# Optimising Security Configurations with Service Level Agreements

**M. Smith, M. Schmidt, N. Fallenbeck, C. Schridde, B. Freisleben**
*Department of Mathematics and Computer Science, University of Marburg, Hans-Meerwein-Strasse,, D-35032 Marburg, Germany*
(matthew,schmidtm,fallenbe,schriddc,freisleb)@informatik.uni-marburg.de

**Abstract.** Security and data integrity are important aspects in the fields of Grid and cluster computing. However, security usually incurs a certain amount of performance degradation and adds usage complexity to a field of computing where performance is crucial and usage complexity is already high. The growing popularity of Grid computing is leading to vastly different security requirements. While some Grid security mechanisms are already configurable, many others such as firewalls and advanced sandboxing techniques are usually configured statically per site according to particular user community needs. In this paper, we present a WS-Agreement approach for a fine grained security configuration mechanism to allow an optimization of application performance based on specific security requirements. The approach is demonstrated using an industrial optimization application from the area of metal casting.

**Keywords:** Grid computing, cluster computing, Service Level Agreement, network-level security and protection, site security monitoring, cryptography.

## 1 Introduction

The Grid computing paradigm is aimed at providing resources (such as compute clusters, data, access to special appliances and even people) as easy as electricity is provided through the electrical power Grid. This necessitates that the Grid must be easy and transparent to access and use. Unlike traditional cluster computing in which only a small number of users work in a closed system, Grid computing exposes local clusters to a large number of users via the Internet using open Grid middlewares such as Globus, gLite and Unicore.

This leads to a great variety of users and consequently a great variety of requirements a Grid environment must fulfil. Traditionally, Grid computing has been utilised mainly by academic organisations in low security (mostly no security) environments. The data and software processed is free and open to all. The only important security aspect was access control to limit resource usage to enable fair use of the academic resources. One example of such a project is the AstroGrid, one of the German D-Grid community Grids.

Lately, industrial interest in Grid computing, especially on-demand Grid computing, has grown and consequently the security requirements have grown as well. In previous work we have analysed the new security threats for on-demand Grid computing [15] and have introduced a number of new security countermeasures [14] to protect commercial Grid applications and users.

While these security mechanism are a first step to enable wide spread commercial adoption of Grid computing, they do incur a performance degradation through the use of firewalling, encryption and virtualisation and thus are a step in the wrong direction for the academic users who simply want as much computational power with a minimal amount of hassle as cheaply as possible, with no security requirements.

Even within a commercial environment there are different levels of security requirements, for instance the engineering users we work with in the InGrid project as part of the German D-Grid initiative require end-to-end encryption to and from the Grid resources and intra-node security through virtualisation on the compute resources [14]. However, other project working on dedicated Grid resources only require encryption of the results since the input data is freely available and does not need protection and the compute nodes are used exclusively.

These different security requirements have the potential to split the Grid community, since the engineers of InGrid will not work on unsecured Grid resources and the AstroGrid users do not accept the extra burden and performance costs of secured Grid resources. This leads to a partitioning of Grid resources, since most Grid sites we know including our own have a specific set of security mechanisms in place which apply to all users.

In this paper, we present an approach to optimise Grid application performance by tuning service and job security settings based on user supplied WS-Agreement specifications. This is facilitated by new flexible Grid security mechanisms, allowing different levels of encryption and sandboxing to be selected on-demand. We suggest a service-oriented mechanism for configuring and selecting security features with performance constraints based on a WS-Agreement approach. The approach is demonstrated using an industrial optimization application from the area of metal casting.

## 2 Problem Statement

In this paper, we deal with performance optimization of different Grid applications with varying security and performance constraints, which we encountered during our work on the German national Grid project D-Grid [12]. The aim of the D-Grid project is twofold. In the first phase, a research Grid is to be created linking the existing high performance compute resources of German universities and research institutions in a free academic Grid. The second phase is to encourage a pay-per-use of the Grid by industrial users. To enable the second phase, new security mechanisms needed to be introduced to the Grid environment which reduced performance and the usability for the communities of the first phase. To better understand why the security mechanisms interfering with applications of the first phase were introduced, in the following we will explain the threats which needed to be countered for applications of the second phase.

### 2.1 New Threats Against Clusters

The introduction of a large number of locally unknown Grid users to the local cluster environment is not welcomed by the cluster administrator or by existing cluster users. While standard operating system user protection schemes are in place, the idea of running unknown code and letting unknown users work on the local system is not acceptable. To deal with this concern, we previously introduced a Java based [16], Jail based[9] and a Xen [3] based virtual execution environment into which Grid users are placed so they can not harm other users. For the later an Image Creation Station (ICS) is used so that Grid users can create Xen images in which they can interactively install their software. This image is then deployed onto the cluster. This counters the concerns of the cluster administrator and the cluster users where the Grid users are concerned, since all Grid users are constrained to their sandbox and do not have access to the rest of the system [15].

However, a further concern of the cluster administrator and the cluster users is the introduction of relatively young and uncertified Grid middleware into the existing secure local cluster setup. The standard Grid setup of Globus, gLite and Unicore calls for the Grid headnode and the cluster headnode to be on the same physical machine or at least in the same network. This endangers the local cluster and its users, since a remote exploit of the Grid middleware [19, 18, 10, 17] allows an attacker to also compromise the cluster network. Furthermore, the Grid headnode must be reachable from the Internet, exposing the cluster to external attacks, which in a purely local setup would not be possible.

To combat this threat, we introduced a Grid enabled dual laned Demilitarized Zone (DMZ) which separates the Grid headnode and the cluster network [13]. However, the creation of a Grid DMZ creates a number of new problems which must be dealt with. Previously, the Grid headnode was responsible for the distribution of job data via direct access to the cluster scheduler. Due to the newly introduced network separation, this approach is no longer possible, since an open connection from the Grid headnode into the cluster network would also open a route for attackers. Furthermore, this separation of the

Grid headnode from the cluster creates some security concerns for the Grid users, which leads to the second problem area discussed in the next subsection.

### 2.2 New Threats Against Grids

Due to the introduction of a DMZ, we face the problem that the Globus Security Infrastructure (GSI), which is the security solution for the Globus Toolkit and gLite, is no longer sufficient to ensure the safety and integrity of user data. In a standard setup, GSI is responsible for the integrity of the data, but the assumption is that the Grid headnode has full access to the cluster. Data encrypted with the GSI is decrypted by the Grid headnode - now in the DMZ - which potentially is compromised. As a consequence, GSI-secured job data stored on the Globus headnode inside the DMZ is no longer safe respectively private. This is unacceptable especially for industrial adoption. Customers wanting to access external Grid cluster resources need to know that their software and data is protected not only during transmission but also during computation on the cluster resources. To this end, we propose an extension of GSI, which cryptographically links the Grid client, through the DMZ and the cluster scheduler, with a virtual execution host, which we introduced in previous work [15, 8], to enable end-to-end cryptographic protection of data.

While the introduction of the DMZ protects the cluster users from Grid attacks and the new encryption scheme protects the safety and integrity of Grid job data, there is still the danger of denial-of-service attacks against insecure Grid middleware in the DMZ. To face this threat, we need a setup that enables us to detect possible attacks and allows us to take appropriate countermeasures. To accomplish this task, we extended standard Network Intrusion Detection Systems (NIDS) by Grid specific attack signatures.

### 2.3 Security Overhead

All of the introduced security mechanisms introduce a performance overhead in the overall Grid operation. While some of the mechanism like the DMZ border firewalls and the NIDS work on a per site basis and must be coped with by all applications, others can be configured on a per users or even per Job/Service call basis:

- Encryption of input data
- Encryption of working data
- Encryption of result data
- Secure ClassLoaders[1]
- Jailing[1]
- Virtualisation[1]
- Inter-node firewalls

This opens up the opportunity to optimise application performance by selectively downgrading or switching of certain security measures which are not required by some applications. To better accommodate the different usage scenarios a service oriented description and configuration mechanism is required to allow manual as well as automated

[1]If sandboxing is implemented for a site, the user has a choice of sandboxing technology. It is not advisable to let users run their software outside a sandbox if there are also users requiring the protection of a sandbox. While it should be possible to prevent users from attacking other sandboxed software using standard operating system security, the risk to other users requiring sandbox protection is to great and thus sandboxing should be a per site decision. However what type of sandbox is used can be left up to the users depending on the type of software to be sandboxed and amount of configuration the users and the administrator are willing to spend setting up the sandbox. See [14] for a discussion of the different sandboxing mechanisms and their performance and configuration constraints.

[1]

[1]

selection and configuration of security mechanisms. This allows application to optimise performance by selectively turning off certain security mechanisms automatically in an otherwise high security Grid environment.

## 3 Approach

We propose a WS-Agreement compatible approach to describing security requirements and capabilities in addition to the traditional WS-Negotiation attributes such as computational needs, quality-of-service (QoS) and pricing. Figure 1 shows a simplified version WS-Agreement scheme for the security mechanisms our Grid site offers. This allows the Grid environment to optimise the security settings for different applications using standardized mechanisms like WS-Negotiation. When a client wants to submit a job or use a service, (s)he can create an instance from this template to specify her/his fine grained security and performance needs. All required security measures TODO mentioned in section 2 can be addressed giving unprecedented access to the otherwise statically configured security mechanisms employed by the Grid site. This allows the user to specify in detail which security mechanisms are required at what strength, thus giving the Grid middleware the possibility to optimise performance based on those requirements. In the following we will look at some of the important security mechanisms in detail and discuss how performance optimisation is affected by them.

### 3.1 Demilitarized Zone

To prevent external intruders from accessing the cluster hardware by exploiting weaknesses in the Grid middleware, the Grid/cluster subnet is divided into two separate subnets, the border network and the cluster network. The DMZ guards both networks with a firewall configured to the specific needs of the network in question. The border firewall filters connections from the Internet and denies unwanted connections to all machines within the DMZ. However, since Grid middlewares require a large number of open ports to function correctly and efficiently, and a large number of fluctuating users need to access the Grid, the border firewall is relatively open.

While it is possible to try and accommodate most Grid applications with one single firewall configuration, this tends to lead to firewalls which are very open and thus not all that effective. On the other hand if the outer firewall is to restrictive many applications will not function correctly. If a firewall is to strict for an application the standard approach is for the user to call the administrator and explain what the application does and why it needs the extra ports. The request to open the firewall some more is then usually turned down. If the user is to be granted access, the administrator has to configure the firewall to fulfil the new policy. This can take some time depending on the workload of the administrator and the often is left in place to long since it requires extra effort to remove the new policies.

Using the firewall element in the WS-Agreement document a user can specify which ports, port ranges and what kind of throughput is required and the Grid site can advertise its firewall settings and capabilities, thus allowing both the automatic selection of sites with fitting firewall policies as well as a convenient semi-automatic way for firewall administrators to adapt to new application needs. For instance well trusted users could be granted special access at a click of a button, instead of having to follow the procedure mentioned above. Since it is well defined which user with which application needs which ports a temporary change in firewall policies is not very time consuming and can easily be changed back with little effort. But most importantly of all it is a transparent mechanism which clearly shows, which user and which applications need which policies.

The Grid headnode is located in the DMZ. The inner firewall guards the cluster network and prevents direct connections to the cluster subnet. To protect the cluster network, the inner firewall is very strict and only allows a single specially designed cluster connector to pass through and does not allow any interactive sessions to pass into the cluster network.

```
<xsd:complexType name="AgreementTermType">
<xsd:sequence>
  <xsd:element name="parties" type="tns:AgreementPartiesType"/>
  <xsd:element name="serviceInstanceHandle" type="xsd:anyURI"/>
  <xsd:element name="dependency" type="xsd:anyURI"
    minOccurs="0"
    maxOccurs="unbound"/>
  <xsd:element name="sandbox" type="tns:SandboxType"/>
  <xsd:element name="firewall" type="tns:FirewallType"/>
  <xsd:element name="input_encryption" type="tns:EncryptionType"/>
  <xsd:element name="result_encryption" type="tns:EncryptionType"/>
  <xsd:element name="working_encryption" type="tns:EncryptionType"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="AgreementPartiesType">
<xsd:sequence>
  <xsd:element name="client" type="xsd:anyURI"/>
  <xsd:element name="provider" type="xsd:anyURI"/>
</xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="SandboxType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Java"/>
    <xsd:enumeration value="Jail"/>
    <xsd:enumeration value="Xen"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="FirewallType">
<xsd:sequence>
  <xsd:element name="firewall" type="xsd:boolean"/>
  <xsd:element name="throughput" type="xsd:int"/>
  <xsd:element name="ports" type="tns:PortRange"
    minOccurs="0"
    maxOccurs="unbound"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PortRange">
<xsd:sequence>
  <xsd:element name="lower" type="xsd:int"/>
  <xsd:element name="upper" type="xsd:int"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="EncryptionType">
<xsd:sequence>
  <xsd:element name="algorithm" type="tns:AlgorithmType"/>
  <xsd:element name="strength" type="xsd:int"/>
  <xsd:element name="sign" type="xsd:boolean"/>
</xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="AlgorithmType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="AES"/>
  </xsd:restriction>
</xsd:simpleType>
```

Figure 1: Grid Security Schema

The cluster resides in the cluster network and consists of a cluster headnode and a set of virtual worker nodes. An architectural overview is presented in figure 2.

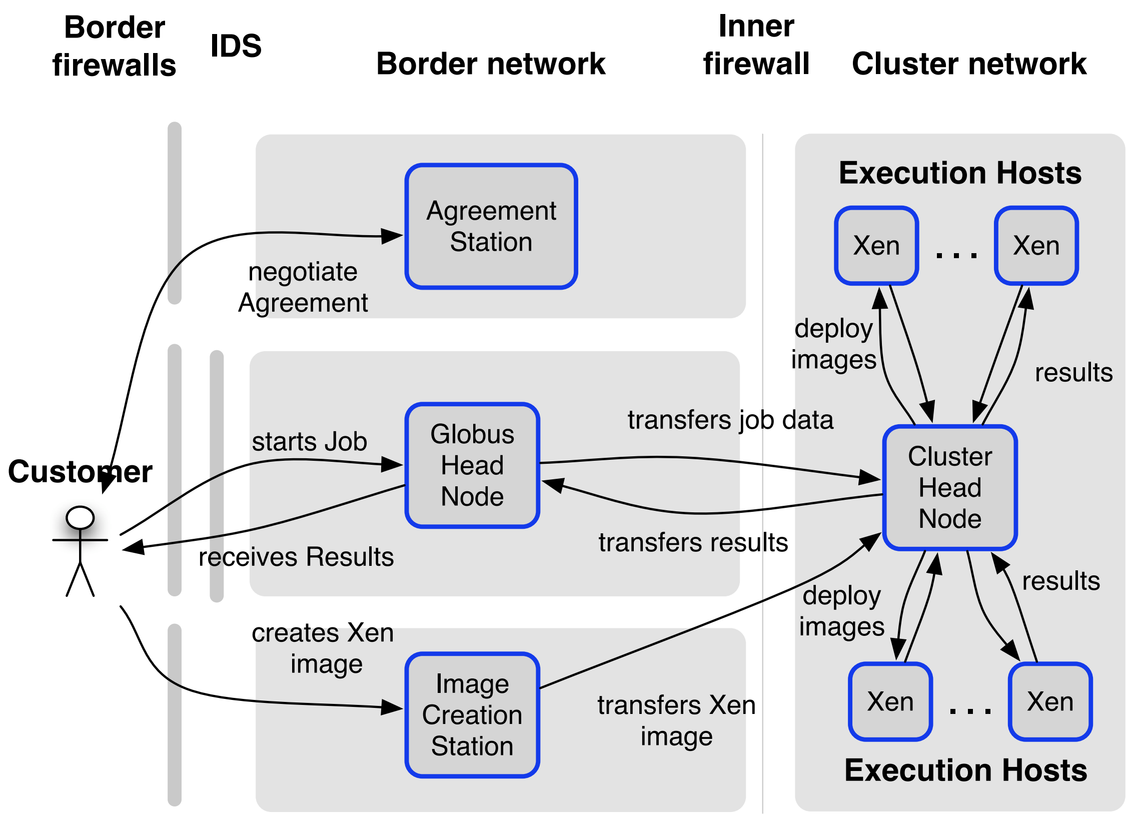

Figure 2: Architectural overview

If the user specified they want to use Xen as their sandboxing mechanism she has to create a Xen image using the ICS. Since the ICS must also allow users to log on from the Internet, it is located inside a part of the DMZ. Based on the X.509 Grid identity, the user can log on to an individually created image and install all required software. The user only gets to log on to his/her own XenU user image and thus can not compromise other images in the DMZ. To prevent an attacker from compromising the Grid headnode and then from there compromising the ICS, a dual laned DMZ approach is employed which restricts access to either the Grid headnode lane or the ICS lane. Thus, if one lane is compromised, the other lane remains unaffected. The other two sandboxing approaches are not affected by the DMZ.

## 3.2 End-to-End Encryption

Since jobs are no longer executed in the same network realm as the Grid headnode, a new encryption scheme is required. When a user submits a job, the client software (e.g. the Gridsphere portal [7]) generates a session key which is encrypted with the public key of the XGE (Xen Grid Engine), a modified version of the Sun Grid Engine used on the cluster headnode. This session key is used to encrypt both the job data and later also the results. Both the job and the encrypted session key can now be transferred through the insecure DMZ. Due to its location in the DMZ, the Grid headnode is considered unsafe, so the job data remains encrypted and the corresponding keys are not available outside of the secure network environment. This security mechanism is expensive since it is the client (with very little computational power) who must encrypt the data. The astrophysicists were particularly unhappy that encryption was "on" both ways since the input data definitely did not need protecting. Using the input_encryption element in the WS-Agreement document it can be specified if the data should be encrypted and what strength the encryption should be. This allows the engineering community to continue using strong encryption both ways but other communities can choose to send the data un-encrypted. Signing of the input data is separately selectable to ensure unaltered communication if needed.

## 3.3 Job Submission, Transfer and Execution

Since direct communication between the Grid headnode and the cluster scheduler is no longer possible, a new mechanism is required to transfer and execute a Grid job. A newly developed software called *Fence* deals with the task of transferring job data to the cluster headnode.

Fence extracts all relevant facts from the Globus job description, stores the encrypted job data locally and sends a message to the Cluster headnode. The message only contains the ID of the new job. On the cluster headnode initiates a connection back to the Globus headnode. This technique allows us to open only one port (for the notification message) in incoming direction on the internal firewall. All following connections are initiated from within the cluster network. After successful transmission, the connection is closed. This part of the security infrastructure is vital for the security of the existing cluster infrastructure and can not be switched of or reconfigured. However the component is transparent for the Grid job and thus poses no significant problem.

The XGE executes jobs not in a native but in a visualised environment, namely the user specific Xen-Image created in the ICS. Alternatively a Systrace or BSD Jail system could be used, or if the application is a pure Java application a Secure ClassLoader sandboxing solution can be used. For more information on these sandboxing solutions please refere to [16, 9, 15].

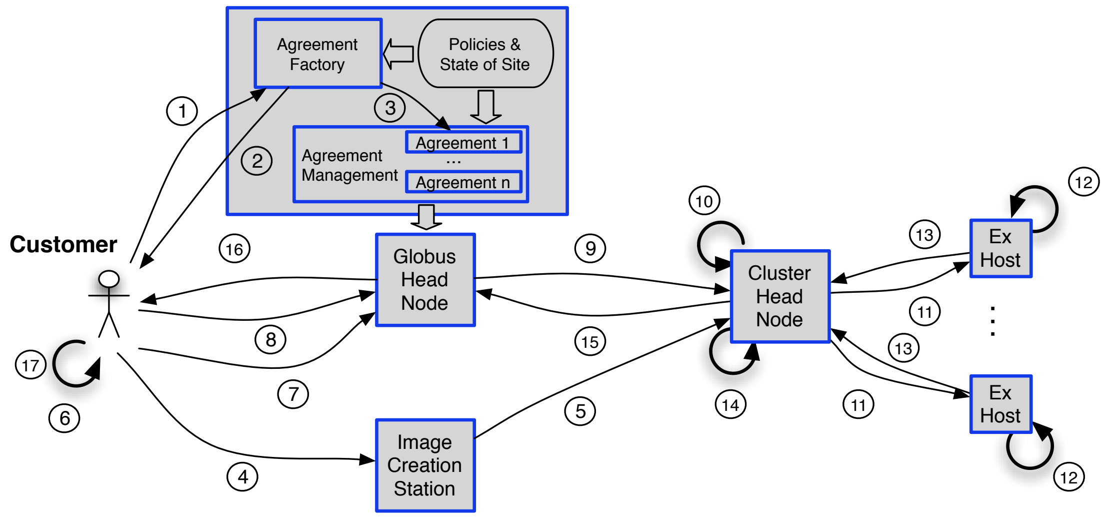

Figure 3: All steps required for a end-to-end encryption

A job submission is divided into several steps. The steps are shown in Figure 3 and explained in the following:

1. The users creates and submits a WS-Agreement file specifying which functional requirements she has (CPU, RAM, Time to finish, etc) as well as which security requirements she has (Encryption, Sandboxing, needed Ports, etc).

2. The WS-Agreement document is processed by the AgreementFactory and based on the evaluation if the job can be executed to the users satisfaction a positive or negative reply is sent. If both functional and security requirements are met, no changes are needed and the job can be optimized according to the required security settings. If however some requirements can not be met such as number of CPUs

or ports are needed which are closed and the site policy does not allow automatic opening of those ports or the user requires native execution (this is particularly relevant for Infiband Grid applications) but the site policy only allows execution of jobs in virtual environments, the user is informed about the restriction applying to this site. Steps 1 and 2 can then be repeated a number of times using WS-Negotiation until either a mutually acceptable setting is found or the request is cancelled.

3. The WS-Agreement accepted by both parties is stored by the AgreementManager and can be accessed by Globus and XGE to facilitate user centric system configuration.

4. If it is required that the job to be executed in a Xen based sandbox (either by the user or by the site policy) the user creates a fully customised Xen-Image for execution at the ICS. This image represents the base for the upcoming computation. If the job should be executed in a Jailing or Secure ClassLoader environment this step is not needed.

5. The image is transferred to the cluster headnode. The connection is initialised from the cluster network. This ensures maximum security, because no open incoming port towards the cluster network needs to be open. If the job should be executed in a Jailing or Secure ClassLoader environment this step is not needed.

6. If encryption is to be used in any part of the system, the client generates a session key and encrypts the session key with the XGE public key. If input_encryption is required it then archives and encrypts the job data with the session key and creates a customized RSL file. The RSL file contains the name of the archive and the encrypted session key.

7. The job data is copied to the Globus headnode via GridFTP.

8. A Globus GRAM call according to the RSL file is launched.

9. Globus hands over the job to Fence, which transfers the new job to the Cluster headnode.

10. The XGE parses the job description, if required decrypts the session key and decrypts the data.

11. The XGE starts the job either in the images created by the users or in a Jailed environment. If the job consists purely of Java services the Secure ClassLoader sandbox is also an option.

12. The job is executed. If working_encryption is required the working data must be encrypted if it is stored on disk.

13. The results are copied back to the XGE.

14. If result_encryption is required, the results get encrypted with the session key.

15. The results are copied back to the Globus headnode.

16. The user fetches the results with GridFTP.

17. If the results were encrypted the user decrypts the results.

## 3.4 Intrusion Detection

To detect possible intruders in advance, a NIDS is installed between the border firewall and the border network. The NIDS runs on a separate system and has no effect on the performance of Grid jobs and thus is not relevant for the optimization of the security mechanisms.

## 4 Experimental Results

While the presented features (DMZ, Fence) are required for effective security in Grid/-cluster environments, they create a certain amount of overhead. The transmission time is extended due to the extra hop via Fence, and the en- and decryption also consume some time compared to a plain text transfere of job and result data. In this section we will present the performance gains which can be achieved through security setting optimisation based on a users security needs..

The test environment consists of 6 machines connected with a 100 Mbit ethernet. The firewalls are Pentium III machines with 1 GHz, 512 MB RAM and FreeBSD installed. The Globus headnode is a Pentium IV with 3 GHz, 1 GB RAM and Debian Linux installed. The same configuration applies to the ICS and the client machine. The Cluster headnode is a Pentium IV with 1.8 GHz, 512 MB RAM and Solaris 10 installed. There are also 4 Pentium IV with 1.8 GHz, 512 MB RAM worker nodes running Debian Linux in a Xen environment.

Our test application is the casting simulation package CASTS (Computer Aided Solidification TechnologieS) [1] which is one of the engineering applications in the D-Grid. CASTS is a dedicated software tool developed for the simulation of casting processes. CASTS calculates transient temperature distributions in mold, core and alloy, taking into account both latent heat release as a function of fraction solid, and heat transfer resistance at material interfaces. The typical data volume for CASTS is about 290 MB.

The NIDS is a passive component, which does not influence job execution time.

### 4.1 Encryption

On of the most relevant issues in Grid security performance optimization is the encryption and decryption. The first measurement was done with transport encryption. We split the measurement in such a way that encryption, transfer and decryption are shown separately. Working data encryption was not used. The execution time is job dependent and thus is factored out of the measurements.

Figure 4 shows the times for the following steps:

1. Start a job on the client side and encrypt the job data with a 48-byte key
2. Transfer the job data via Fence to the XGE
3. The XGE detects the new job and decrypts the data

We conducted 50 trials to get a robust mean, which is about 154 seconds. The chart is divided into three parts. The bottom part displays the time needed to encrypt the data. The middle part displays the time needed to transfer the files via Fence from the Globus headnode to the Cluster headnode. The upper part displays the time needed to decrypt the data. The difference between the two cryptographic processes results from the diverse hardware (1.8 GHz CPU vs. 3 GHz CPU). The small variance in the transfer time is due the polling nature of the XGE scheduling algorithm.

In the second measurement no encryption was used so only the transfer time through the DMZ is added as overhead compared to an unsecured Grid. Figure 5 shows the time in seconds on the ordinate and 50 trials on the abscissa. The mean is about 49 seconds. Due to the watchdog characteristic of the XGE, we can see the same variance here as already described in the last case.

As can be seen the security overhead is doubled through encryption. And this applies for both input and result data. Compared with the time this CASTS job needs to complete (about one hour), the extra time (plus 105 seconds) is worth the gained security. However a casts job is only 290MB large which is small compared to other Grid applications which can easily have input data in the gigabytes. In these cases the encryption overhead is less negligible.

If a user skips the process of result encryption the overhead time also decreases about 50 seconds in case of a CASTS job. As stated in the last paragraph the time depends on the size of the data.

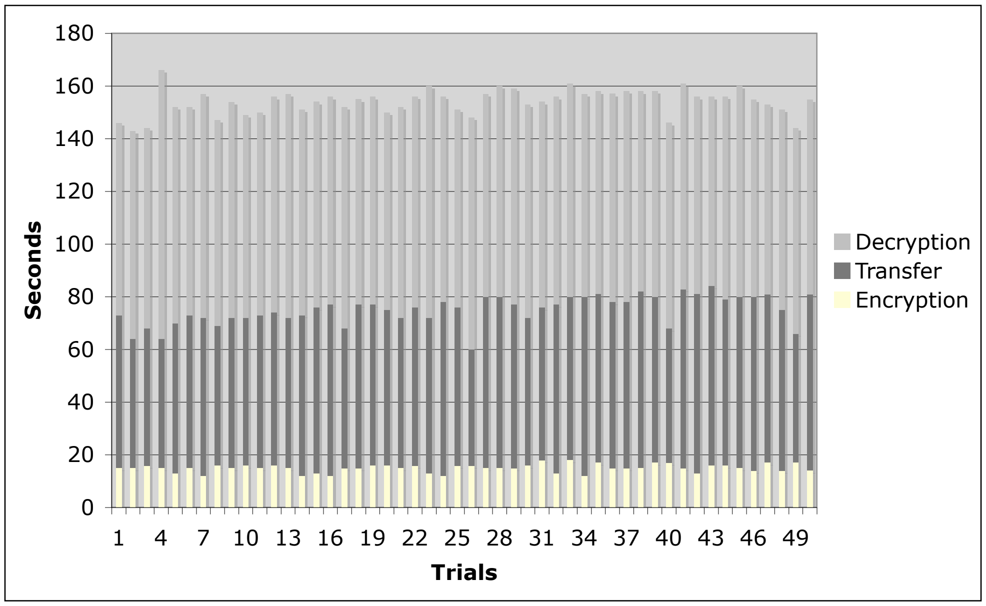

Figure 4: Time for fully encrypted job submissions in 50 trials

### 4.2 Sandboxing

Three different sandboxing techniques can be used:for pure Java services a Secure Class-Loader isolation is possible. For native applications a Jailing and a Xen based approach are possible.

#### 4.2.1 Secure ClassLoaders

Isolation of pure Java service is realised using a changed class loader delegation scheme. Since classes also need to be loaded by the standard implementation and the disposable and secure class loaders follow the implementation pattern of the regular class loaders, they do not impose an additional cost for loading of the classes. This security scheme also does not influence regular and repeated service invocations.

#### 4.2.2 Jailing

A significant overhead is created by the isolation scheme for native code fractions of a service using Systrace or BSD Jails. In this scheme, execution of the native code fractions of a service are performed in a process space separate from the JVM and sand-boxed by means of the underlying operating system. The solution is specifically aimed at isolation of native code that gets called from the Java service implementation via the JNI. A Java test class was created that uses a native method implemented using C. The absolute time before and after an invocation of the native method in the Java class was measured in a loop of 500 invocations. In the first case, the original native library compiled using gcc 4.0.1 was used. Then, the native library was replaced by a transparent proxy library that used RPC communication to perform method invocation on the original library in a process separate from the JVM. Again, the runtime of the native method invocation in the Java class was measured. The regular call took 3 microseconds on the average while the call using separate process spaces took 566 microseconds. This increase in time required

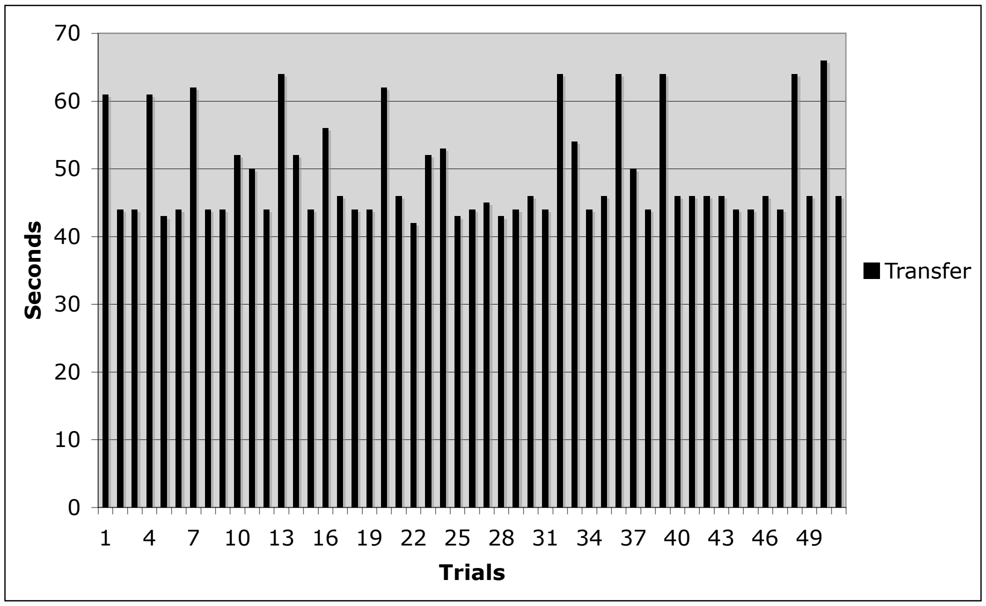

Figure 5: Time for unencrypted job submissions in 50 trials

to perform a native call by a factor of 182 is only acceptable if relatively few native calls are required from the Java code into native libraries.

### 4.2.3 Xen

There have been a number of performance studies of Xen and on the whole we have made the same observations. The real overhead for most applications including Casts is roughly 5%. For more detailed performance studies of Xen please consult [3, 6]. Figure 6 shows the results from [6].

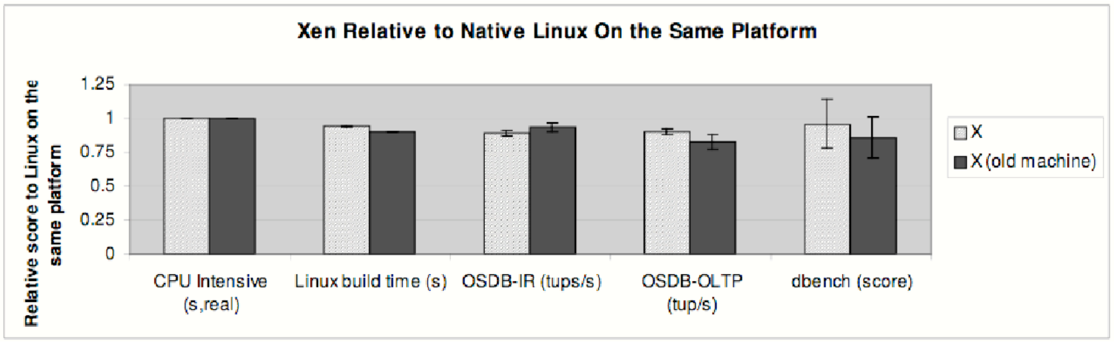

Figure 6: Relative Performance of Xen to native Linux on the same platform. Source [6]

## 4.3 Firewall

To compare the filter overhead introduced by the firewalls, we transferred data with and without the firewalls. To gain a robust mean, we transferred the data 100 times. The volume of the data is the same as above. Figure 7 shows the results of the measurement.

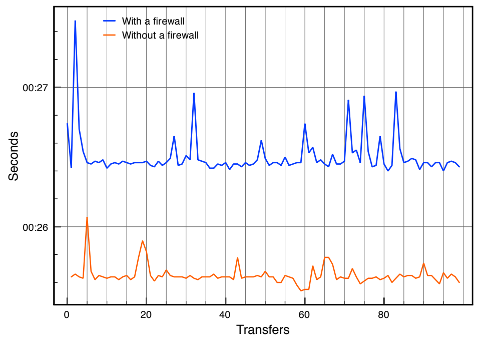

Figure 7: Transfer data chart with/without a firewall

The upper curve displays the transfer time through the firewalls, the lower curve displays the transfer time without firewalls. The mean of the upper curve is about 26.7 seconds, the mean of the lower curve is about 25.7 seconds. The firewall is one of the components which can not be switched of by users via the WS-Agreement mechanisms, however as the measurements have shown the firewall overhead is relatively small. The main problem posed by firewall in Grid environments is not the performance but the configuration, which is simplified by the WS-Agreement approach.

## 5 Related Work

Globus offers a wide range of remote services, so firewalls rules have to be chosen carefully in order to avoid disturbing legal users. Von Welch [20] analyzes Globus versions 3 and 4 in terms of network ports and data streams. Based on that, a fine-grained firewall configuration can be created, so that authorized users can work without disruptions, while at the same time blocking most unwanted traffic. A similar study was done by Baker et al. [2]. The authors do not discuss dynamic configuration of firewalls based on WS-Agreement.

A lot of work has been done in the area of WS-Agreement and WS-Negotitation mainly concerning itself with meeting minimum requirements such as number of CPUs, RAM, etc. and minimising cost. Legal aspect also play a major role in WS-Agreement work, since WS-Agreements are seen as legally binding contracts and as such mechanisms must be supplied allowing different departments to cooperatively create WS-Agreements. WS-Agreement and WS-Negotiation deal with matching customer demand with provider capabilities. A good overview of this topic can be found in: [11]. To the best of our knowledge not much work has been done in the area of configuring security settings to optimise overall performance based on minimum security requirements specified through WS-Agreement documents.

In [4] a solution is presented for the problem of Web service composition, focusing on security issues. The approach allows the composition of Web services according to specified security requirements of both Web service requestors and providers.

In [5] the authors propose an evaluation framework that includes a flexible quality meta-model for formalising Customer and Provider views of quality, and a decisional model defining a systematic approach for comparing offered and requested quality of services. Security is included as a static parameter to be matched.

None of the approaches allow security configurations to be modified on-demand according to user requirements. Security is always dealt with as a static take it or leave it attribute of the system.

## 6 Conclusions

This paper presented a WS-Agreement based approach for a fine grained security configuration mechanism to allow an optimization of application performance based on specific security requirements. A WS-Agreement schema is introduced containing security mechanisms for a Grid protected by a DMZ and offering end-to-end encryption. The Globus Security Infrastructure (GSI) was extended to cope with the DMZ and made configurable to dynamically allow performance optimisation based on WS-Agreement specifications. Different Sandboxing mechanisms can be easily selected depending on user requirements and site capabilities. Job data can be encrypted in a fine grained manner during all stages of transmission and storage in insecure networks. This novel Grid setup allows both academic and industrial customers to ensure the safety of their data to their required security level while at the same time allowing the Grid environment to optimized job performance by dynamically configuring the security mechanisms to suit the users needs. The additional overhead created by the security measures can now be controlled by the users, allowing a dynamic job based security/performance trade off decision to be made on a per job basis instead of a per site basis.

There are several areas of future work. Currently, the security specification document is specific to our Grid environment a more generic specification will be created using security features offered by other D-Grid sites. Furthermore the evaluation mechanisms are only partially automated future work will include automatic and semi-automatic tool-support for users and administrators to enable more convenient control. Also an integration with GridWay and our GridBPEL engine is planed to further automate Grid security configuration.

## 7 Acknowledgments

This work is partly supported by the German Ministry of Education and Research (BMBF) (D-Grid Initiative).

## References

[1] Access e.v., aachen. casts homepage. http://www.access.rwth-aachen.de/00010_00013_process_simulation.htm, May 2007.

[2] Mark Baker, Hong Ong, and Garry Smith. A report on experiences operating the globus toolkit through a firewall. Technical report, Distributed Systems Group, University of Portsmouth, September 2001.

[3] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, 2003.

[4] B. Carminati, E. Ferrari, and P.C.K. Hung. Security conscious web service composition. In *International Conference on Web Services*, pages 489 – 496, 2006.

[5] V. Casola, A. R. Fasolino, N. Mazzocca, and P. Tramontana. A policy-based evaluation framework for Quality and Security in Service Oriented Architectures. In *IEEE International Conference on Web Service*, pages 1181 – 1190, 2007.

[6] Bryan Clark, Todd Deshane, Eli Dow, Stephen Evanchik, Matthew Finlayson, Jason Herne, and Jeanna Matthews. Xen and the art of repeated research. *USENIX 2004 Annual Technical Conference*, pages 135–144, 2004.

[7] The GridSphere Developers. The gridsphere portal framework. http://www.gridsphere.org, May 2007.

[8] Niels Fallenbeck, Hans-Joachim Picht, Matthew Smith, and Bernd Freisleben. Xen and the art of cluster scheduling. *Super Computing 06, Virtualization Workshop*, page (to appear), 2006.

[9] T. Friese, M. Smith, and B. Freisleben. Native code security for grid services. page (accepted for publication), 2007.

[10] The Grid Security Vulnerability Group. Critical vulnerability: Openpbs/torque. http://security.fnal.gov/CriticalVuln/openpbs-10-23-2006.html, October 2006.

[11] P.C.K. Hung, Li Haifei, and Jun-Jang Jeng. Ws-negotiation: an overview of research issues. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, pages 10–20, 2004.

[12] D-Grid Initiative. D-grid integrationsprojekt. http://www.d-grid.de/, May 2007.

[13] M. Schmidt, M. Smith, N. Fallenbeck, H.J. Picht, and B. Freisleben. Building a demilitarized zone with data encryption for grid environments. In *Proceedings of First International Conference on Networks for Grid Applications*, page to appear, 2007.

[14] Matthew Smith, Michael Engel, Thomas Friese, Bernd Freisleben, Gregory A. Koenig, and William Yurcik. Security issues in on-demand grid and cluster computing. In *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, page 24, Washington, DC, USA, 2006. IEEE Computer Society.

[15] Matthew Smith, Thomas Friese, Michael Engel, and Bernd Freisleben. Countering security threats in service-oriented on-demand grid computing using sandboxing and trusted computing techniques. *J. Parallel Distrib. Comput.*, 66(9):1189–1204, 2006.

[16] Matthew Smith, Thomas Friese, and Bernd Freisleben. Intra-Engine Service Security for Grids Based on WSRF. In *Proc. of Cluster Computing and Grid*, pages 644–653, Cardiff, UK, 2005.

[17] Internet Security Systems. Unicore client keystore information disclosure. http://xforce.iss.net/xforce/xfdb/30157, November 2006.

[18] Globus Security Team. Globus security advisory 2007-02: Gsi-openssh vulnerability. http://www-unix.globus.org/mail_archive/security-announce/2007/04/msg00000.html, March 2007.

[19] Globus Security Team. Globus security advisory 2007-03: Nexus vulnerability. http://www.globus.org/mail_archive/security-announce/2007/05/msg00000.html, May 2007.

[20] Von Welch. Globus toolkit firewall requirements. http://www.globus.org/toolkit/security/firewalls/Globus-Firewall-Requirements-9.pdf, October 2006.