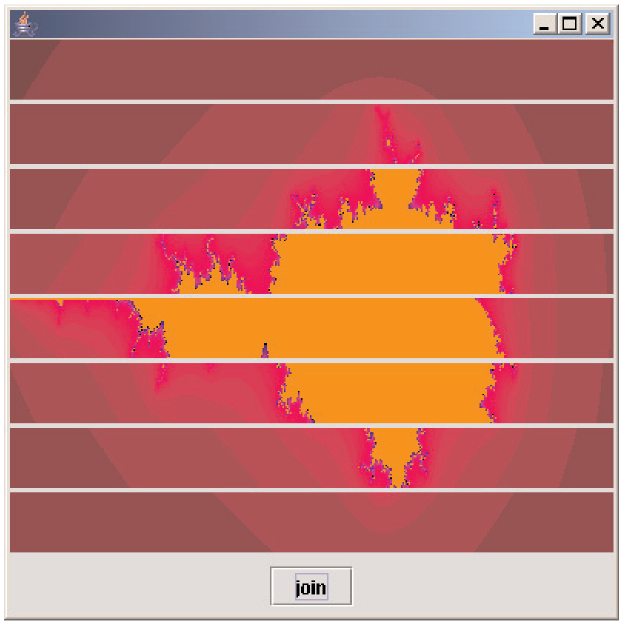


Figure 4.4 Visual outputs from the jobs

Figure 4.5 shows the jobs after they have been spliced together into a single image.

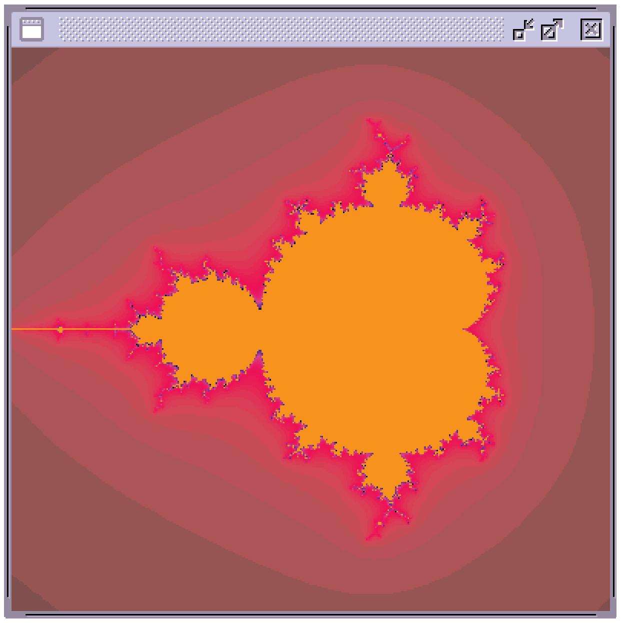


Figure 4.4 Mandelbrot set

5. CONCLUSION

There are several grid systems on the market today. One of them is RJS. RJS differs from most of the grid systems available in many significant ways. The major contributions and advantages of RJS are its deployment, portability and security.

RJS' deployment has many benefits over the grid systems on the market. Unlike many, the RJS system can be deployed and started on a new web server within minutes, through the use of Java Web Start technology.

The uniqueness of the RJS lies in the portability feature. Utilizing Java Web Start technology occurs so that the system is portable to various operating systems and automatic in its deployment.

The third advantage of RJS is its security. The system is secure from the job deployment (SSH) to the job execution. All communications sent on unsecured channels is encrypted and transmitted over secured protocol (SSL). Further, the owner of the CS PC can refuse suspicious jobs by not trusting his or her certificates. Finally, using a message digest algorithm to verify the integrity of the jar files before web start will execute them thwarts man-in-the-middle attacks.

5.1 DISCOVERED PROBLEMS

The design of RJS was changed periodically in order to fix unexpected problems that occurred during the development of the system. In the original design, the RJS was to process jobs as jar files. While testing that approach, we discovered problems with loading additional resources that may be needed and deployed with the job. Although there is nothing wrong with loading extra resources, the size of the additional resources becomes a concern when the resources are platform dependent. For example, if the job requires an extra three resource jars, and all of them are platform dependent, nine resource jars would have been deployed with the job instead of only three. In order to correct this potential problem, the design was upgraded to ensure that jobs are deployed as Java Web Start applications. The importance of Java Web Start technology is its ability to load specific resources to appropriate operating systems.

5.2 FUTURE WORK

In the future, a leasing mechanism in the LUS should be introduced. Currently, once the CS is discovered by LUS, it stays registered even if it is no longer available. The only time the CS is removed from LUS list is when it is picked to execute a job, but is not present. Leasing would improve the current system by eliminating CSs that are no longer available. This would be accomplished by enabling CS to renew its lease with LUS periodically. If CS does not renew its lease, it is then removed.

Another possible improvement is on Web Server. A communication tool could be introduced to notify a user when the job is executed and the answer is ready. A simple email program can be written to send a message to user's account. A more sophisticated way would be to introduce an application that would be running on the user's PC. This application would periodically scan user home/ans directory looking for answer jars. Once an answer is found, it is automatically uploaded to a user directory.

The third and most important area of future work is automatic idle detection. In the current system LUS and CS are started manually. An automatic launcher that can detect idleness of a PC and start computation server on the grid is needed. One way to do this is by introducing screen savers. Screen savers already can detect when a computer is idle.

5.3 KNOWN ISSUES

Java Web Start versions 1.4 and older are unable to register users' certificates. Because of this limitation all CS users, run older versions of Java Web Start, will be asked to trust every new job that is posted to grid. In newer versions of Web Start this issue is fixed, where users can trust the signers certificate only once (by clicking Always button) and all jobs signed by the same certificate will execute automatically. This is documented in the j2se enhancements section on Sun's site http://java.sun.com/j2se/1.5.0/docs/guide/deployment/enhancements-1.5.0.html.

Source code for this project is freely available from http://www.docjava.com under the section titled "Java for Programmers".

6. LITERATURE CITED

[Basney] "A Roadmap for Integration of Grid Security with one time Passwords" by Jim Basney, Von Welch, Frank Siebenlist, April 18, 2004,
http://www.ncsa.uiuc.edu/~jbasney/ grid -otp.pdf

[Chen] "Java Parallel Secure Stream for Grid Computing" by Jie Chan, Walt Akers, Ying Chen and William Watson III, High Performance Computing Group,
http://www.jlab.org/hpc/papers/jparss.pdf

[Condor] 3.7.3 Security Configuration,
http://www.cs.wisc.edu/condor/manual/v6.4/3_7Security_In.html

[Condor2] "Condor and the grid" by Karthik Ram Venkataramani,
http://www.ccr.buffalo.edu/grid/ download/ condor -and-the-grid.pdf

[Cornwall] "Security in multi-domain Grid Environments" by Linda Cornwall and team,
2004 Kluwer Academic Publisher, April 15,2004,
http://www.urec.cnrs.fr/publications/GridJournal-EDG-SCG-paper.pdf

[Flenner] "First Contact: Is There Life in JavaSpace" by Robert Flenner,
http://www.onjava.com/lpt/a/750

[Garms] "Professional Java Security" by Jess Garms and Daniel Somerfield, Wrox Press Ltd., Birmingham UK, 2001

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

[Globus2] Overview of the Grid Security Infrastructure (GSI),
http://www-unix.globus.org/security/overview.html

[Goffings] "RMI Through a Firewall" by Tim Goffings,
http://www.javacoding.net/articles/technical/rmi-firewall.html

[GPDK] "Grid Portal Development Kit", DOE Science Grid,
http://www.doesciencegrid.org/gridportal.html

[JGrid] "The security architecture of the JGrid System" by Mark Magyarodi
http://pds.irt.vein.hu/documentation/JGrid_security.pdf

[GRIDBUS] "The Gridbus Middleware 2004 ",
http://www.gridbus.com/middleware/

[Kerberos] "Kerberos: The Network Authentication Protocol",
http://web.mit.edu/kerberos/www/#what_is

[Lyon] "Project Initium: Programmatic Deployment" by Douglas Lyon, June 24, 2004.
http://show.docjava.com:8086/pub/document/jot/web.pdf

[Lyon2] "The Initium X.509 Certificate Wizard" by Douglas Lyon, private communication with the author, November, 2004.

http://show.docjava.com:8086/pub/document/jot/initium.pdf
[Myer] "Grid watch: GGF and grid security" by Thomas Myer, 13 May 2004,
http://www-106.ibm.com/developerworks/library/gr-watch4.html
[onJava] "Secure Your Sockets with JSSE",
http://www.onjava.com/pub/a/onjava/2001/05/03/java_security.html
[OpenSSH] OpenSSH,
http://www.openssh.org
[Rivest] "MD5 Algorithm" by Ronald L. Rivest, April 1992,
http://www.kleinschmidt.com/edi/md5.htm
[SETI] Seti@home,
http://setiathome.ssl.berkeley.edu/
[Silva] "Manage X.509 certificates in your grid with Java Certificate Services" by Vladimir Silva, ibm.com,
http://www-106.ibm.com/developerworks/grid/library/gr-jsc/, October 2003
[Silva 2] "Using Java technology with Globus Grid Security Infrastructure" by Vladimir Silva, ibm.com,
http://www-106.ibm.com/developerworks/library/gr-ggsi/, September 2003
[SSH] "Ssh (Secure Shell) " by Thomas Konig, University of Karlsruhe,
http://www.rz.uni-karlsruhe.de/~ig25/ssh-faq
[Sun2] "Jini Network Technology",
http://wwws.sun.com/software/jini/
[Sun 2004] "Java Secure Socket Extension (JSSE)" by Sun Microsystems,
http://java.sun.com/products/jsse
[Sun3] "Java Web Start White Paper" by Sun Microsystems, July 2005,
http://java.sun.com/developer/technicalArticles/WebServices/JWS_2/JWS_White_Paper.pdf
[XSOAP] "Grid Web Services: Security in XSOAP and XCAT",
http://www.extreme.indiana.edu/xgws/security/