Grid on Demand

Alain van Hoof and Willem Toorop

alain.vanhoof@os3.nl
willem.toorop@os3.nl

Under supervision of Dr. Paola Grossa and Drs. Rudolf Strijkers

p.grosso@uva.nl
strijkers@uva.nl

Abstract

Currently, e-Science applications run on dedicated computing platforms. The scientific/e-Science computing community developed the Grid to share resources and maximize compute capacity available for e-Science applications.

Applications exist that incidentally have urgent high demand for compute capacity, for example in case of emergencies. Also, some researches need to do experiments with Grid computing itself. For such cases, the Grid is not ideally equipped.

Recently Cloud computing emerged, providing on demand compute resources on a pay-per-use basis. Clouds promise infinite virtual resources. This report addresses the utilization of (additional) resources offered by Clouds for Grid applications. To this end, a working implementation of a Grid enabled Cloud compute cluster is presented and assessed.

The research question this report tries to answer is:

"Can Grid computing be offered as a Cloud service?"

University of Amsterdam

System & Network Engineering *Research Project 2*

Contents

1 Introduction

Grid computing has traditionally been the dedicated computing platform for compute intensive scientific (e-Science) applications. Grid is developed by the Grid community to share resources of the existing dedicated compute clusters and to enable large-scale e-Science experiments. Because of Grid those applications and experiments potentially can leverage the sum of the compute capacity of all participants. Still, orchestrating fair sharing among the participants is a complicated affair. Users of the Grid might sometimes need more than they have immediately available.

Cloud computing is emerging and offers compute resources as a service. The Cloud compute service is seemingly provided without limits, it promises an almost infinite number of resources that can be added and removed dynamically. Large scale e-Science applications with a sudden demand for compute power (urgent computing) can benefit from Cloud's immediate availability of resources and the predictability of the type of resource leased.

Combining the properties of both Grid and Cloud for the purpose of supporting current or newly developed e-Science applications with a sudden demand for compute power, this report investigated the following research question:

Can Grid computing be offered as a Cloud service?

One example that comes to mind is the IJkdijk[1] project and the closely related UrbanFlood[2] project. These projects have a extra and high demand for resources when calamities happen. For the daily operation of these projects it is unfeasible to invest in the resources needed. But when a dyke breaks extra resources are needed to calculate the effects of, and predict the flooding. The ability to quickly, but temporarily, utilize massive amounts of compute power to have the result as quickly as possible, would under such circumstances be very desirable. An on demand Grid could provide this ability.

Besides the support for e-Science applications, developers of Grid related software/application can create a test bed on demand with unlimited size (in both the number of Grid resources and compute nodes).

This report investigates the possibility of running a Grid resource based on the Globus Toolkit on the Amazon Elastic Compute Cloud. Section 2 addresses the definitions of "the Grid" and "the Cloud" to properly understand their meaning and it introduces the term Urgent Computing in the e-Science environment. Section 3 presents an implementation of a Grid resource on the Amazon Elastic Compute Cloud as a mean to study what is involved in such an implementation. The developed implementation is then assessed using a "real life" biomedical application and a Workload Scheduling system developed by the SNE research group of the UvA in section 4. Finally, our conclusions are presented in section 5.

[1] http://www.ijkdijk.nl
[2] http://urbanflood.eu

2 The Grid and The Cloud

Both the Grid and the Cloud where created with the same goals: Sharing resources to minimizing idle resources. This sharing of resources increases computing power and storage capacity by utilizing the aggregated resources. The Grid shares the resources among equipollent stakeholders that administer the resources that are their own. In contrast, with Clouds, the resources are under the administrative control of the Cloud provider and are leased out to users for a fee. These differences in the relation between the users and providers with Grids and Clouds have led to different architectural properties (or features)[26]. The following sections define the Grid and the Cloud properties and services, describe the resources of interest for e-Science applications and ends with a summarizing overview. What we think can be created with a Grid in the Cloud is in section 2.5. The last two sections shows where this Grid on Demand is useful.

2.1 The Grid

Since the introduction of the term Grid different definitions have been in use. To end the debate, a definition was defined by Ian Foster in "What is the Grid? A Three Point Checklist" [8]. This definition has been widely accepted by the Grid community and is as follows:

> "Coordinates resources that are not subject to centralized control using standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of service."[8]

Traditionally the Grid is used for compute and storage intensive applications. Often high bandwidth within the Grid is a requirement for these applications. A term created for this type of applications is e-Science.

To coordinate the sharing of resources between different users the concept of Virtual Organization (VO) has been created (figure 1). This enables global distributed collaborations involving large numbers of people. It makes data and computing intensive scientific experiments feasible.

2.1.1 Grid services

In the informational document "The Open Grid Services Architecture, Version 1.0." by I. Foster et.al. [16] an extensive list and description of the services a Grid provides are given. The following list is an excerpt.

Security Service The main service offered by Grid is the support for the distributed collaborations to address large-scale computation problems. The certification infrastructure is not only a procedural implementation, a Grid is running a authentication and authorization service.

Data Service This services places (move or copy) data to where it is needed, it manages replicated copies, runs queries and updates, and can transform data into new formats.

Execution Management Service The management of Jobs from start to finish. This involves selecting the resources, preparing the resources for execution and monitoring job execution.

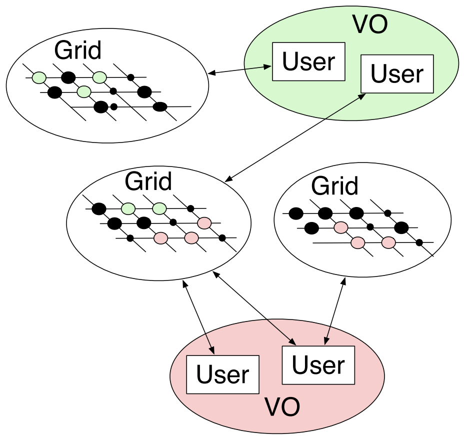


Figure 1: Users in a VO can use part of the shared resources

Resource Management Service The service that manages the resources themselves. On the node level but also on the Grid level and even higher levels.

Self-Management Services To reduce the complexity and cost of own resources this service tries to perform self-configuring, self- healing and self-optimizing of the resources.

Information Services Allows access and manipulation of information about applications, resources and services. Quality of Service is a main concern of this service.

2.1.2 Properties of the Grid

The list in hereafter describes the properties of Grid that we think are important. The same properties are described for cloud in section 2.2.3.

Heterogeneity The characteristics of the resources offered differ per resource group (Grid), this allows applications to run in a environment especially suited for that application.

Sharing of resources Storage, Memory, Compute and Network are shared using a certificate infrastructure where users are in VOs that have access to a group of resources (Grids).

Scalability By combining Grids, which are often already large systems, even larger systems can be created. The certificate infrastructure takes care of the combining of resources.

Elasticity The time to add access to an extra Grid is within day's and under control of the certification infrastructure.

Device and location independence Resources are on and off-site and accessed via dedicated networks or Internet. Traditionally access is via a workstation or via the Grid itself.

Reliability Grid can offer redundancy when multiple Grids with the same characteristics can be accessed. Due to the heterogeneity of the grids this can be a challenge

Maintenance Part of the resources are on-site and need maintenance from the Hardware level and up.

Cost To share resources of a Grid (and therefore get access to other resources) a Capital investment has to be made. Maintenance fees, administration cost and utility (power/cooling/floorspace) cost exist.

2.2 The Cloud

In “The Business Perspective of Cloud Computing- Actors, Roles, and Value Networks” [19] a number of papers and authors defining the term Cloud are combined to the following definition:

> ”An IT deployment model, based on virtualization, where resources, in terms of infrastructure, applications and data are deployed via the internet as a distributed service by one or several service providers. These services are scalable on demand and can be priced on a pay-per-use basis”[19]

Unlike Grid where VOs are used, in Clouds access to resources is provided by a Cloud provider often by means of a financial transaction (public Clouds section 2.2.2). On a pay-per-use basis users can access a large number of resources provided by the Cloud provider.

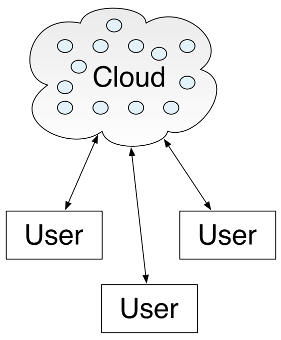


Figure 2: Users can use (a part of) the shared resources in the Cloud

2.2.1 Cloud services

There are many different levels of services. The paper “Above the Clouds: A berkeley view of Cloud computing” [1] define Software as a Service (SaaS) and Infrastructure as a Service (IaaS). Besides elaborating on a comprehensive definition of Cloud computing using a vast amount of other papers/authors Leimeister, Riedl, Böhm and Krcmar [19] additionally define the terms Platform as a Service (PaaS) and Hardware as a Service (HaaS). The services have a layered dependency, and can be displayed (figure 3) in a layered model where HaaS is the lowest layer and SaaS the highest level.

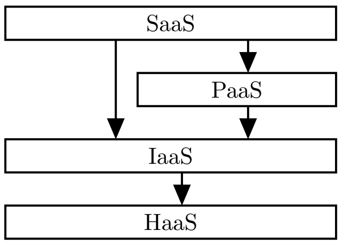


Figure 3: Layered Cloud services

The layered services of Cloud in Figure 3 are described in more detail below.

SaaS Off-the-shelf Applications are provided to the client. Small adjustments to the application are possible (logo or workflow for example).

PaaS A framework or software system (middleware) to build and deploy application. PaaS service are often designed for a specific SaaS application. Can provide middleware (databases), mostly of interest to developers.

IaaS Using the provided virtual hardware platform a client can create its own server environment or in other terms virtual hosting on demand. The client is in control of all layers above the virtualization layer.

HaaS The physical computer infrastructure. Traditionally offered as an outsourcing service.

2.2.2 Public, private and hybrid Clouds

Cloud services can be offered as an commercial product by an external company. When an external company acts as a Cloud provider this is called a public Cloud. A Cloud is called private when the Cloud services are offered with in the company itself by for example the IT-department. A combination of both a private Cloud and public Cloud, for example to a resources not available in the company when the resources are needed, is called a hybrid Cloud.

2.2.3 Properties of the Cloud

In the commercial environment of Cloud computing a great number of terms are used to describe the properties of Cloud. The following list describes a number of properties that are also mentioned as the properties of Grid (section 2.1.2) but the description differs as the implementation of the property differs.

Homogeneity The properties of the resources offered via IaaS are all very similar. This is due to the virtualization of the Hardware platform. Often a limited list of different type of virtual machines is offered.

Sharing of resources Storage, Memory, Compute and Network are shared by means of a financial transaction using ”the more you pay the more you get” principle.

Scalability Expand and shrink resources as needed. The expansion is seemingly without limit but in practice a cloud provider has limits.

Elasticity Easy and fast addition and removal of the resources. The time to add or remove a resource is within minutes and under complete control of the user, either via a webinterface or a Application Programming Interface (API).

Device and location independence Resources are off-site and accessed via the Internet, so users can connect from anywhere with any device. This property implies security issues.

Reliability Cloud service providers offer redundancy often with multiple datacenters located globally. When using IaaS the user has to provide the logic for the redundancy.

Maintenance Resources are in the Cloud and often accessed via a web-bowser, so minimum client application maintenance is needed. When using IaaS for example no hardware maintenance is needed.

Cost No capital investments and "pay by use". When using IaaS the cost for system administration still exists but the cost for hardware maintenance will disappear.

2.3 Resources

e-Science applications have need for basic resources both provided by the Grid and the Cloud. The Cloud can offer different types of services but the resources needed by Grid applications are almost always offered as IaaS. Table 1 shows the shared resources.

Resource	*Capacity*	*Shared as*
Processor	Flops	Compute power
Hard-disk and Memory	Bytes	Storage space
Network	Bytes/sec	Communication Bandwidth

Table 1: Resources shared by the Grid and the Cloud

2.4 Overview

A side by side comparison of the most important properties of both Grid and Cloud is done in table 2. A more detailed description of the properties is given in the sections of Grid (section 2.1.2) and Cloud (section 2.2.3).

Grid and Cloud (IaaS) both provide the same resources (table 1). The quality of service of these resources differ for Grid and Cloud. The sharing of the network resources in combination with the virtualization layer added by IaaS is a known issue in Cloud computing [10]. The e-Science applications where a high quality of service of the network is needed due to inter node communication are probably not suited to run on a Cloud IaaS. The other resources (Compute/Storage/Memory) have a much better quality of service in

Property	**Grid**	**Cloud (IaaS)**
Distribution	By Collaboration	By Cloud Provider
Resource Type	Heterogeneous	Homogeneous
Abstraction	Middleware (open standards)	Virtualization
Scalability	Combine existing resources	Add/Remove resources on request
Elasticity	Add resources via procedures (days)	Add/Remove resources very fast (minutes)
Sharing Guarantee	Security infrastructure	Financial infrastructure
Cost	Initial(investment) and power/cooling/space	Pay-per-use (hour)
Maintenance	Hardware maintenance	No hardware maintenance

Table 2: Properties of the Grid and the Cloud compared

the Cloud but due to the use of dedicated hardware in the Grid these resource will not (yet) reach the same level as with Grid.

2.5 A Grid in the Cloud

The properties of the Grid and the Cloud have similarities and differences (see section 2.4). For the purpose of e-Science we introduce a concept called *Grid on Demand*. Using the elasticity and scalability of Cloud computing (IaaS) and providing the abstraction of a Grid interface on top of the virtualization of Cloud, current and future e-Science applications can use a Grid on Demand when there is a high demand for resources for a short period of time. The access to the Grid, the Grid interface and the underlying dynamic cluster all are running in a Cloud environment and we therefore argue that *Grid on Demand* is "a Grid in the Cloud". Due to the "pay-per-use" model and the lack of initial investment cost of Cloud resources, a user can run a Grid on Demand with minimal budget. The financial implication of running a Grid on Demand are always known to the user. The pricing model of the Cloud provider is a known fact and can be used to calculate exact cost.

2.5.1 Urgent computing

As stated before Grid on Demand is an elastically scalable solution for e-Science applications. There are many different kind of e-Science applications and they run well on a Grid environment. When there is a sudden and high demand for resources by an e-Science applications, Grid has difficulties providing these immediately. It is this need for "urgent computing" that a Grid running in the Cloud can provide. The example of UrbanFlood[3] that wants to calculate flooding predictions when actual flooding happens was already given in the introduction. The biomedical application (WAVE section 4.2) used in the performance test of Grid on Demand is an other example. A patient has to wait for the simulations to finish before a surgeon can operate. Fast results will reduce

[3] http://urbanflood.eu

waiting time and allows the surgeon to operate soon (hours as opposed to days) after the patient had the initial sonic-scan. A Grid on Demand provides the temporary but high volume of compute resources needed in both examples.

2.5.2 Grid application development

A Grid on Demand is under complete control of a user. When that user is a developer of Grid related software like a scheduler (section 3.3) or a workflow manager like WS-VLAM (section 4.3) the user has a Grid testbed at its disposal. The developer can create a testbed on demand with unlimited size in both the number of Grids and compute nodes. There is no restriction to start more than one Grid on Demand with a number of compute other than the maximum allowed by the Cloud provider.

3 Implementation

This section presents the implementation of a Cloud based compute resource to study what is involved in incorporating such a resource in the Grid.

First we will sketch a complete picture of the implementation that we created in 3.1, because we believe this contributes greatly in understanding the issues that were involved in its implementation, which are subsequently addressed. Globus Toolkit as the implementation choice to enable the resources Grid participation follows in 3.2, with a detailed description of the protocol and mechanisms involved to participate in the Grid in 3.2.1. This study leads to two implications:

1. A mechanism to deliver or store context is necessary for the implementation (3.2.2).

2. Such a Cloud based Grid resource has limitations on its usage (3.2.3).

Possibilities to address the limitation on the usage are also presented in 3.2.3.

We continue the section with presenting the Torque Resource Manager in 3.3. From the study of the implications of its implementation, again the need to deliver or store context emerge (3.3.1). How previous studies similar to ours addressed this necessity is evaluated in 3.4. Then we introduce Amazon Elastic Compute Cloud focusing on the recently added feature Amazon Elastic Block Storage that enables persistent disk storage of the Virtual Machine instances in 3.5. Finally, combining all the described ingredients, we present the architecture of our implementation in 3.6.

In Appendix A, the usage is demonstrated describing the different administrative procedures involved with our implementation.

3.1 A sketch of Grid on Demand

Our implementation is realized by extending an existing Amazon Machine Image (AMI), containing the Ubuntu Lucid Linux operating system, with Torque Resource Manager (Torque) and Globus Toolkit (GT). An initial instance of the AMI operates as the cluster's head node. This head node is also configured as an initial compute node. A new Certificate Authority (CA) is created with which an initial host and Grid user certificate are generated.

The instance can be configured further into an elastic cluster by providing Amazon Access Credentials. Dynamically added compute nodes will be based on the same AMI as the head-node. Additional parameters can be set to specify when new compute-nodes should be launched and of what instance type they should be. Taking into account Amazon's hourly payment model, cost is minimized by automatically terminating compute-nodes when they are idle and their uptime reaches a whole hour.

Configuring and monitoring of the cluster, and Grid user administration, is offered through an easy to use web-interface.

Amazon Elastic Block Storage (EBS) is used as the storage mechanism for the AMI to accommodate the administration of different stages in which the grid can be configured. Using EBS, a configured instance can be shutdown in two different ways: for reuse, and for sharing.

- When shutdown for reuse, the newly created CA, user certificates and settings concerning the dynamic cluster (including the Amazon Access Credentials) are preserved. The stopped image may later be restarted already configured with a specific CA and cluster settings. On restart, only a new host certificate will be generated for the new hostname the instance will get.

- When shutdown for sharing, the CA, host and user-certificate and the cluster settings will be removed, so that on startup all those settings will be regenerated and ready for configuration.

Using EBS, a stopped instance can be used to create a new AMI based on the content of the volume of the stopped instance. The two shutdown methods allow to further extend the original AMI with desirable software and configurations (perhaps to accommodate specific applications) for personal use, but, by shutting down for sharing, may also be offered to the public without revealing credentials (for using the personal Grid and for using the Amazon Web Services).

3.2 The Grid: Globus Toolkit

The Globus Toolkit (GT) [13] is a open source toolkit developed by the Globus Alliance for building Grids. GT is very popular software and widely deployed. It makes use of open standards such as defined by the Open Grid Services Architecture (OGSA) developed within the Open Grid Forum (OGF), which are also used by other Grid software. OGSA and other open Grid standards find their origins in the Globus Alliance.

GT is composed of different tools, services and components laid out on different layers following the principles of the "hourglass model" [5]. In a "hourglass model" a diverse multiplicity of higher and lower level services are able to inter operate through a small set of protocols in the middle: the narrow neck of the hourglass.

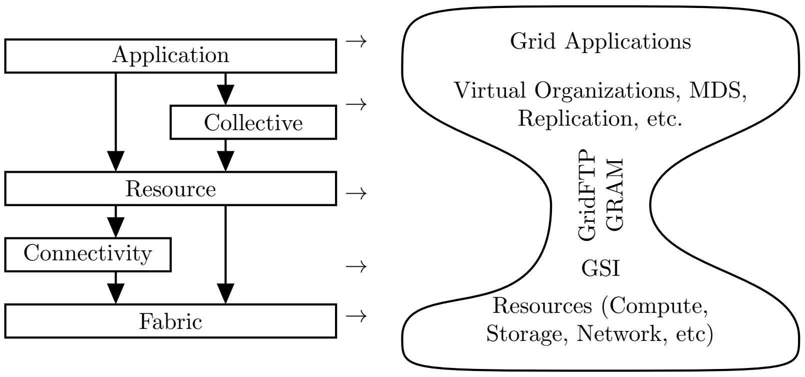

Application
Collective
Resource
Connectivity
Fabric
Grid Applications
Virtual Organizations, MDS, Replication, etc.
GridFTP
GRAM
GSI
Resources (Compute, Storage, Network, etc)

Figure 4: Layered Grid protocol architecture. The downward arrows indicate the usage of an underlying lower layer by a layer above. In the hourglass on the right, the type of services (and the services themselves) are shown on the same level as the layer they belong to.

In figure 4 the layered architecture that has been the model for GT is shown. The narrow neck is established by the Connectivity and the Resource layer [11]. From the bottom-up, the following layers are defined:

Fabric On this layer are the actual resources being shared and used in Grid collaborations. GT has been designed to use existing software as much as possible. For example existing cluster resource managers that can allocate compute-nodes to process batch jobs will, when available, be used as Compute resource. Also, storage systems that utilize the network, such as the Network File Systems can be used as underlying storage resource.

Though, if underlying software to handle basic resources is not available, GT can provide the missing functionality[4].

Connectivity On this (narrow neck) layer the authentication and authorization framework, that is essential for the sort of collaborations which the Grid provides, are realized. GT uses the open Grid Security Infrastructure (GSI) standard. GSI has important implications for our use-case implementation which will be explained later on.

Resource This (narrow neck) layer delivers access to the resources offered by the Fabric layer in a uniform standards based way. The Gird collaborative modus operandi is established by giving access through the Connectivity (and thus security) layer.

In figure 4, two services are shown: Grid Resource Allocation Manager (GRAM) and Grid File Transfer Protocol (GridFTP), which address management and usage of Compute and Storage resources respectively.

The layer is concerned with the information and managements of single resources. Information and management of multiple resources is in the Collective layer.

Collective Protocols and services in this layer are concerned with the discovery, allocation and coordination of multiple resources. A canonical example is the Monitoring and Discovery System (MDS). It can be used to discover what resources are considered part of a VO and to monitor those resources.

The Collective layer hosts a diverse set of services and protocols, specifically targeted at multiple Grid resources, such as (besides Monitoring and Discovery): Data replication, Community authorization and Collaborative services. These type of services especially concern members of VOs, who may access resources which they do not now beforehand.

Application This layer finally is where the Grid applications, using the Grid services in the underlying layers, reside. GT offers APIs and Software Development Kits (SDKs) on the Collective and Resource layer for programmers of Grid applications.

This "hourglass" architecture implies that the Resource and Connectivity layer provide the minimum Grid services that all Grid resources should have in common. The Collective layer enables for more diverse, different, more directed towards specific use-cases configurations of specific Grid deployments.

[4] Although it has no native support for cluster configurations

In our use-case implementation it would be desirable to only implement the minimum Grid services (on the Resource and Connectivity layer), delivering the most generic Grid resource, not configured towards specific usage. GT is composed as a loosely coupled set of tools, services and components, allowing to realize such a deployment. Of course, on the Fabric layer, still choices have to be made, non the least the choice to use IaaS in the Cloud as underlying Virtual Machines.

Because of GT's prominence in the Grid world, its usage of open standards which makes it highly inter-operable with existing Grid deployments and its modular architecture, allowing us to select only a bare minimum of components to create a generic Grid enabled resource, have led us to take GT as the software of choice for our use-case implementation.

3.2.1 Configuring

Our implementation of GT will be installed on a Virtual Machine image. Different users might instantiate the image to operate as a Grid resource in different environments. Dynamic configuration of the Grid resource towards those environments is needed.

Configuration of GT is mostly concerned with GSI. It is useful to have a closer look at the workings of GSI to appreciate the affairs involved. To this end we will regard a typical Grid usage example. In figure 5 such an example is illustrated.

The Grid User wishes to do an computation on a certain set of data. The type of the computation is completely parallel, and the more resources work on the computation the quicker the result will come in. The Grid User is authorized to use Grid resources in four different organizations. It has the data, on which the computation should be done, stored at a Storage resource offered by Organization A. It instructs the Scheduler resource at Organization C, to perform the computation on the data stored at the Storage resource at Organization A. The Scheduler knows about two Compute resources at two different organizations (B and D) which are offered to the Grid User (perhaps because of his VO membership). It splits the computational job in two, and delegates the parts to the two Compute resources. The Compute resources access the data at the Storage resource and perform the computation. When finished, the Scheduler then combines the results and returns that to the Grid User.

With Grids, authorization to use certain resources is offered to the Grid *users* (or VO members). The resources themselves are *not* authorized to use other resources. The data on Storage is available to the Grid User and not to the Compute resources. To realize this GSI has been developed which introduced the concept of *delegation*.

When the Grid User instructs Scheduler to do its task, is delegates also the permission to access the data from Storage and to perform computations *on behalf of* the Grid User. With its delegation, Scheduler may then further delegate permissions. In our case Scheduler further delegates the permission to access the Grid User's data from Storage to the Compute resources.

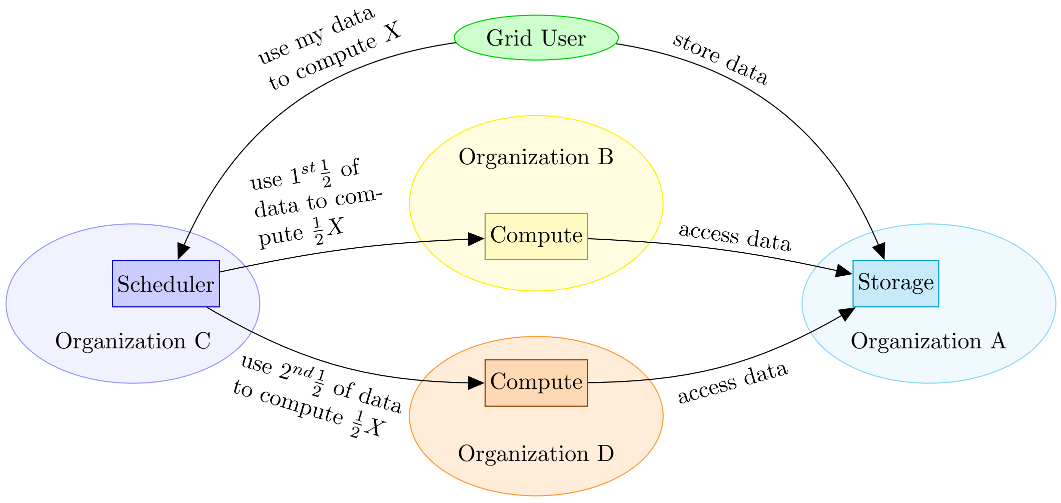


Figure 5: Typical Grid usage example spanning multiple organizations

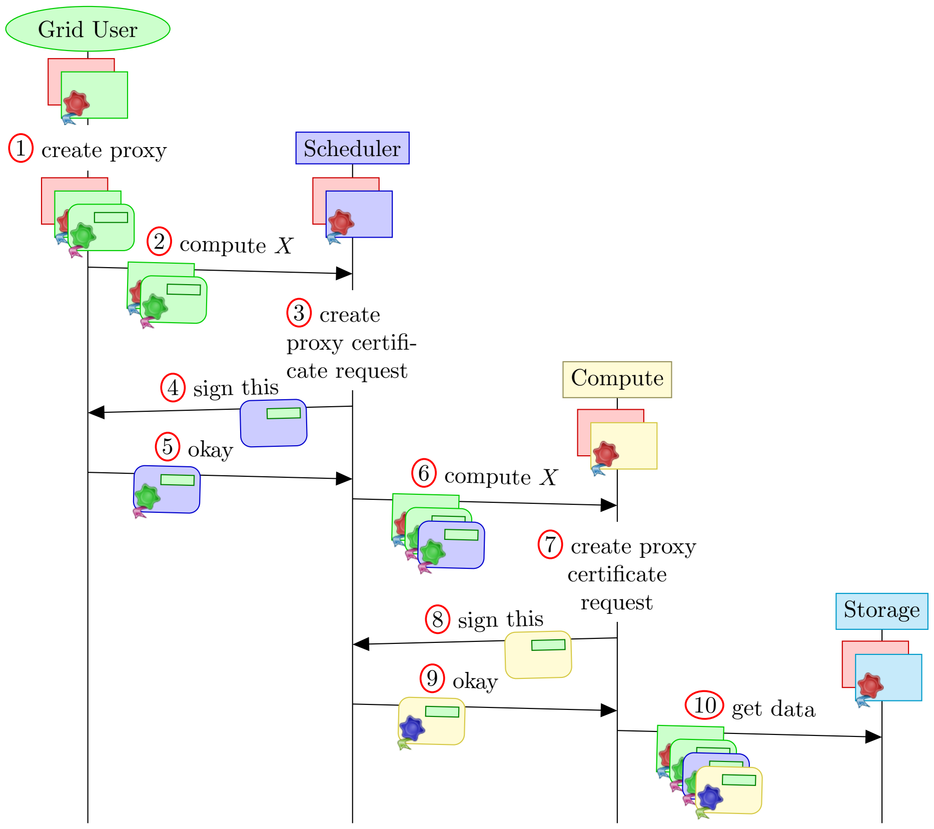


Figure 6: Proxy certificates

To accomplish delegation the Globus Alliance has developed a mechanism using Public Key Infrastructure (PKI), where Public Key End Entities (such as users or services) can request the signing of a Proxy Certificate (PC) by another Public Key End Entity or another PC to be able to authenticate as the signer for specific by the signer determined purposes.

In figure 6 is illustrated how PCs accomplish the delegation needed in the usage example in figure 5. We assume that the Grid User has its data already stored at Storage. Furthermore, all Public Key End Entities (the Grid User and the resources) already have a CA signed certificate.

1. To be able to authenticate to any Grid resource, the Grid User first creates a PC for itself. In a way it delegates the ability to authenticate as itself to itself. In this way, authentication in the Grid can be handled in a uniform way: by PC path validation. It also has the advantage that the password protected private key of the CA signed certificate of the Grid User needs to be unencrypted (for which the password is asked) only once: to sign the PC request.

   The private key of the PC is not password protected, and secured by the precautions of the Operating System only. The PC has only limited lifetime (typically one day) which alleviates the damage when compromised considerably.

   The subject of a PC consists of the subject of the issuer (the signer) appended with a Common Name (CN) attribute with an unique value for the purpose of the delegation. In this way, the subject always starts with the subject of the End Entity for which the delegation is created, appended with a series of CN attributes reflecting the number of times the certificate is delegated. This will be further illustrated in the following successive steps below.

   The green rectangle in the certificate, reflects that the subject starts with the same value as that of the Grid User's End Entity Certificate (EEC). The issuer of the certificate is indicated with the "Seal of Approval" symbol ( ) in the color of the End Entity that actually signed the certificate. But, be aware that the subject of the EEC of the End Entity that is the signer of a PC, does not have to be in the issuer field of the certificate as will be further illustrated in the following steps bellow. Though, in this initial PC, the issuer field *is* the same as the value of the subject field of the EEC because it is actually signed by the EEC and not by another PC.

2. The Grid User contacts Scheduler to submit a task (to compute X on its data at Storage). The Grid User and Scheduler are mutually authenticated with Transport Layer Security (TLS). The Grid User with its PC and Scheduler with its EEC.

   The Scheduler's EEC should be signed by a by the Grid User recognized CA. The International Grid Trust Federation (IGTF) distributes a package that contains the CA certificates of all well known accredited CAs [5].

[5] Root certificates of well known Grid CAs:
https://dist.eugridpma.info/distribution/igtf/current/

3. Scheduler now creates a process to perform its assigned task. It knows a Compute resource best suitable to perform the task, which is accessible by the Grid User and currently available. To delegate the task to that resource it needs to be able to act on behalf of the Grid User.

   Therefor Scheduler creates a new PC request. Because the PC is intended to inherit the identity of the PC of the Grid User, it has as the subject, the subject of the Grid User's PC appended with a CN attribute unique for the purpose of the delegation. Although, in figure 6 the PC request is colored blue, no information in the PC request indicates that it is Scheduler who created it. The color is just to indicate that it is Scheduler who has the Private key associated with that PC request.

4. Scheduler sends its new PC request to the Grid User for signing over the mutual authenticated TLS channel. The Grid User can therefor be sure that it is Scheduler who requests the delegation.

5. The Grid User signs the PC request with its own PC. The "Seal of Approval" symbol is green, to indicate that it is the Grid User's PC that signed Scheduler's PC request. The issuer in the signed PC (which the Seal represents) has thus the subject of the Grid User's PC. Nothing in Scheduler's PC indicates that Scheduler holds the Private key associated with it.

   Note that the Seal on Scheduler's PC is different from the one on the Grid User's PC (although both green), which is signed by the Grid User's EEC. The colors of the Seals in figure 6 reflect who holds the Private key that created the signature.

6. Scheduler contacts it chosen Compute resource and asks to perform the task (compute X). Scheduler and Compute are mutually authenticated with TLS. Scheduler with its PC and Compute with its EEC.

   Nothing in the PC of Scheduler indicates that it is Scheduler who contacts Compute. To Compute, it is as if the Grid User just created a series of certificates, each signing the next.

7-10. These steps are similar to those of three till six. To perform the task, Compute needs to access data at Storage. Thus, it creates a new PC request, with as subject that of Scheduler's PC appended with a CN. Scheduler signs it with its PC (setting the issuer field to the value of the subject field of its own PC) and returns it.

10. Although the private keys of the stack of certificates are with different parties (reflected in the different colors), the value of the subject field starts for all the same: with the value of the Grid User's EEC's subject (at the bottom of the stack). All issuer fields also start with this value (although the private keys that made the signature are with different parties, reflected in the colors of the Seals), except the first (the EEC) that has the value of the subject of the CA.

> We have repeated steps three till six in seven till ten to illustrate that even the signing with a private key not hold by the Grid User makes no difference in the perception of the PC by a resource. Compute's PC has been signed by Scheduler, but this involvement is completely hidden to Storage. To Storage the PC presented by Compute might just as well have been a series of certificates all generated by the Grid User itself.

In the example the service offering resources and the user of the resource (which might also be a service at another resource) are mutually authenticated using TLS. The user is authenticated using proxy certificate path validation. The fact that at the end of the path, an users EEC is signed by a CA established the identity of the user.

But just path validation of the EEC of the resource does not suffice to establish the identity of the resource by the user. An EEC might be signed by a CA to be used for a specific resource, but what prevents the holder of that EEC to use it for a completely different resource?

The TLS specification [6] does not specify how to establish a resource's (or server's in TLS terminology) identity. This is left to the specification for the applications using TLS. However, it is common practice that applications establish a server's identity based on their dns name. Currently a draft is evaluated by the Internet Engineering Task Force (IETF) that generalize this approach in a "Best Current Practice" standard [24]. GT also uses the dns name of a resource to validate its identity.

3.2.2 Implications for the architecture of the implementation

The usage of an EEC intended for the resource's host to authenticate a resource has important consequences for our implementation. A newly instantiated Grid resource (in a IaaS cloud) will normally have a IP-address and dns name assigned to it dynamically. To create an EEC for the just created Grid resource using normal procedures would take a lot of time (perhaps several days) and involve procedures that require manual interposition of the user creating the dynamic Grid resource. To have the instance running and waiting for the certificate would be very unserviceable and not in line with the dynamic nature of Cloud. Augmenting the Grid with the dynamism of the Cloud is one of the primary goals we are pursuing with this project.

Amazon sells a service to launch instances with static, account associated, IP-addresses: Amazon EC2 Elastic IP Addresses (EIP). When an image would always be instantiated with the same IP-address (and dns name), it may be equipped with a valid EEC signed by a widely accepted CA. Such a Grid resource will be able to fully participate in bigger Grid collaborations and all the benefits of using GT will stand out well.

However, this will not be a convenient solution when a Grid resource is needed immediately and the procedure of buying a static IP address and requesting a host certificate has not been completed. Also, for an experimental temporary Grid resource, this would be too cumbersome.

Alternatively, each owner of an account for IaaS with a Cloud provider, may take the role as CA. Since this person is also launching the instances (directly or via a service) it may provide the instance with the contact details of a "context broker" which may be used by the newly created Grid resource to intermediate with the signing of its EEC request[6]. Or a persistent storage could be used accessible for the instance in which the private key of the CA is stored, so that the instance can sign its own EEC request[7]. In any case, such a “self signed” resource will not just-like-that be able to collaborate in all the different possible ways as a static resource with an EEC signed by a widely accepted CA as will be explained next.

3.2.3 Implications for the usage of the implementation

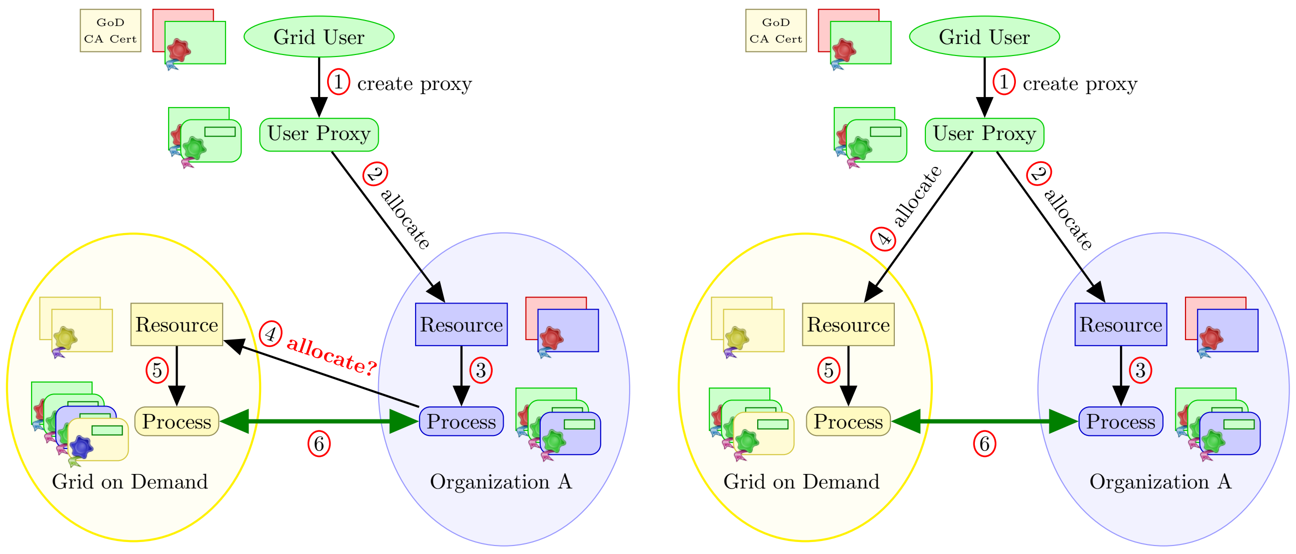


Figure 7: Delegation using Proxy Certificates. On the left the type of usage example illustrated in [14, 4]. On the right a RFC 3820 [25] type of example.

The use case descriptions in the papers [14, 4] that have led to GSI describe mechanisms in which a process is delegated to use a resource that is authenticated by its EEC. This limits the way a Grid resource with an self signed EEC can be used in bigger Grid collaborations. The left site of figure 7 illustrates the problem. The Grid User has instantiated a Grid on Demand, and wishes to use it through a Resource. Resource might for example be a scheduler that may address many different computing resources in many different organizations and also (if all else is occupied) the Grid on Demand. Because the scheduler Resource does not know the CA that signed the Grid on Demand Resource’s EEC, it is unable to validate the identity. Even though the scheduler Resource acts on behalf of the Grid User that created the Grid on Demand Resource in the first place.

In the standard for Proxy Certificates, RFC 3820 [25], another approach to delegation is given[8], which is illustrated on the right side of figure 7. In stead

[6] This approach is taken by “one-click” virtual clusters [17] which will be covered in 3.4.1
[7] This is more or less the approach we have taken
[8] The example is given in section 2.3 of RFC 3820 the third option under “*Proxying can be used to secure all of these interactions*”.

of letting the Resource instantiate a Process allocating the Grid on Demand Resource, a Process is created that *waits* for the Grid on Demand service for the Grid User (in the form of the by Resource created Process) to become available[9]. Both processes may then mutually authenticate using the PCs that represent the Grid User. The PCs in use by the processes have the Grid User's EEC at the end of the certificate chain signed by a well known CA.

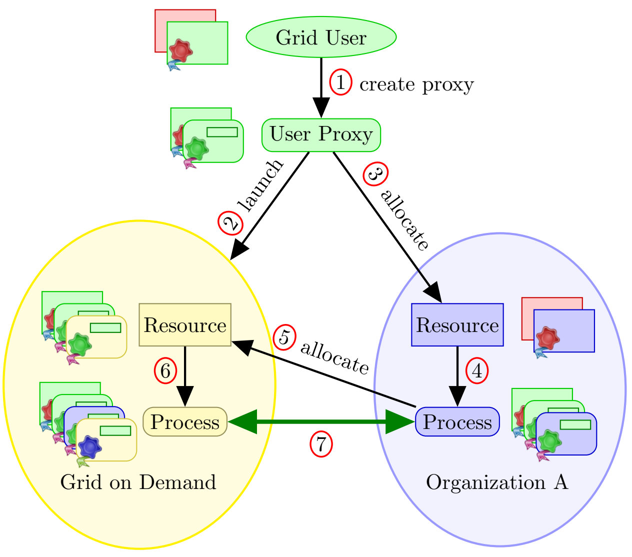


Figure 8: Our suggestion

As an alternative, we can also imagine that resource identity validation might be extended in such a way that resources are considered legitimate by a process, when it uses a PC (in stead of an EEC) representing the same user as the PC of the process. Dynamic resources could then be equipped with PCs signed by the EEC of the user that instantiated the dynamic resource, and would be usable in bigger Grid collaboration, but only for the user that instantiated the dynamic resource (see figure 8). We argue thus, that GSI should be extended to not only work for resource allocation, but also for resource instantiation.

In practice, it would for now be best to only use a dynamic resource directly, or through tightly by the user controlled resources, or one could buy a static IP address for the resource and go through the procedure of requesting a host certificate with the local Grid CA.

[9] This may happen in different ways. Process could contact Process to inform it about its existence, or the Grid User could inform Process about Process.

3.3 The Resource Manager: Torque

Computing resources in the Grid are often not mere simple single computers, but clusters[10] of cooperating computers orchestrated by a Job Scheduler. Such a scheduler typically runs on a cluster's "head node" and deploys "jobs" on "compute nodes" (also called "worker nodes"). The scheduler maintains one or more queues in which jobs may be submitted. Different properties such as priority to run, or authorization requirements, may be set on different queues.

The jobs themselves are sometimes called batch jobs, because their nature is often a bulk of certain tasks that should be performed without manual intervention. However, modern Job Schedulers also allow for interactive jobs.

Interaction between jobs can be performed in different ways. Message Passing Interface (MPI) is the dominant communication model used within clusters and supercomputers. For communication outside a cluster, Grid middleware such as GT is more appropriate because of its organizations-transcending distributed model; GT has services and tools for registering and discovering jobs and the actions and services they deliver to facilitate this.

Our implementation aims to behave as a compute cluster Grid resource. Different from ordinary static clusters, we will exploit the elastic nature of the Cloud, and launch more compute nodes when needed similar to the implementation described in [21].

GT natively supports three different Job Schedulers at the Fabric layer: Condor, Load Sharing Facility (LSF) and Portable Batch System (PBS). PBS is an interface specification that has multiple implementations, among which the Open Source Torque Resource Manager (Torque). GT has also third party support for the Oracle Grid Engine (OGE).

We had no preference for a specific Job Scheduler other then that its configuration should be as simple as possible. Torque is the Job Scheduler used by [17] (an implementation of a cluster in the Cloud similar to ours which we will address in 3.4.1) and showed up as the Job Scheduler of choice in online tutorials[11] on setting up a Grid resource with GT .

Torque has a basic Job Scheduler that deploys a maximum fixed number of jobs per compute node, possibly based on the number of cores of that node. It also supports more advanced Job Schedulers, that monitor load on compute nodes and schedule jobs minding the available compute capacity. We have not looked at those advanced schedulers, but we believe it would be a worthwhile improvement because virtual machines in an IaaS Cloud are provided by hardware that is likely to also be in used by other parties.

3.3.1 Implications

Compute nodes in a Torque cluster run a service that executes, monitors and manages the execution of jobs on that node: the Machine Oriented Mini-server (MOM). MOMs are registered and directed from a Server process on the head node. The Job Scheduler interacts with this Server to make scheduling decisions on the jobs waiting and running in the queues, which, with Torque, are also handled by the Server process.

[10] or even supercomputers
[11] For example `http://www.globusconsortium.org/tutorial/`

For a dynamic cluster this means that dynamically instantiated compute nodes need to know how to contact the head node. Also, they need to be registered with the head node. The head and compute nodes need to be aware of the "context": being the cluster of which they are part.

Informing the compute nodes of the head's location and vice versa, could be handled by an external "context broker"[12]. Monitoring of the job queues and spawning of new compute nodes when needed however, does not fit this model. Since the queues are on the head node, this process should either be on the computer that also instantiated the head-node where the credentials for spawning instances reside. Or it should be on the head-node itself. It the latter case, it is important to have a storage mechanism for the credentials needed to launch new instances specific for the user, similar to the requirements for the storage of the private key for the CA for the Grid on Demand as described in 3.2.2. Alternatively, the head node could reside on a static node outside the Cloud, so that no storage mechanism for personal settings (and credentials) is needed[13].

There is an oddity which we have to take into account with our implementation. Although Torque is perfectly capable of staging in and out jobs and redirecting input and output using Secure Copy (`scp`), in conjunction with Grid Resource Allocation Manager (GRAM), a shared filesystem has to be in place. We do not know the reason for this, and it has been questioned by others before[14]. Open Science Grid Consortium (OSGC) has developed an additions to GRAM, NFS Lite[15], that alleviates this problem, but only for Condor.

Thus, the head and compute nodes need a shared filesystem. The obvious choice would be for the head node to share its filesystem with the compute nodes. This again could be directed by a "context broker" or from the head node.

3.4 Previous work

Within the Globus Alliance is a project called the Incubator Management Project (IMP), which responsibility it is to help new projects to join the Globus Alliance. Under IMP's umbrella, from 2006 a project started to "study the dynamic creation of execution environments": Virtual Workspaces. The Virtual Workspaces project has developed (and still develops) Nimbus to this end.

Nimbus is Open Source set of tools that deployed on a cluster delivers Cloud IaaS. To launch and manage Virtual Machine (VM) instances, Nimbus offers a Cloud client program. This client program interacts with the Nimbus service using methods also in use with GT: Using X.509 credentials and Web Services Resource Framework (WSRF). But Nimbus also provides an implementation of the Amazon Elastic Compute Cloud (EC2)'s Web Services Description Language (WSDL) based API.

[12] This approach is taken by one-click virtual clusters [17] which will be covered in 3.4.1

[13] This approach is taken by "Elastic Site: Using Clouds to Elastically Extend Site Resource" [21] which we will address in 3.4.2

[14] A message on the GT users mailing-list questioning the necessity for NFS when using GRAM with PBS:
http://lists.globus.org/pipermail/gt-user/2006-November/002165.html

[15] NFS Lite documentation:
http://osg-docdb.opensciencegrid.org/cgi-bin/ShowDocument?docid=382

Given the close relationship with Globus Alliance, it is not surprising that within the Nimbus development efforts, projects emerged closely related to our implementation.

3.4.1 Contextualization: Providing One-Click Virtual Clusters

This interesting paper [17] describes the process of instantiating fixed sized clusters on a IaaS Cloud (Nimbus or EC2 based). It separates the steps involved in instantiating and configuring the VMs in four different roles.

The Appliance Provider supplies VM images, or appliances. Each appliance has a disk image, with a "Context Agent" pre-installed that will be responsible for configuring the instance when the appliance is launched. Each appliance has also a "Context Template" describing what has or may be configured on instantiation.

The Appliance Deployer is the user launching appliances. It selects the appliance (for example a clusters head node) from the Appliance Provider, fills in the "Context Template" to specify how (in what context) the appliance should be configured. The filled in template is then stored, as "Appliance Context" in a ...

Context Broker. The Deployer then instantiates the appliance on a ...

Resource Provider ... with the contact details of the Context Broker. On booting the running instance will contact the Context Broker for details on how to be configured. It will also provide information to the Context Broker about the environment of the running appliance within the Resource Provider (such as its IP-address(es) and dns name), further filling in the Appliance Context. Other instantiated appliances (such as the compute nodes) might be waiting for that information to complete *their* Appliance Context and finalize their configuration.

This separation of roles adds orthogonality to the solution making it very flexible. It has the advantage that credentials needed to instantiate appliances never leave the Deployer's system. Also, credentials to be used by the appliances to mutually authenticate, may be generated by the Deployer, using the Deployer's private key to sign EECs of the appliances and informing them about the others identity via the Context Broker.

The disadvantages are the need for an external Context Broker, and the need for the Cloud client program on the Deployer's system. Nimbus Cloud client software contains the components to instantiate virtual clusters on Nimbus Clouds or Amazon EC2. However, from the documentation[16], we quote:

> *You must also have access to a broker service. You can run your own copy of the broker if you like, but at this time it is recommended to use the service already running on the University of Chicago Nimbus Cloud.*

[16] Advice to use Chicago's broker service:
http://www.nimbusproject.org/docs/2.4/clouds/appendix.html#ec2

Also, the Appliance Provider needs to have the Context Template stored alongside the disk image of an appliance. In existing commercial appliance stores, such as that of Amazon EC2, there is no default way for doing this.

We believe that one of the powers of Cloud computing is the possibility to try out solutions without having to go through the hassle of configuring and setting up a lot of context. Although the solutions concept of a Context Broker is developed with the alleviation of such configuring in mind (especially in collaboration of multiple machines), setting up the environment to use the solution (the client software and manual editing of Extensible Markup Language (XML) Context Template files that manually have to be associated with the appliances) is still a hassle within itself. Still, if this modus operandi would become commonplace, and Amazon would run its own Context Broker for use within its system, this would be the most flexible approach, allowing for the most diverse application environments.

3.4.2 Elastic Site: Using Clouds to Elastically Extend Site Resources

Another interesting paper [21] with two of the same authors as the One-Click Virtual Clusters paper, but it does not build upon the solution from the previous paper. Whereas the cluster in One-Click Virtual Clusters was hosted entirely within the Cloud, the solutions described here assumes an existing physical cluster that elastically extends into possibly multiple Clouds. An Elastic Site Manager is presented as a service that monitors the Cluster Queue and launches or kills new VMs depending on the workload. The Context Broker paradigm introduced in [17] (see 3.4.1) is used to inform the freshly launched compute nodes of the location of the head node. Different scheduling algorithms are evaluated on their effectiveness with different jobs submission patterns typical for specific applications.

The solution extends a personal owned physical cluster resource. Deploying compute nodes in possibly multiple different Clouds kills the homogeneity of the original physical cluster. Users of the original physical cluster could count on certain Quality of Services of the cluster, such as communicational properties between the compute nodes, that can not be guaranteed anymore.

We believe that such scheduling decisions could be better handled by a Grid (meta)scheduler. A Grid (meta)scheduler may take into account specific Quality of Service requirements (or preferences) that must be met (our should be considered) for certain jobs, and further submit the jobs accordingly on a physical or virtual cluster.

Still, the Elastic Site Manager is exactly what we need for our Grid on Demand.

3.5 The Cloud: Amazon Elastic Compute Cloud

Amazon is one of the first commercial IaaS providers. It started selling access to its internally developed[17] VM based resource sharing system in 2006 as the Amazon Elastic Compute Cloud (EC2).

Handling of the VM images, instances, EIPs etc. involved in managing an EC2 Cloud, is offered via a web interface, command-line tools and an API. The remote function calls defined in the API, are offered by Simple Object

[17] Developed to alleviate peaks in demand and more efficiently utilize their available capacity

Access Protocol (SOAP) (for which a WSDL description is available) and HTTP queries. Because of Amazon's popularity, this API has become a De facto standard for managing Clouds, nowadays also offered by Open Source Cloud initiatives such as Nimbus and Eucalyptus.

With EC2 the VM images are addresses via an Amazon Machine Images (AMIs). Amazon has no concept of a possibly external Appliance Provider, like with Nimbus (see 3.4.1). AMIs have to be selected from the repository in the region where the instance will be launched. Regions refer to the location of the physical data centers. Currently, Amazon has four regions: US East (Virginia), US West (Northern California), EU West (Ireland) and Asia Pacific (Singapore).

AMIs might be provided by Amazon, be privately owned or provided by the community. Privately owned AMIs may, by a simple check box selection, made public. Others may find a public AMI as a community provided AMI and launch instances from them.

AMIs have two distinct ways in which the VM image may be stored: by Amazon Simple Storage Service (S3) and by Amazon Elastic Block Storage (EBS). S3 was the original method. When launching S3 backed AMIs, the VM image would be copied from S3 to the disk of the VM instance. When the instance would shutdown, it would just disappear permanently, without the possibility to start the instance again from the same disk. All modifications to the disk image would be permanently lost.

To create new AMIs, one had to download the disk image from S3, modify it and upload it again to S3. The new uploaded disk image then had to be registered to associate it with a new AMI.

Since August 2008[18] a new storage mechanism was introduced: EBS, introducing as new infrastructure for disk images. EBS made it possible to attach persistent storage images (called volumes) to running instances, that would not be destroyed on termination of the instance. Volumes may be copied to snapshots (even from an running instance), which in turn can be used to copy to new volumes for new instances.

Since December 2009[19] it became possible to boot instances from volumes. Furthermore, AMIs may since then be associated with snapshots. This feature created the possibility to administer VMs in a very easy and convenient way.

Figure 9 shows an Remy Evard life cycle diagram [7] of an EBS backed AMI. The cyan colored box in the upper left denote the AMI A and its associated snapshot S. On launch, an instance I_A is created based on the AMI. The instance has a volume associated with it V_S on which the content of snapshot S is copied. The instance I_A is then booted from V_S. It will get a new hostname H assigned.

The instance I_A may now be permanently terminated, loosing its volume. It might also be stopped, in which case the volume V_S and all modification made during the uptime of I_A are preserved. A stopped instance may be started again, booting it from this modified volume. The running instance will receive a new hostname H possibly (probably) different from its previously assigned hostname.

A stopped instance may also be used to create a new AMI A'. The contents

[18] Press release announcing EBS:
`http://phx.corporate-ir.net/phoenix.zhtml?c=176060&p=irol-newsArticle&ID=1189249`

[19] Announcement of booting instances from EBS volumes:
`http://www.allthingsdistributed.com/2009/12/amazon_ec2_boot_from_ebs.html`

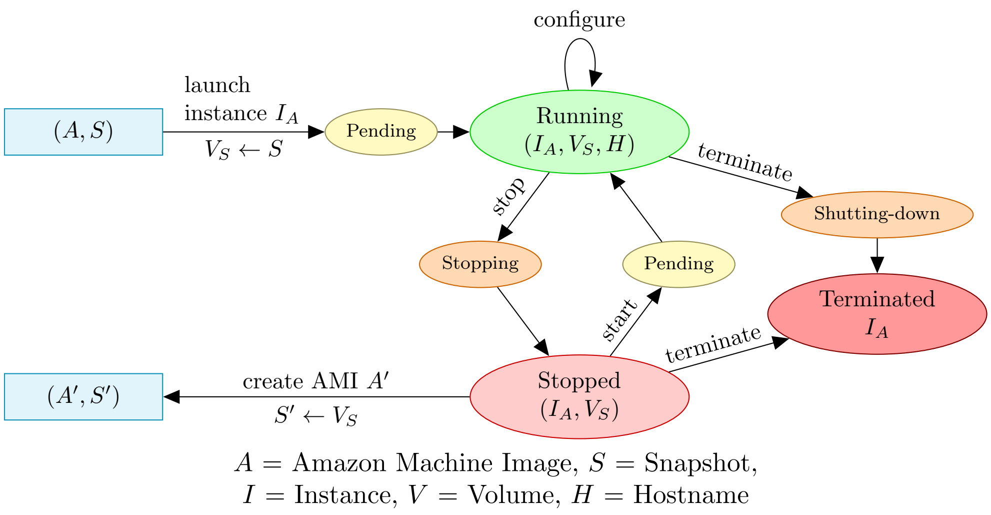


Figure 9: Life cycle of Virtual Machines using EBS

of the modified volume V_S are then copied to a new snapshot S' that will be associated with the new AMI A'. A' may now be made public. It can itself be launched again to further work on the new volume created from the new snapshot, which eventually might lead to a new published AMI again. Also, the old AMI A might itself be reused to create a new development branch (or fork). This modus operandi is similar and has much of the power and flexibility of revision control systems, although merging of different branches is not possible of course.

3.6 Architecture

In our studies of the ingredients needed to build a Cloud based Grid resource, we have seen that providing the instances with the necessary context is the main issue that needs to be resolved.

For Torque, the instances first need to know their role: head node or compute node. The head nodes subsequently need to know the location of the compute nodes, both for deploying jobs and to authorize the mounting of the shared filesystem. The compute nodes need to know the location of the head node, to report on their status and to mount the shared filesystem from. Furthermore, elasticity of the cluster can only be accomplished if an “Elastic Site Manager” has the necessary credentials and is authorized to launch and kill instances in the Cloud (see 3.3.1).

GT needs host EEC signed by a CA that is preferably widely known, but otherwise should at least be known to the users of the Grid resource (see 3.2.2).

In the works of Keahey, Freeman and Marshall [17, 21] a “Context Broker” is presented and used as a supporting service in orchestrating the necessary context to the dynamically instantiated VMs in the correct order and at the right time (see 3.4.1 and 3.4.2). Although the flexibility gained in introducing such an extra service, it also burdens the users of their systems with the need for an environment to be able to use it.

We believe that one of the powers of Cloud computing is the ability to instantiate complete preconfigured systems ready for immediate use or at most needing minimal adaptation to fit in existing environments. Therefor, we have followed a different approach, and have created a Grid enabled Cloud cluster that is run at EC2 alone, completely independent of existing environments. There is no need for a Cloud client and no Context broker is involved.

When a cluster is instantiated in [17], all instances are started at the same time. The Context Broker makes sure the head node learns the locations of the compute nodes, and vice versa. In [21], the head node is not in the Cloud. A Elastic Site Manager monitors the queue on the head node and launches compute nodes in the Cloud. Again, the Context Broker is used to let them know how to contact the head node and to let the head node know the location of the freshly launched compute node. The Elastic Site Manager needs the credentials and authorization to launch nodes in the Cloud.

When launching instances with EC2, several parameters have to be given. The *instance type* (see 4.5), the name of an account associated *public key* to be able to access the machine, an account associated *security group* in which firewall settings for the instance are defined, and an *user data* parameter that can be used to provide scripts to be run just after the instance completed its boot procedure, or in which information can be given on how the instance should be configured. A handle is returned to the launcher, with which information on the just launched instance can be retrieved. Such as its status (pending, running, stopping, etc.) and the IP address and hostname it eventually received through Dynamic Host Configuration Protocol (DHCP).

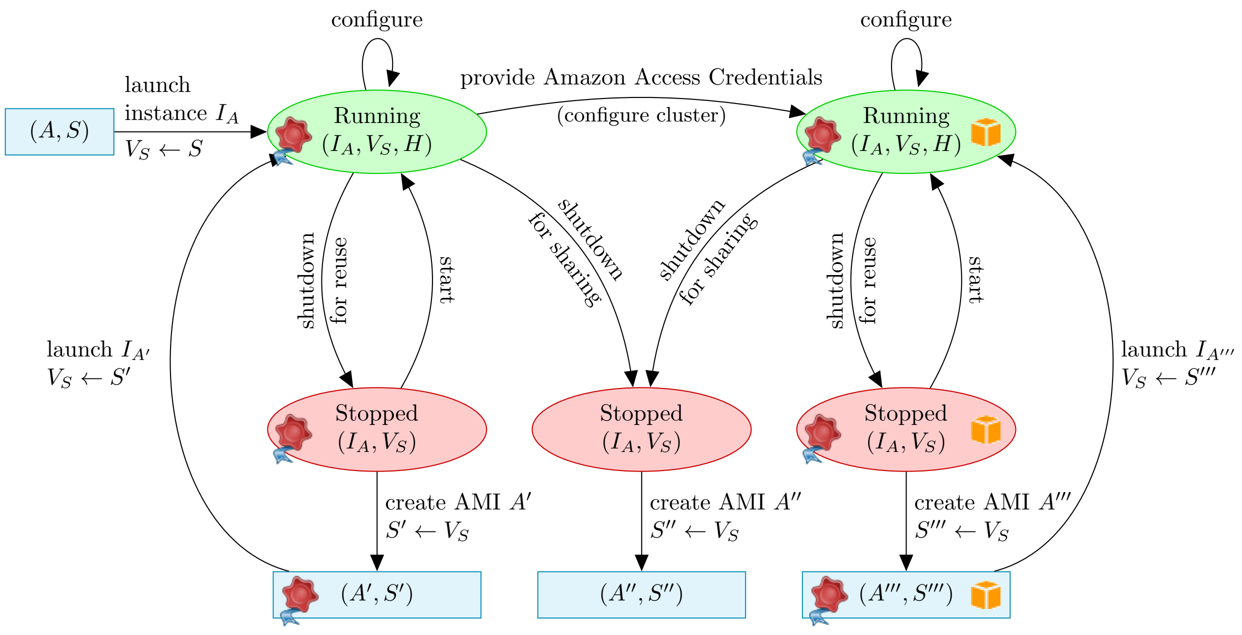
In the scenario of [21] this means that in stead of using a Context Broker, the location of the head node can be passed to the launched compute nodes via the *user data* parameter. The compute node may subsequently contact the head node directly (in stead of via a Context Broker) to notify it is ready. The same interaction can be used to authorize and mount the for GRAM needed shared filesystem (see 3.3.1).

The obvious location for *our* Elastic Site Manager is on the head node, but it needs Amazon's Access Credentials to be able to launch compute nodes. We can not store the credentials in the public AMI. An account associated store is needed to be able to store the credentials, so they do not have to be in the AMI. EBS provides just that (see 3.5). It also enables a way to conveniently store an account associated CA. The recent ability to boot from an EBS volume makes it even more convenient as can be seen in figure 10.

3.6.1 Incorporating Amazon's administrative features in Grid on Demand

The AMI in the upper left corner of figure 10 is our public Grid on Demand AMI. It is based on an existing AMI of Canonical with Ubuntu Lucid Linux installed, and extended it with GT, Torque, our own Elastic Site Manager and all the necessities to create the Grid on Demand as outlined in 3.1.

The with this public AMI associated snapshot S has no sensitive information, such as a private key for the CA and the Amazon Access Credentials, on it. On an initial first time usage launch of the AMI, a new for the holder of the Amazon account intended CA is created. Its private key is used to sign the EEC needed for the resource's hostname H to be able to participate in the Grid. As a bonus, also an account on the AMI intended to be used as a Grid user is provided with



A = Amazon Machine Image, S = Snapshot,
I = Instance, V = Volume, H = Hostname

Figure 10: Different configured stages in the life cycle of Grid on Demand

a EEC and a PC is initialized, so the Grid user account can be used immediately to use the resource. All those configurational actions are stored on the instance's volume V_A.

Secure SHell (`ssh`) is Amazon's default method to connect to Linux instances. The private key on the on launch specified keypair has to be used to authenticate. Configuration of Grid on Demand is through a web interface. For security, it only listens on the local interface. To use the interface, a local port has to be forwarded to the instance's local interface (port 80). Port forwarding is a feature found in many `ssh` client (among which OpenSSH[20] and PuTTY[21]). This way, the standard Amazon way of protecting access to the instance is also used to protect the usage of the web interface.

The web-interface can be used to authorize Grid users with an EEC signed by a real Grid CA. Grid on Demand has the by the IGTF distributed list of accredited Grid CAs on board and updates them regularly. To be able to use the Grid on Demand as a Grid resource, one has to make sure that all systems accessing the Grid on Demand acknowledge its CA. To this end the CA's certificate needs to be downloaded from the web interface, and added to the list of certified CAs of those systems.

The web interface also provides two buttons to shutdown the instance: One to shutdown for reuse, and one to shutdown for sharing. When shutdown for

[20] OpenSSH is a FREE version of the SSH connectivity tools:
https://www.openssh.org/
[21] PuTTY is a free implementation of SSH for Windows and Unix:
http://www.chiark.greenend.org.uk/~sgtatham/putty/

reuse, the instance is simply shutdown. The CA, the EEC of the local Grid user and the list of authorized externally signed Grid users are preserved on the volume V_S. When the stopped instance is now started again, all those settings keep preserved. No new CA or EEC of a local Grid user is created. The list of authorized Grid users is the same as on shutdown. Only a new host EEC will be generated to facilitate the new hostname the instance will get.

The stopped instance I_A can also be use to create a new ***private*** AMI. This preconfigured AMI may then be used to launch multiple Grid's for use with the in the AMI A''s volume S' contained list of authorized users, or from the systems that already know the CA that is stored in S'.

Alternatively, the instance I_A might also have been shutdown for sharing. In that case, the CA, Grid users EEC, list of authorized users and all other "personal" Grid on Demand configurations are removed before the VM is halted. The AMI A'' created from the volume of that stopped instance may safely be offered to the public again. This might be useful to branch of versions of Grid on Demand targeted at specific applications, or on which certain services from the collective layer are deployed (see 3.2)

With the web interface also the Amazon Access Credentials can be provided, after which Grid on Demand may operate as an Elastic cluster. Similarly as before, the Access Credentials are preserved when shut down for reuse, and removed when shut down for sharing.

3.6.2 Components

We have implemented an Elastic Site Manager (ESM), similar as in [21]. The ESM is implemented in python. It monitors the Torque job queue with the help of the pbs_python[22] package.

We have defined a simple retention formula to prevent over provisioning of compute nodes. To determine if new compute nodes should be started, it maintains a history of the number of pending jobs W_t (the startup window), over the last ℓ_w number of seconds at moment t. The startup window length ℓ_w can be configured in the web interface. The number of compute nodes that will be launched L_t at time t is determined by the number of cores of the instance type that will be launched C_T and the current number of cores of compute nodes that have already been started but have not joined the cluster yet (Cores Pending) C_t^P, the current number of running nodes R_t and the minimum and maximum cluster size that can be configured with the web interface $S_{\min}$ and $S_{\max}$, and is given equation 1.

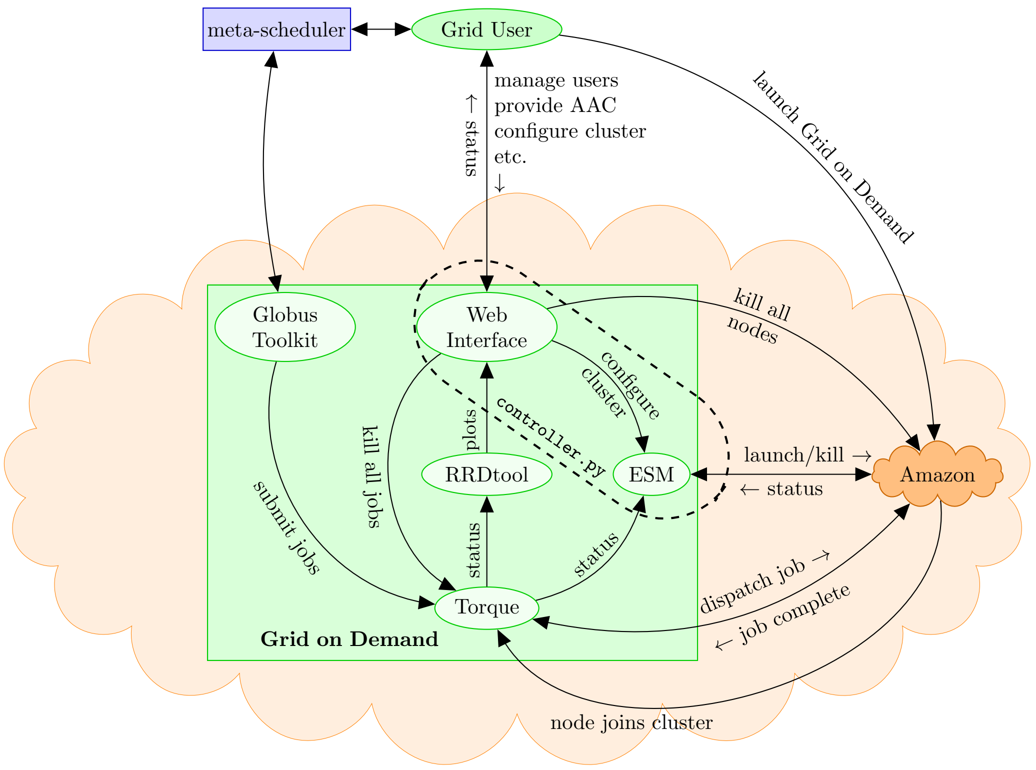
$$L_t = \min\left\{\max\left\{\max\left\{\left\lceil\frac{\min W_t - C_t^P}{C_T}\right\rceil, 0\right\} + R, S_{\min}\right\}, S_{\max}\right\} - R \qquad (1)$$

where min and max are functions that give the smallest and largest number from the set that is the argument respectively.

Compute nodes are terminated when they are idle, and their uptime t_{up} comes close to a whole hour. If t_{up} is in seconds, then $t_{\text{up}} \bmod 3600 > 57 \times 60$.

Note that L_t is negative when there are more nodes running then maximally allowed ($R > S_{\max}$). This can happen if $S_{\max}$ is set to a lower number through

[22] pbs_python: https://subtrac.sara.nl/oss/pbs_python



ESM = Elastic Site Manager
AAC = Amazon Access Credentials

Figure 11: Components in Grid on Demand

the web interface. Grid on Demand currently does not anticipate this. In future version it might remove $-L_t$ from the list of compute nodes managed by Torque giving a preference to idle compute nodes. Those compute nodes could terminate normally when $t_{\text{up}} \bmod 3600 > 57 \times 60$, or might be added to the cluster again when $L_t > 0$.

All managing (launching, killing and retrieving status information) of compute nodes is done with boto[23].

As is shown in figure 11, both the web interface and the ESM are provided by the same python script: `controller.py`. The web interface component, in the script, further provides graphical plots that display the progress of running and queued jobs and running and pending compute nodes (expressed in number of cores). The plots are generated by Round Robin Database Tool (RRDtool)[24] through PyRRD[25].

[23] boto: A Python interface to Amazon Web Services:
http://boto.s3.amazonaws.com/index.html
[24] RRDtool is the OpenSource industry standard, high performance data logging and graphing system for time series data:
http://oss.oetiker.ch/rrdtool/index.en.html
[25] PyRRD is a pure-Python OO wrapper for the RRDtool:
http://code.google.com/p/pyrrd/

4 Performance

To test the performance of Grid on Demand a comparison needs to be made with a real cluster having a Grid interface (section 4.1). An actual e-science workload will generate a representative load on both the real cluster and Grid on demand. When this test is performed using an existing Grid application utilizing the Grid interface (Globus Toolkit) it also shows that Grid on Demand can be used in existing environments without modification. For the e-science workload the WAVE workload has been used described in section 4.2. The WAVE workload uses WS-VLAM as a workflow manager, section 4.3 describes the concept a workflow manager and WS-VLAM in particular.

4.1 DAS-3

The Distributed ASCI Supercomputer 3 (DAS-3)[26] is a five-cluster wide-area distributed system designed by Advanced School for Computing and Imaging (ASCI). It is meant for doing experimental research on parallel and distributed programming. For the tests the 32 node DAS-3 cluster of the University of Amsterdam (UvA) is used. The UvA part of the Distributed ASCI Supercomputer 3 (DAS-3-UvA) has GT 4.02 installed and is running Scientific Linux 4.6. On DAS-3-UvA only one CPU per node is used when jobs are submitted due to the setup of the scheduler .

4.2 The WAVE workload

The WAVE application is a biomedical application. It can help surgeons to create a arteriovenous (AV) fistula in the blood vessels of the upper limb for hemodialysis. Using an ultrasound scan of the patients arm it tries to calculates, by means of simulations, the correct place to create the AV fistula. Intensive calculations are done by a standalone application that fetches the data and generates the output. There is no need for communication between the calculations. By variating the input data per calculation an extensive simulation can be done. This type of application can be typed as an ”embarrassingly-parallel parameter space study” [16]. The workflow of WAVE the application has been developed in WS-VLAM (section 4.3)

4.3 WS-VLAM workflow manager

A workflow manager for grid computing [28] is a (graphical) tool to assist complex e-science application creation. It abstracts the complexity of the middleware and frameworks that provide the capabilities for distributed execution in the Grid environment. The UvA developed WS-VLAM[18] workflow management system targets to support efficient and scalable execution of large workflow applications on the Grid. WS-VLAM creates a sequence of job submissions to GT taking into account the dependencies between the jobs for example.

[26] http://www.cs.vu.nl/das3/index.shtml

4.3.1 Adjustment to WS-VLAM

A first test run showed that the workflow manager was not optimized for the elasticity of Grid on Demand. WS-VLAM submitted jobs until the job queue was full and it did not notice the automatic expansion of the queue size. This was corrected by the developers of WS-VLAM.

4.4 Compiled and installed versions

Table 3 shows an overview of the versions installed in the Ubuntu 10.04 LTS 64bit AMI to create the performance test enviroment.

Software	Compiled	Version
WS-VLAM runtime	From source	wsruntime-31.03.2010
WAVE	Statically linked binary	Provided by System- and Network-Engineering (SNE) group
GT	From source	4.2.1
Torque	From source	2.4.8

Table 3: Installed versions of the performance test software components

4.5 Amazon instance types

Amazon offers different types of instances (virtual machines) to run a AMI. These instances have different characteristics which can influence the performance of the AMI. Table 4 gives an overview of the available 32bit and 64bit instance types when the tests where performed. The term Compute Unit (CU) is defined by Amazon as follows: "One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor".

API name	**32/64bit**	**Memory(MB)**	**Cores**	**CU**
`m1.small`	32bit	1.7	1	1
`m1.large`	64bit	7.5	2	4
`m1.xlarge`	64bit	15	4	8
`m2.xlarge`	64bit	17.1	2	6.5
`m2.2xlarge`	64bit	34.2	4	13
`m2.4xlarge`	64bit	68.4	8	26
`c1.large`	32bit	1.7	2	5
[`c1.xlarge`]	64bit	7	8	20

Table 4: Available Amazon instance types

4.6 Performance tests

To test the elasticity/scalability of Grid on Demand as opposed to the static setup of the DAS-3-UvA cluster two workloads were tested. One with a number of jobs less than the number of nodes in the cluster and one with more jobs, in this case 3 times more. than the number of nodes. These values were chosen to test the advantages of the dynamic scalability of Grid on Demand. For DAS-3-UvA the hypothesis is that when a workload has more Jobs than that there are nodes available to execute them, the jobs will be placed in the queue (pending). These pending jobs will be executed when running jobs are finished. For Grid on Demand the number of CPU's available is always equal (or more) than the number of jobs due to the dynamic scaling. Therefore the total execution time of the workload will equal the total execution (running + pending) time of the slowest job.

$$\begin{aligned}
C_a &= \text{Cores available}\\
W_t &= \text{Workload execution time}\\
W_j &= \text{Number of Jobs in Workload}\\
J_t &= \text{Job total execution time}\\
J_{t-max} &= \text{Maximum Job total execution time}\\
J_p &= \text{Job pending time}\\
J_r &= \text{Job running time}\\
J_t &= J_p + J_r
\end{aligned}$$

The predicted total workload execution time on DAS-3:

$$W_t \equiv J_t \times (W_j \mod C_a) \tag{2}$$

For Grid on Demand :

$$W_t \equiv J_{t-max} \tag{3}$$

4.7 Performance results

A schematic setup of the performed test is shown in figure 12.

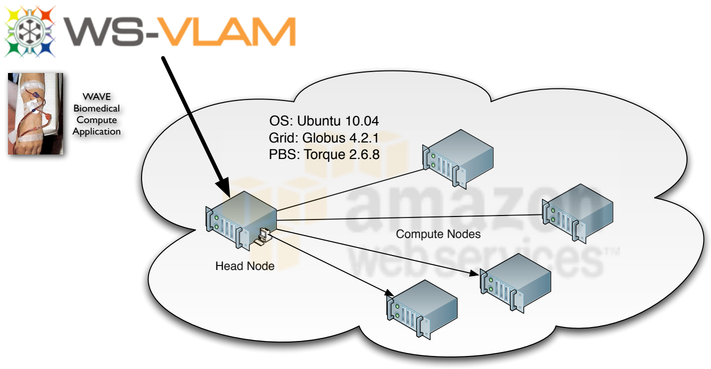


Figure 12: Setup of GoD, WS-VLAM and a real-life application

As described in section 4.2 each job run by the WAVE workload uses a different dataset. To avoid the influence of the different data on the job running time

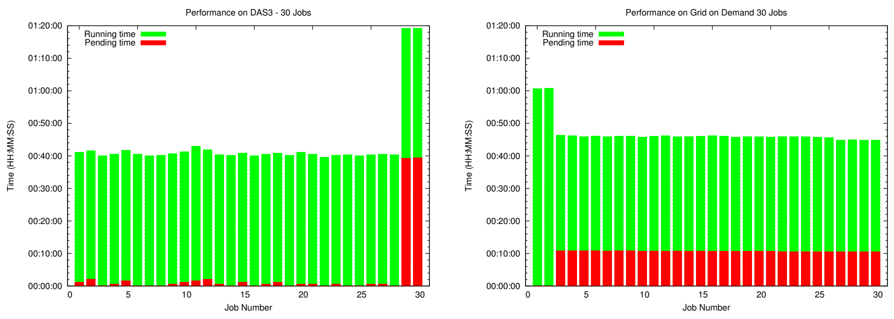


Figure 13: Results of a 30 job workload

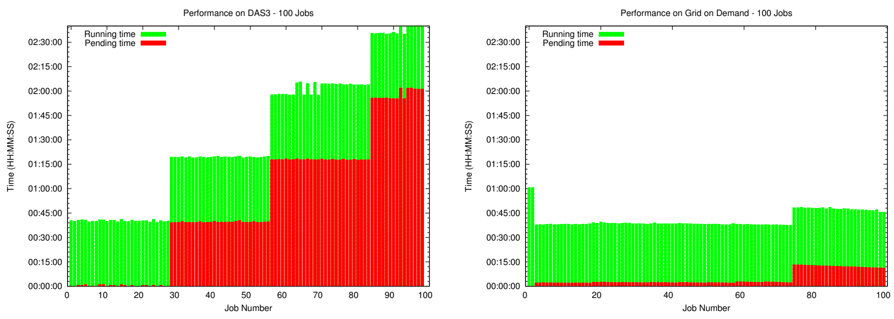


Figure 14: Results of a 100 job workload

the same dataset is used for all the jobs in all the performed tests. The head-node is running on the `m1.large` (see section 4.5) instance. The dynamically started compute nodes are `c1.xlarge` instances (see section 4.5).

Total execution time The time it takes for the complete workload to finish as seen by WS-VLAM

Job pending time The time a job is pending before it is run as seen by WS-VLAM

Job running time The time a job is running until it finishes as seen by WS-VLAM

The graphs in figure 13 and figure 14 confirm the predicted results. The first 2 jobs for Grid on Demand in both the 30 en 100 job run show a very different job running time compared to the other job running times in the 30 and 100 job. This is due to the participation of the head-node in the job queue. The head-node on the m1.large instance has different characteristic than de dynamically created compute nodes running on the c1.xlarge image this is clearly visible in the running time results and are therefore removed from the following test results in table 5 and table 6. In the 30 job performance test of DAS-3-UvA the last 2 jobs show a very different job pending time compared to the other job pending times. This is due to the (unexpected) availability of only 28 nodes in theDAS-3-UvAcluster, the 2 jobs are waiting for 2 other jobs to finish. These 2 jobs are removed from test results in table 5 and table 6. To create an equal comparison between both the 100 job runs for DAS-3-UvA and Grid on Demand the last 2 jobs on DAS-3-UvA where also removed.

Grid	Number of Jobs	Total execution time
DAS-3-UvA	28	00:42:58
Grid on Demand	28	00:46:22
DAS-3-UvA	98	02:46:36
Grid on Demand	98	00:52:26

Table 5: Total execution time of Grid on Demand compared to DAS-3-UvA

The content of table 5 is as predicted. The workload execution time on the DAS-3-UvA cluster is almost equal to the job execution time when the number of jobs in the workload is less than the number of CPU's available. The workload execution time in DAS-3-UvA is 4 times longer then the job execution time when the number of jobs is more than 3 times the number of available CPU's. Grid on Demand scales with the number of jobs, and the total workload execution time of a 98 job workload is close to the 28 jobs workload. The difference between the 100 and the 30 job workload is still significant and as can bee seen in figure 14 is due to the sudden increase of the pending time around job number 70. Why this is, needs further investigation.

Grid	Jobs	Average	Min	Max	σ
DAS-3-UvA	28	00:40:01	00:39:26	00:41:19	00:00:21
Grid on Demand	28	00:34:21	00:35:34	00:35:34	00:00:20
DAS-3-UvA	98	00:41:09	00:39:23	00:47:47	00:02:34
Grid on Demand	98	00:40:03	00:39:27	00:41:19	00:00:21

Table 6: Job running time of Grid on Demand compared to DAS-3-UvA

In both the used grid and cloud every resource has the same characteristic. One of the issues when using shared resources can be that others uses of the shared resource can influence the performance of the resource. The values in table 6 show the average running time of of the jobs per workload and the standard deviation σ of the jobs in that workload. Looking and the minimum and maximum values, it shows that Grid on Demand performance (in this case) has decreased when the 100 job workload is run. In the 100 jobs workload run the values of DAS-3-UvA have a big standard deviation σ in DAS-3-UvA but not in Grid on Demand, this would suggest (in this case) that the cloud offered a more consistent resource. For Grid on Demand further investigation is needed to test these variations. In a more general study of these variations in the Amazon EC2 environment has been conducted by S. Ostermann et al. [22].

4.8 Performance tests cost

Using a cloud provider for the resources of the performance test has the advantage that the cost of this these performance tests can be calculated (section 2.5). The price (July 2010) for the use of one hour for the head-node, running on the `m1.large` in the EU-West-1 region is \$0.38. The compute nodes running on the `c1.xlarge` instances \$0.76 per hour. For both the 30 job and 100 job test the head node was running 2 hours. For the 30 job test 4 compute nodes (8 CPU) ran for 1 hour, for the 100 job, 13 compute nodes (8 CPU) ran 1 hour. The total cost for the all instances used during the performance tests: $4 * \$0.38 + 17 * \$0.76 = \$14.44(\text{excl VAT})$

5 Conclusion

This research project is done as part of our Master course in System- and Network-Engineering at the University of Amsterdam. Prior to the completion of this report we gave a presentation of our work. When presenting figure 12 on a slide leading up to our description of the implementation, we received an objection that it does not illustrate a Grid. And in fact, it is does not.

Grid computing is an impressive endeavor in collaboration with the aim of increasing available compute and storage capacity to all participants by resource sharing. Figure 12 shows just a cluster. In this research we have explored the potential of using such a Cloud based cluster as a Grid resource and as such make it part of the Grid. We have explored augmenting the Grid with a Grid resource on Cloud fabric; “a Cloud in the Grid”, and not “a Grid in the Cloud”.

In the following sections the research question defined in the introduction will be answered:

5.1 Can Grid computing be offered as a Cloud service?

Cloud resources can be leveraged to augment the Grid, but under certain conditions.

The conditions are twofold and in analogy with the classic Grid papers [11, 9] we categorize them as anatomical and physiological.

5.1.1 Anatomical

By equipping a virtual cluster with the Globus Toolkit (GT) we were able to use it with an existing Grid application. But the Grid application was under our direct tight control. When the Cloud resource would be used indirectly, for example using a third party metascheduler that would allocate resources in the Cloud as a last resort, authentication problems would emerge because of the unestablished Certificate Authority endorsing the identity of the Cloud resource.

Grid Security Infrastructure (GSI) has the crucial role of enabling Grid collaborations and the compliance with the agreements in them. GSI is designed to enable the allocation of resources. Dynamically instantiated resources were not anticipated during its design, but we believe GSI is flexible enough to address the matter. Based on an usage example of [25] we have illustrated in 3.2.3 that by reorganizing and anticipating the allocation of resources, the current architecture of our implementation would be sufficient to be able to collaborate in bigger Grid collaborations.

As a simpler and more realistically achievable alternative we suggest trusting a resource when it is endorsed (signed) by the same “User End Entity Certificate”, as the Proxy Certificate used in allocating the resource. A user, represented by delegation (or directly), can trust itself. However, this implies that the dynamically instantiated resource would need to communicate with the Grid user to receive its delegation. The independent nature of our solutions is not optimal for such application. The integration of the Nimbus Cloud Client with GT, would make it a more suitable candidate.

5.1.2 Physiological

Cloud computing has many attractive properties that Grids do not have [1]. It has the illusion of infinite availability of compute capacity on demand, where Grid collaborations give access to static perceptible fixed sized resources. It does not involve upfront commitments, where Grid collaborations are perpetuated in agreements. Resources are paid only when used, where for Grid participation one has to contribute a resource.

By augmenting Grid with a Cloud resource, the Grid inherits those benefits. Leveraging a Cloud based Grid resource, existing Grid applications will enjoy the benefits of Cloud computing without loosing the architectural and organizational (anatomic and physiologic) advantages of the Grid.

This is ideal for Urgent Computing, where the demand for capacity may peak on occasions, because it effectively eliminates the need to provision ahead to anticipate those incidents. Also experimental use cases, for which using the bigger Grid is not appropriate or unattainable, can benefit greatly from Grid on Demand. It is available directly, without needing permissions or having to behave.

Still, a subscription with a Cloud service involves an agreement with a Cloud provider that puts limitations on its usage. An Amazon Elastic Compute Cloud agreement has a default limitation of twenty running instances per region (of which there are currently four). Of course, this is merely an administrative matter because more then one subscription could be leveraged to add computing power.

Before we started our research the default limitation of twenty running instances per region was extended to seventy running instances per region. As an ultimate proof of concept we wanted to run a thousand instances. According to [23] one Elastic Compute Unit (ECU) equals 4.4 billion Floating Point Operations Per second (GFLOPs). The "High-CPU Extra Large Instance" (`c1.xlarge`) type has 20 ECUs. When launching a thousand `c1.xlarge` instance types we will employ a resource with the theoretical capacity of 35.2 TFLOPS, costing $760.00 per hour. As of June 2010 this would have put the resource at position 219 in the top 500 supercomputing list[27]; a formidable result. However one has to take into account that the physical hardware underlying the instances may bear more then one instance.

During our research we were unable to negotiate, an extension to instantiate a thousand running instances in the European region. Especially running the `c1.xlarge` was an issue, one costumer using eight thousand CPU's would put a big strain on the European region where "currently there is a lot of demand for `c1.xlarge` instances" as a business development manager for Amazon EC2 stated.

[27] The top-500 Supercomputers: http://www.top500.org/list/2010/06/300

6 Acknowledgments

This report was created as part of a four week research project done as part of the SNE master[28] at the UvA. The research was done at the UvA SNE group location at Science Park Amsterdam. We would like to thank the SNE master education for providing resources and time for this research project. A special thanks to Paola Grosso and Rudolf Strijker for their guidance and explanations that gave us the insights needed to do this project. A very special thanks to Adam Belloum and Dmitry Vasyunin who helped us understand the Grid and were very helpful to connect WS-VLAM to our Grid on Demand.

[28] http://www.os3.nl

A Appendix: Grid on Demand User guide

This appendix demonstrates the usage of Grid on Demand by way of screenshots.

A.1 Launching Grid on Demand

Before launching Grid on Demand, it is important to have the correct firewall configuration for the instance. Figure 15 shows the configuration of the Security Group that should be chosen on launch. Port 2119 for GRAM's gatekeeper service, 2881 for GridFTP, 8443 for GT web services and port 22 to be able to access the instance and Grid on Demand's web based user interface.

After clicking the "Launch Instance" button from the Amazon Management Console, we choose the latest version of Grid on Demand from the community AMIs repository (figure 16 and 17).

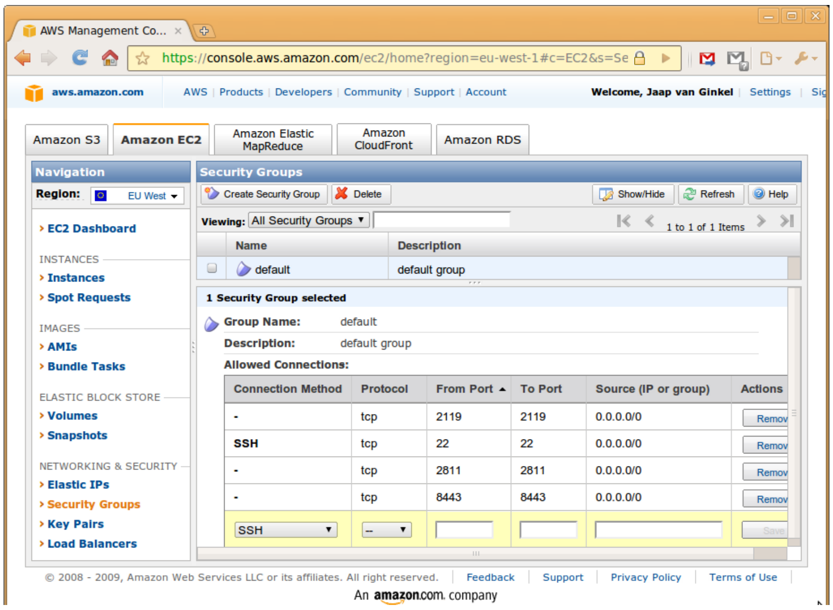
The status of the just created instance can be monitored in the "My Instances" section of the console. When connecting to the running instance in the default Amazon way (figure 18, 19 and 20), we are told to either connect as "`ubutu`" for maintenance (for example to equip the volume with application specific components or to enable Grid services on the Collective layer (see 3.2)), or as "`griduser`". The instructions shown also settle the local port forwarding so that the web interface of Grid on Demand becomes accessible from the local computer.

In figure 21 we login as "`griduser`" and the web interface of Grid on Demand is available via `http://localhost:8080/` as shown in figure 22.

Besides the functionality to further configure Grid on Demand, the web interface allows for monitoring of the Elastic Cluster. It shows the number of running and queued jobs, running and pending compute nodes and the total money spent using the Grid on Demand *cluster*.

The Grid on Demand head node is also operating as a compute node. Jobs are deployed among cores. Because we have launched an instance type with two cores (`m1.large`) we are able to simultaneously run two jobs. The RRDtool plots reflect the available slots for running jobs and will show the progress of the status of the jobs over time. The plots from top till bottom show increasing larger time spans. The top plot has a one minute span, below that a 10 minute span, then four hours, twelve hours, 24 hours, one week and four weeks.

Because we have logged in as "`griduser`" we can immediately use Grid on Demand. A PC is already initialized. We may now use GridFTP, the GT web services, or submit a job via GRAM's gatekeeper service as show in figure 23.



Name	Description
default	default group

1 Security Group selected

Group Name: default

Description: default group

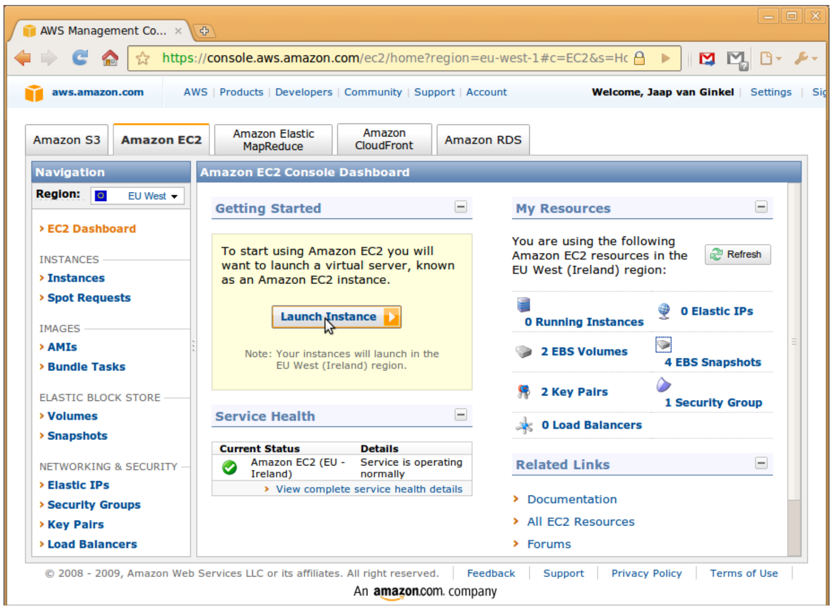
Allowed Connections:

Connection Method	Protocol	From Port	To Port	Source (IP or group)	Actions
-	tcp	2119	2119	0.0.0.0/0	Remov
SSH	tcp	22	22	0.0.0.0/0	Remov
-	tcp	2811	2811	0.0.0.0/0	Remov
-	tcp	8443	8443	0.0.0.0/0	Remov
SSH	--				Save

© 2008 - 2009, Amazon Web Services LLC or its affiliates. All right reserved. | Feedback | Support | Privacy Policy | Terms of Use

An amazon.com. company

Figure 15: Firewall Settings



To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

Note: Your instances will launch in the EU West (Ireland) region.

You are using the following Amazon EC2 resources in the EU West (Ireland) region:

0 Running Instances	0 Elastic IPs
2 EBS Volumes	4 EBS Snapshots
2 Key Pairs	1 Security Group
0 Load Balancers	

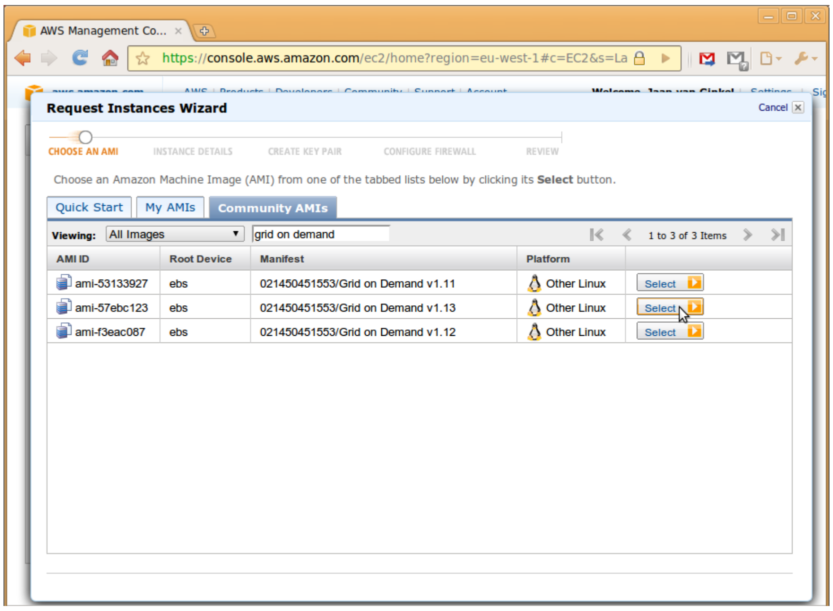
Current Status	Details
✓ Amazon EC2 (EU - Ireland)	Service is operating normally

› View complete service health details

© 2008 - 2009, Amazon Web Services LLC or its affiliates. All right reserved. | Feedback | Support | Privacy Policy | Terms of Use

An amazon.com. company

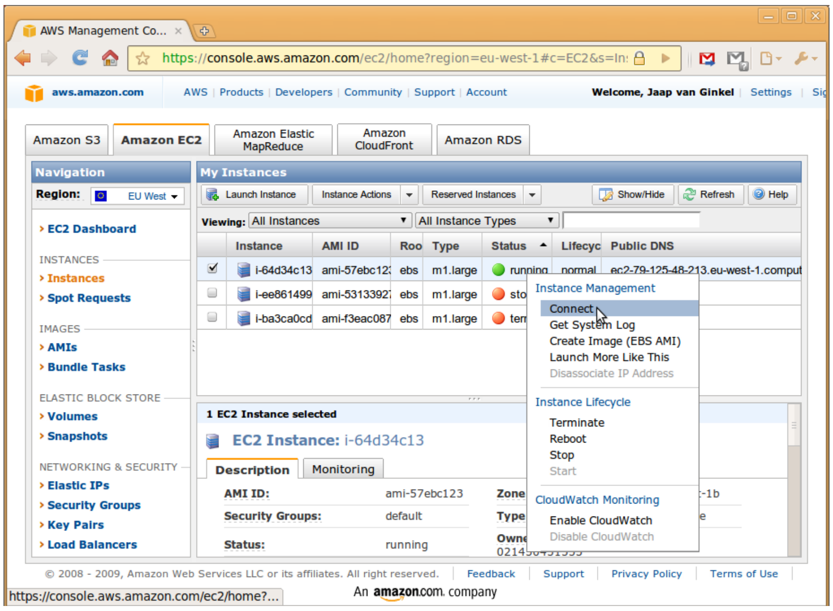
Figure 16: About to launch Grid on Demand



Choose an Amazon Machine Image (AMI) from one of the tabbed lists below by clicking its **Select** button.

AMI ID	Root Device	Manifest	Platform	
ami-53133927	ebs	021450451553/Grid on Demand v1.11	Other Linux	Select
ami-57ebc123	ebs	021450451553/Grid on Demand v1.13	Other Linux	Select
ami-f3eac087	ebs	021450451553/Grid on Demand v1.12	Other Linux	Select

Figure 17: Selecting the AMI



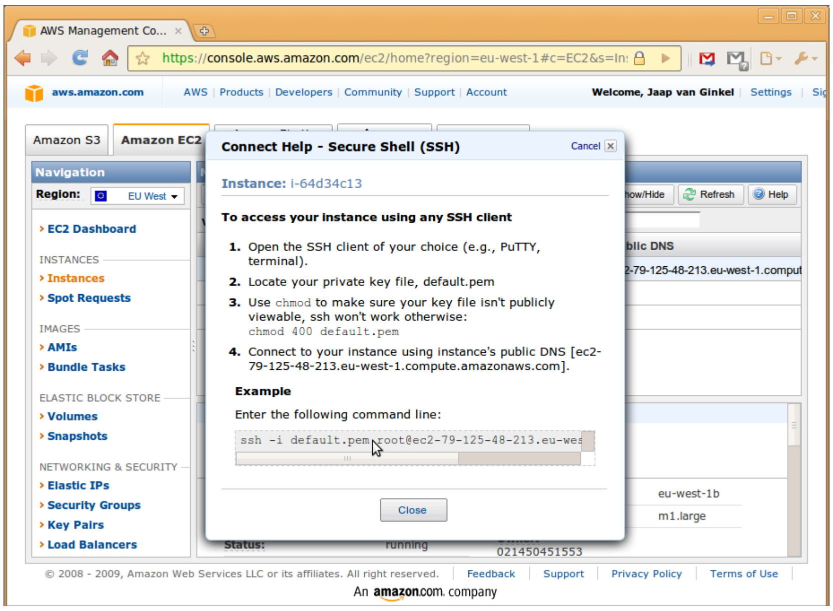
	Instance	AMI ID	Roo	Type	Status	Lifecyc	Public DNS
✓	i-64d34c13	ami-57ebc12	ebs	m1.large	● running	normal	ec2-79-125-48-213.eu-west-1.comput
	i-ee861499	ami-5313392	ebs	m1.large	● sto		
	i-ba3ca0cd	ami-f3eac087	ebs	m1.large	● ter		

1 EC2 Instance selected

EC2 Instance: i-64d34c13

AMI ID:	ami-57ebc123	Zone	-1b
Security Groups:	default	Type	e
Status:	running	Owne	

Figure 18: Connecting with the “head node”



Connect Help - Secure Shell (SSH)

Instance: i-64d34c13

To access your instance using any SSH client

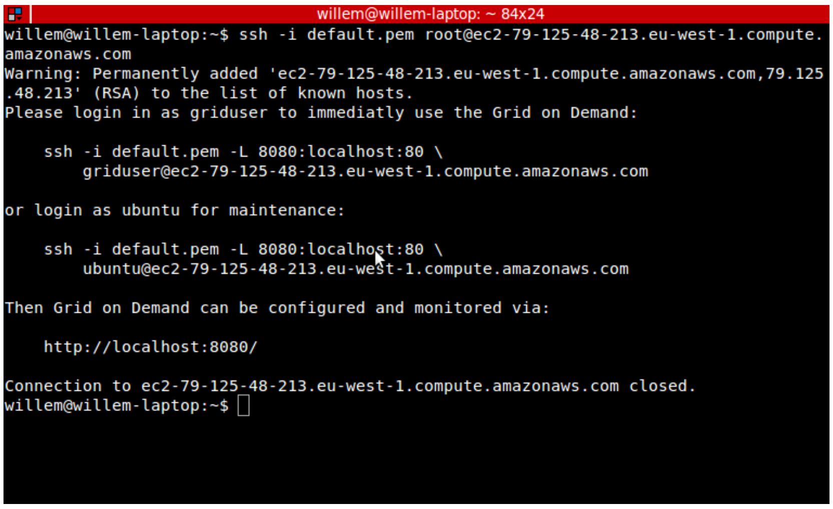
1. Open the SSH client of your choice (e.g., PuTTY, terminal).
2. Locate your private key file, default.pem
3. Use `chmod` to make sure your key file isn't publicly viewable, ssh won't work otherwise:
   `chmod 400 default.pem`
4. Connect to your instance using instance's public DNS [ec2-79-125-48-213.eu-west-1.compute.amazonaws.com].

Example

Enter the following command line:

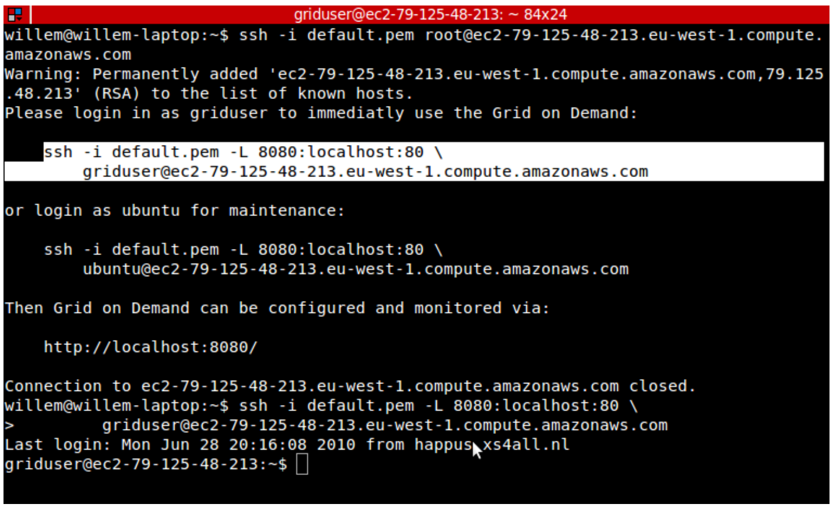
```
ssh -i default.pem root@ec2-79-125-48-213.eu-wes
```

Figure 19: Copy the suggested `ssh` command ...



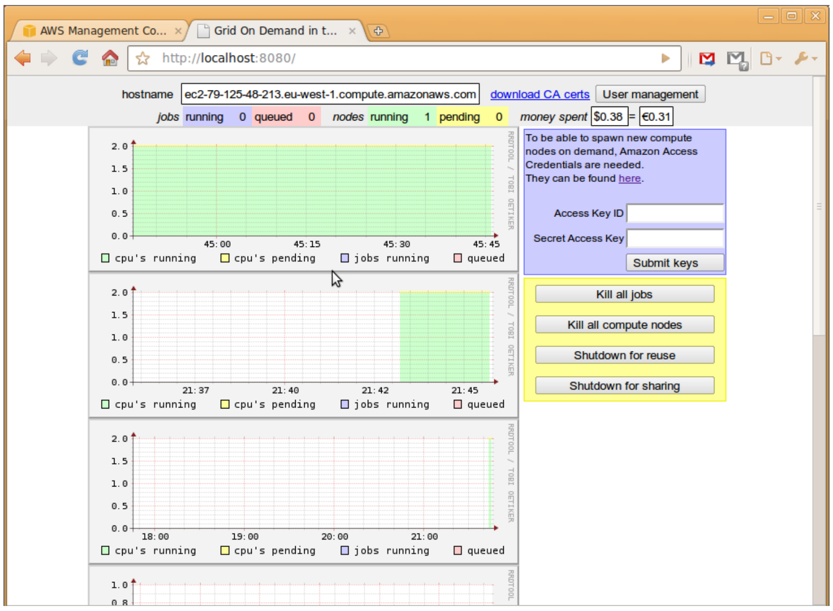
```
willem@willem-laptop:~$ ssh -i default.pem root@ec2-79-125-48-213.eu-west-1.compute.
amazonaws.com
Warning: Permanently added 'ec2-79-125-48-213.eu-west-1.compute.amazonaws.com,79.125
.48.213' (RSA) to the list of known hosts.
Please login in as griduser to immediatly use the Grid on Demand:

    ssh -i default.pem -L 8080:localhost:80 \
        griduser@ec2-79-125-48-213.eu-west-1.compute.amazonaws.com

or login as ubuntu for maintenance:

    ssh -i default.pem -L 8080:localhost:80 \
        ubuntu@ec2-79-125-48-213.eu-west-1.compute.amazonaws.com

Then Grid on Demand can be configured and monitored via:

    http://localhost:8080/

Connection to ec2-79-125-48-213.eu-west-1.compute.amazonaws.com closed.
willem@willem-laptop:~$ 
```

Figure 20: ... and paste



```
griduser@ec2-79-125-48-213: ~ 84x24
willem@willem-laptop:~$ ssh -i default.pem root@ec2-79-125-48-213.eu-west-1.compute.
amazonaws.com
Warning: Permanently added 'ec2-79-125-48-213.eu-west-1.compute.amazonaws.com,79.125
.48.213' (RSA) to the list of known hosts.
Please login in as griduser to immediatly use the Grid on Demand:

    ssh -i default.pem -L 8080:localhost:80 \
        griduser@ec2-79-125-48-213.eu-west-1.compute.amazonaws.com

or login as ubuntu for maintenance:

    ssh -i default.pem -L 8080:localhost:80 \
        ubuntu@ec2-79-125-48-213.eu-west-1.compute.amazonaws.com

Then Grid on Demand can be configured and monitored via:

    http://localhost:8080/

Connection to ec2-79-125-48-213.eu-west-1.compute.amazonaws.com closed.
willem@willem-laptop:~$ ssh -i default.pem -L 8080:localhost:80 \
>         griduser@ec2-79-125-48-213.eu-west-1.compute.amazonaws.com
Last login: Mon Jun 28 20:16:08 2010 from happus.xs4all.nl
griduser@ec2-79-125-48-213:~$ 
```

Figure 21: Logged in as `griduser` with local port forwarding



hostname ec2-79-125-48-213.eu-west-1.compute.amazonaws.com

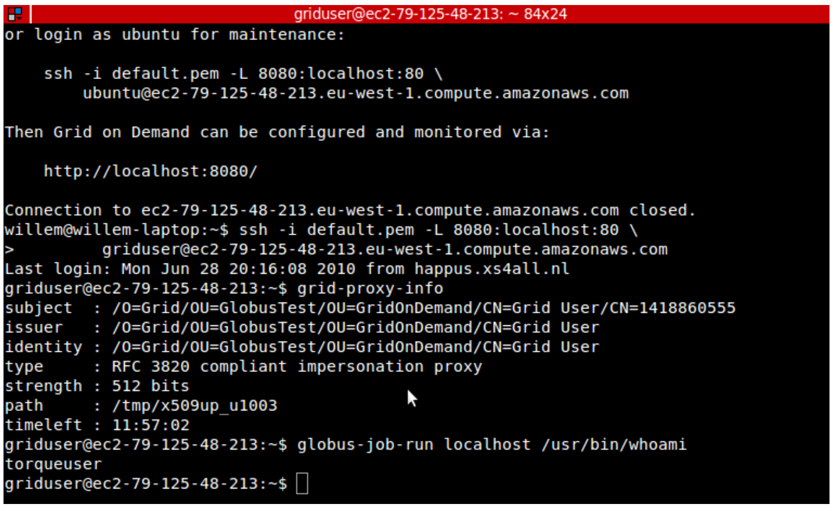
jobs running 0 queued 0 *nodes* running 1 pending 0 *money spent* $0.38 = €0.31

To be able to spawn new compute nodes on demand, Amazon Access Credentials are needed. They can be found here.

Access Key ID

Secret Access Key

Figure 22: The web interface is now available



```
or login as ubuntu for maintenance:

    ssh -i default.pem -L 8080:localhost:80 \
        ubuntu@ec2-79-125-48-213.eu-west-1.compute.amazonaws.com

Then Grid on Demand can be configured and monitored via:

    http://localhost:8080/

Connection to ec2-79-125-48-213.eu-west-1.compute.amazonaws.com closed.
willem@willem-laptop:~$ ssh -i default.pem -L 8080:localhost:80 \
>          griduser@ec2-79-125-48-213.eu-west-1.compute.amazonaws.com
Last login: Mon Jun 28 20:16:08 2010 from happus.xs4all.nl
griduser@ec2-79-125-48-213:~$ grid-proxy-info
subject  : /O=Grid/OU=GlobusTest/OU=GridOnDemand/CN=Grid User/CN=1418860555
issuer   : /O=Grid/OU=GlobusTest/OU=GridOnDemand/CN=Grid User
identity : /O=Grid/OU=GlobusTest/OU=GridOnDemand/CN=Grid User
type     : RFC 3820 compliant impersonation proxy
strength : 512 bits
path     : /tmp/x509up_u1003
timeleft : 11:57:02
griduser@ec2-79-125-48-213:~$ globus-job-run localhost /usr/bin/whoami
torqueuser
griduser@ec2-79-125-48-213:~$ 
```

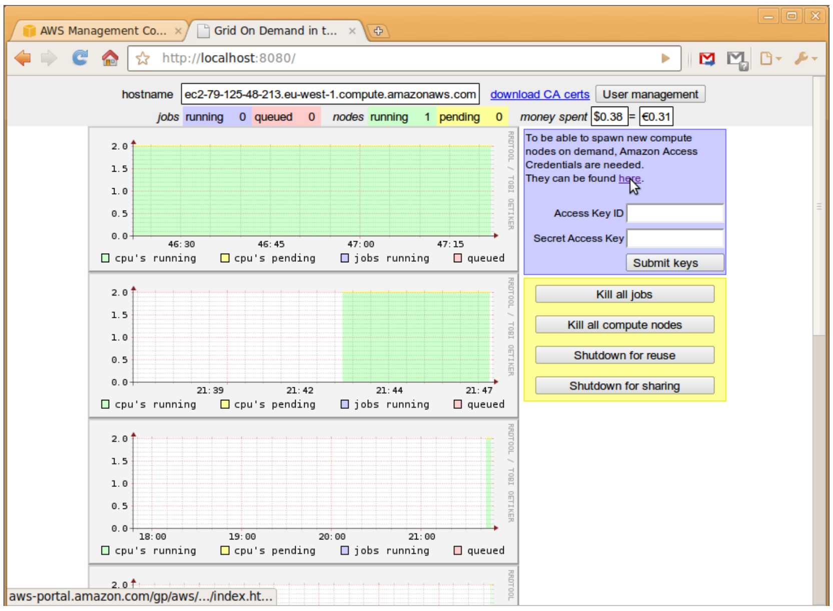
Figure 23: Grid on Demand can immediately be used

A.2 Configuring the cluster

To enable the Elastic Cluster functionality, we now have to provide the Amazon Access Credentials. A link to the location of those credentials is conveniently given in the web interface (figure 24). After copying (figure 25) and submitting (figure 26) the Amazon Access Credentials the cluster configuration pane shows up (figure 27).

In the configuration pane we can specify how many compute nodes should minimally and maximally be run. The instance type of compute nodes will be the one selected as the “startup type”.

To prevent excessive costs when many tiny tasks are submitted, a “startup window” setting can be given. New compute nodes will be launched to accommodate the minimum number of jobs that were pending during last number of seconds in “startup window”.



hostname ec2-79-125-48-213.eu-west-1.compute.amazonaws.com download CA certs User management

jobs running 0 queued 0 *nodes* running 1 pending 0 *money spent* $0.38 = €0.31

To be able to spawn new compute nodes on demand, Amazon Access Credentials are needed. They can be found here.

Access Key ID

Secret Access Key

Submit keys

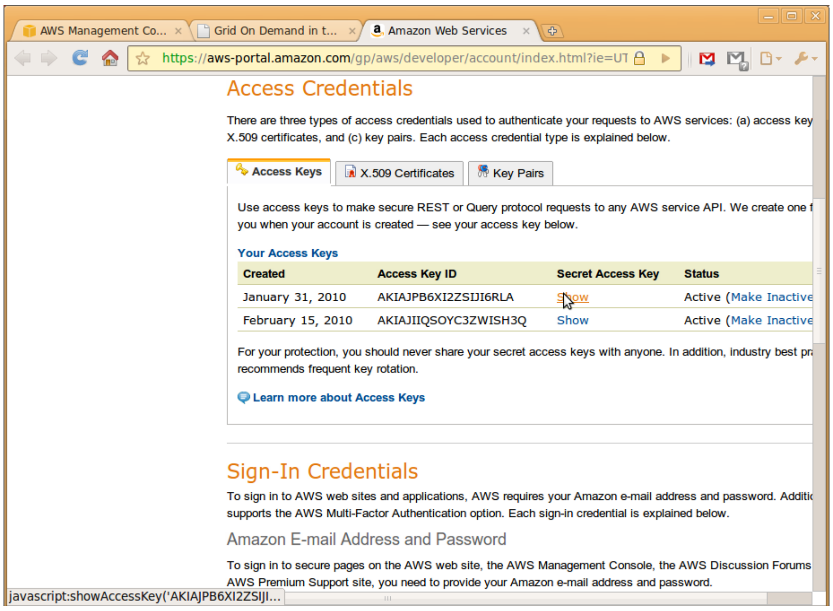
Kill all jobs

Kill all compute nodes

Shutdown for reuse

Shutdown for sharing

Figure 24: Click here to view your Amazon Access Credentials


AWS Management Co...
Grid On Demand in t...
Amazon Web Services
https://aws-portal.amazon.com/gp/aws/developer/account/index.html?ie=UT
javascript:showAccessKey('AKIAJPB6XI2ZSIJI...

Access Credentials

There are three types of access credentials used to authenticate your requests to AWS services: (a) access key
X.509 certificates, and (c) key pairs. Each access credential type is explained below.

Access Keys | X.509 Certificates | Key Pairs

Use access keys to make secure REST or Query protocol requests to any AWS service API. We create one f
you when your account is created — see your access key below.

Your Access Keys

Created	Access Key ID	Secret Access Key	Status
January 31, 2010	AKIAJPB6XI2ZSIJI6RLA	Show	Active (Make Inactive
February 15, 2010	AKIAJIIQSOYC3ZWISH3Q	Show	Active (Make Inactive

For your protection, you should never share your secret access keys with anyone. In addition, industry best pr
recommends frequent key rotation.

Learn more about Access Keys

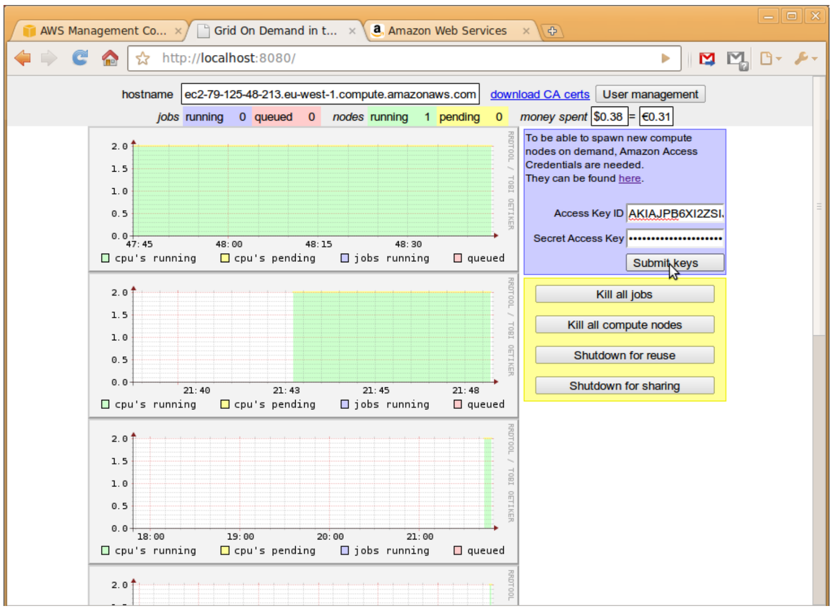
Sign-In Credentials

To sign in to AWS web sites and applications, AWS requires your Amazon e-mail address and password. Additio
supports the AWS Multi-Factor Authentication option. Each sign-in credential is explained below.

Amazon E-mail Address and Password

To sign in to secure pages on the AWS web site, the AWS Management Console, the AWS Discussion Forums
AWS Premium Support site, you need to provide your Amazon e-mail address and password.

Figure 25: Copy ...



hostname ec2-79-125-48-213.eu-west-1.compute.amazonaws.com

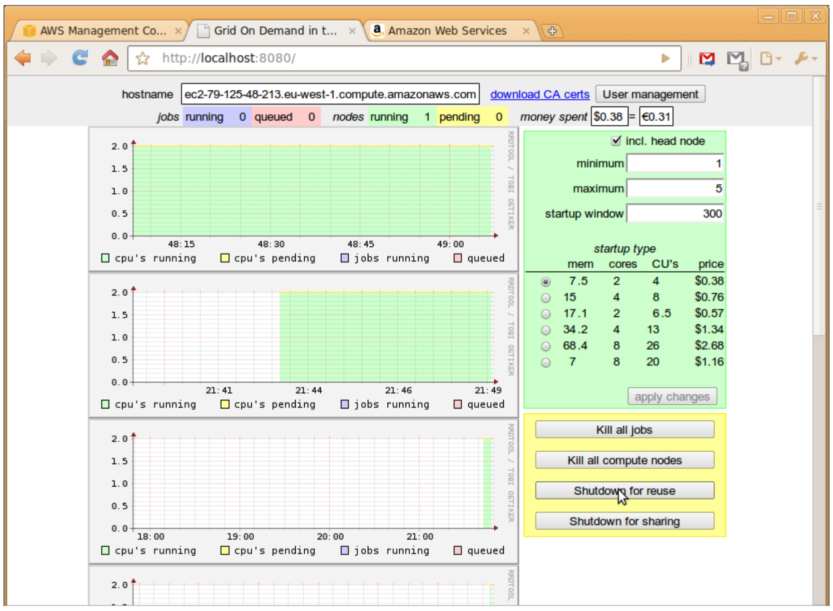
jobs running 0 queued 0 *nodes* running 1 pending 0 *money spent* $0.38 = €0.31

To be able to spawn new compute nodes on demand, Amazon Access Credentials are needed. They can be found here.

Access Key ID AKIAJPB6XI2ZSI

Secret Access Key ••••••••••••••••••••

Figure 26: ... and paste



hostname ec2-79-125-48-213.eu-west-1.compute.amazonaws.com

jobs running 0 queued 0 *nodes* running 1 pending 0 *money spent* $0.38 = €0.31

✓ incl. head node

minimum 1

maximum 5

startup window 300

startup type

	mem	cores	CU's	price
●	7.5	2	4	$0.38
○	15	4	8	$0.76
○	17.1	2	6.5	$0.57
○	34.2	4	13	$1.34
○	68.4	8	26	$2.68
○	7	8	20	$1.16

Figure 27: The Elastic Cluster is ready an can be configured

A.3 Real Grid certificate usage

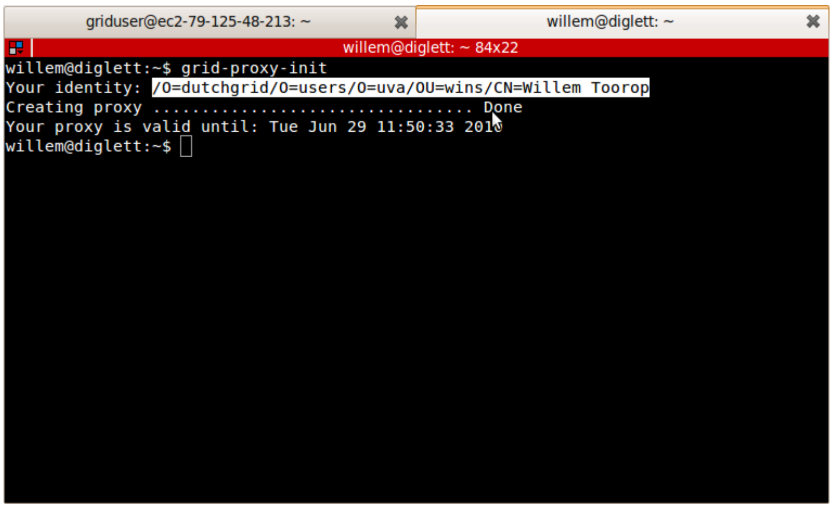
Grid on Demand can be used with real accredited CA signed Grid Proxy Certificates (PCs), enabling the potential to participate in bigger Grid collaborations.

In figure 28 we first initialize our own PC at our own workstation.

In figure 27 in the upper right corner of the interface is a button to enter the "User Management" of the interface, shown in figure 29. To authorize a Grid user to use Grid on Demand, the subject of his or her EEC (his or her identity) has to be added to the list by copying the subject in the input field and clicking on "add". Figure 29 shows that we have just authorized ourself.

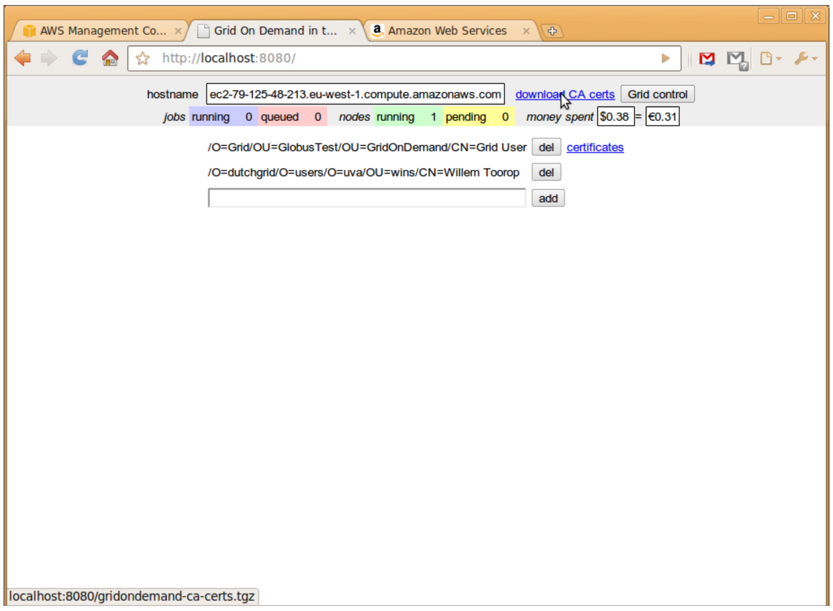
Because of the mutual authentication in Grid usage, we also have to be able to establish the identity of our Grid on Demand. For this, we have to add the newly created CA for this specific Grid on Demand instantiation to our list of trusted CAs. Right next to the "hostname" of our head node, is a link to download the CA certificate package.

Figure 30 shows how the CA certificate package is unpacked, so that the identity of Grid on Demand can be established and is thus authenticated to be used. As the preamble of the next subsection, we submit ten jobs each job lasting five seconds longer and the time between submissions gradually increasing.



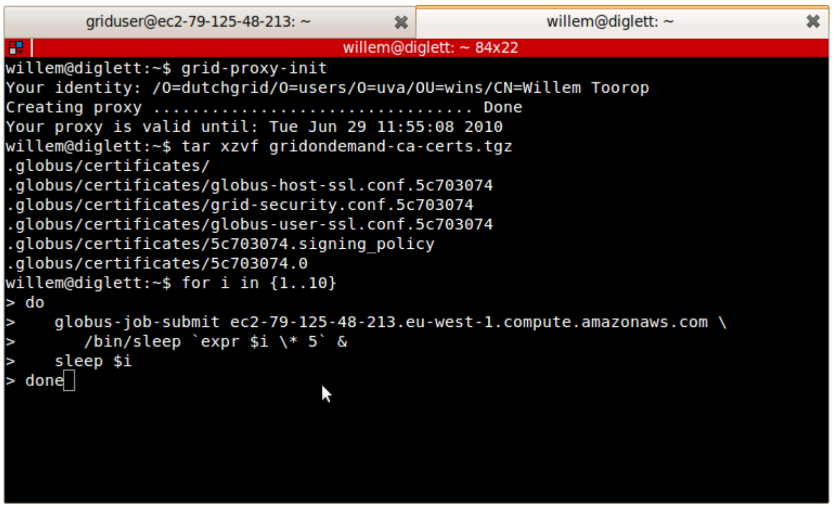
```
willem@diglett:~$ grid-proxy-init
Your identity: /O=dutchgrid/O=users/O=uva/OU=wins/CN=Willem Toorop
Creating proxy ................................. Done
Your proxy is valid until: Tue Jun 29 11:50:33 2010
willem@diglett:~$
```

Figure 28: Initializing a real Grid proxy certificate. Copying the identity line



```
http://localhost:8080/

hostname  ec2-79-125-48-213.eu-west-1.compute.amazonaws.com
jobs running 0 queued 0   nodes running 1 pending 0   money spent $0.38 = €0.31

/O=Grid/OU=GlobusTest/OU=GridOnDemand/CN=Grid User
/O=dutchgrid/O=users/O=uva/OU=wins/CN=Willem Toorop

localhost:8080/gridondemand-ca-certs.tgz
```

Figure 29: Identity line added to the list of authorized users for this resource



```
willem@diglett:~$ grid-proxy-init
Your identity: /O=dutchgrid/O=users/O=uva/OU=wins/CN=Willem Toorop
Creating proxy ................................ Done
Your proxy is valid until: Tue Jun 29 11:55:08 2010
willem@diglett:~$ tar xzvf gridondemand-ca-certs.tgz
.globus/certificates/
.globus/certificates/globus-host-ssl.conf.5c703074
.globus/certificates/grid-security.conf.5c703074
.globus/certificates/globus-user-ssl.conf.5c703074
.globus/certificates/5c703074.signing_policy
.globus/certificates/5c703074.0
willem@diglett:~$ for i in {1..10}
> do
>     globus-job-submit ec2-79-125-48-213.eu-west-1.compute.amazonaws.com \
>        /bin/sleep `expr $i \* 5` &
>     sleep $i
> done
```

Figure 30: Unpacking the CA package, and submitting “jobs”

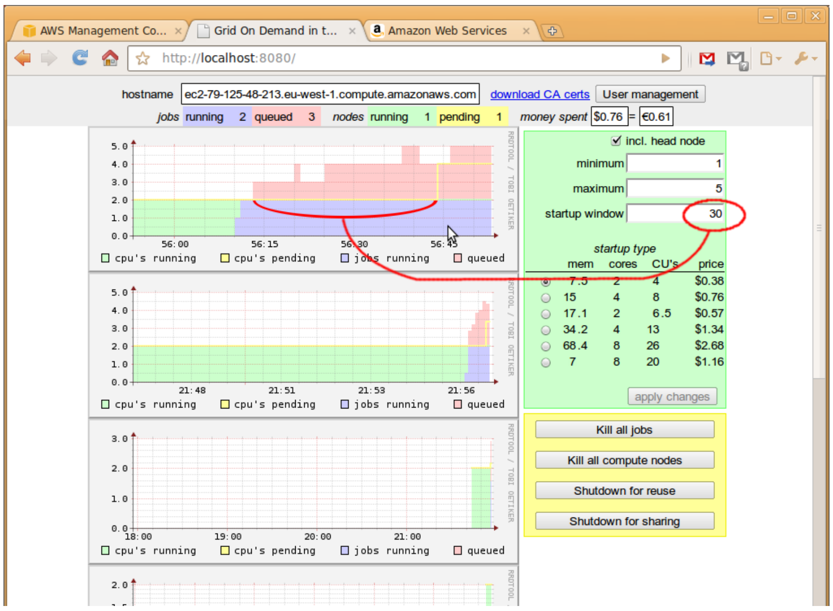
A.4 Elasticity of the cluster

In the RRDtool plots we can see the progress of the ten submitted jobs (figure 31). Because the jobs are so short, it would have been wasteful to just launch new compute nodes to accommodate pending jobs.

The “startup window” setting delivers the retention needed. In figure 31 it is set to 30 seconds. On 21:56.44 for the last 30 seconds, the lowest demand was for an extra slot to be able to run one job. Only one extra instance is needed. Because we have instantiated a 64-bit instance type, the smallest available instance type has two cores. On 21:56.44 it is launched potentially providing two extra slots for jobs, indicated by the yellow line to show it is pending and will soon join the cluster.

In figure 32 the just launched compute node joins the cluster indicated by the yellow (pending) slots becoming green (available/running). Apparently Torque also needs some time to be able to schedule the queued jobs on the freshly available slots, because on 21:57.52 the queued job is scheduled on a slot that became available on the head node. On 21:57.59 we therefor started an extra job, which is run on the new available compute node.

On Amazon, instances are payed per whole hour. Grid on Demand takes this into account and terminates idle compute nodes just before it would reach another whole hour of up time, and would cost another hourly fee. Figure 33 shows that on 22:54.04, the compute node is killed, 2 minutes and 40 seconds before it would reach a whole hour.



http://localhost:8080/

hostname ec2-79-125-48-213.eu-west-1.compute.amazonaws.com download CA certs User management

jobs running 2 queued 3 *nodes* running 1 pending 1 *money spent* $0.76 = €0.61

✓ incl. head node

minimum 1

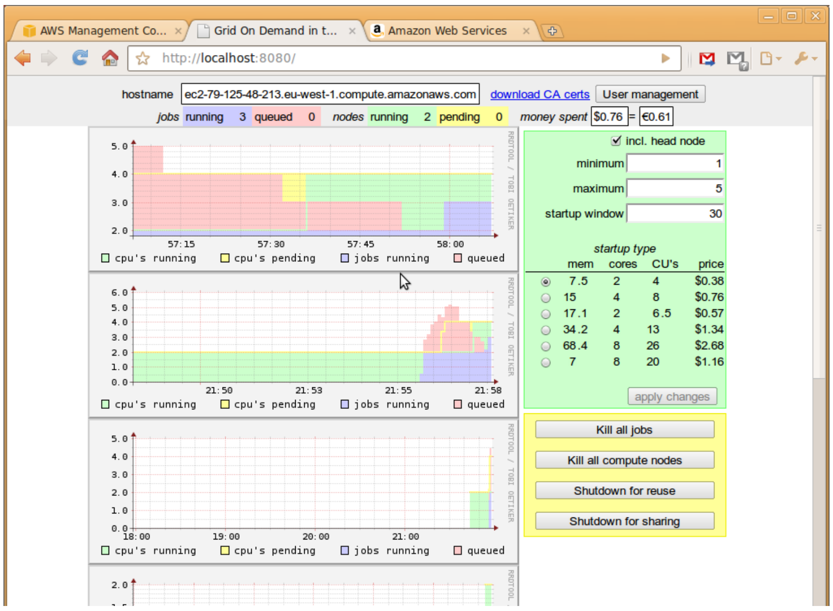
maximum 5

startup window 30

startup type

	mem	cores	CU's	price
●	7.5	2	4	$0.38
○	15	4	8	$0.76
○	17.1	2	6.5	$0.57
○	34.2	4	13	$1.34
○	68.4	8	26	$2.68
○	7	8	20	$1.16

Figure 31: Progress of the jobs and retention of launching compute nodes



hostname ec2-79-125-48-213.eu-west-1.compute.amazonaws.com

jobs running 3 queued 0 *nodes* running 2 pending 0 *money spent* $0.76 = €0.61

✓ incl. head node

minimum 1

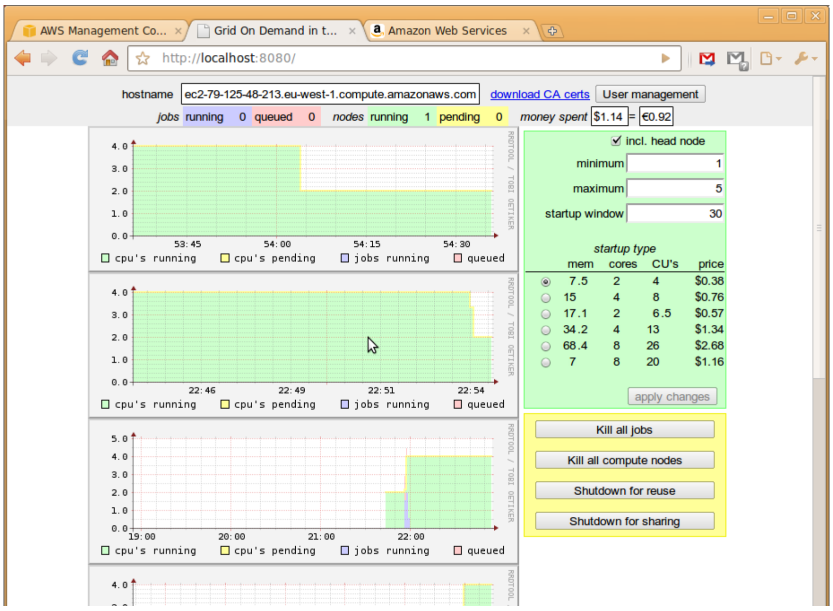
maximum 5

startup window 30

startup type

	mem	cores	CU's	price
●	7.5	2	4	$0.38
○	15	4	8	$0.76
○	17.1	2	6.5	$0.57
○	34.2	4	13	$1.34
○	68.4	8	26	$2.68
○	7	8	20	$1.16

Figure 32: The freshly started compute node joins the cluster



hostname ec2-79-125-48-213.eu-west-1.compute.amazonaws.com

jobs running 0 queued 0 *nodes* running 1 pending 0 *money spent* $1.14 = €0.92

✓ incl. head node

minimum 1

maximum 5

startup window 30

startup type

	mem	cores	CU's	price
●	7.5	2	4	$0.38
○	15	4	8	$0.76
○	17.1	2	6.5	$0.57
○	34.2	4	13	$1.34
○	68.4	8	26	$2.68
○	7	8	20	$1.16

Figure 33: The idle compute node terminates just before the new hourly fee

Grid on Demand determines the uptime of a compute node on the launch time reported by Amazon. It kills idle compute nodes 3 minutes before reaching a whole hour of uptime. Because the compute node was killed 2 minutes and 40 seconds, and not 3 minutes before reaching a whole hour of uptime, we can deduce that the by Amazon reported launch time is 20 seconds later then the time we actually started the compute node.

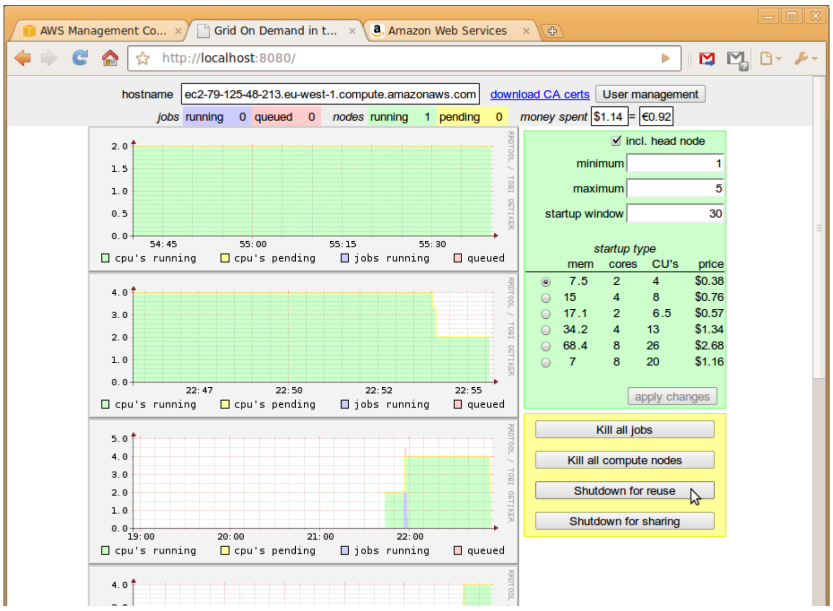
A.5 Saving your settings

Our instance of Grid on Demand has a newly created CA. We have been through the hassle of configuring our own Grid infrastructure to acknowledge this CA, and now we would like to save the Grid on Demand with its CA and the Amazon Access Credentials, so that we can instantiate more Grids like them. To do this we have to click "Shutdown for reuse" (figure 34).

Grid on Demand then shows the money spent and the instance identity (figure 35), which may now be used to create an AMI from, for *personal* use.

That new AMI can then be used to launch multiple Grid on Demands from, which are already equipped with the settings for your personal environment. Further developments can be made based on the AMI. The AMI could be equipped with software targeted at specific Grid applications, or specific services on the Collective layer could be enabled and configured.

When such a further developed AMI might be interesting for other parties, and one wishes to make it public, "Shutdown for sharing" shuts the instance down shredding[29] the CA, Amazon's Access Credentials and all other sensitive information. An AMI should then be created from the for sharing shutdown stopped instance, which may "safely" be made public.


hostname ec2-79-125-48-213.eu-west-1.compute.amazonaws.com

jobs running 0 queued 0 *nodes* running 1 pending 0 *money spent* $1.14 = €0.92

✓ incl. head node

minimum 1

maximum 5

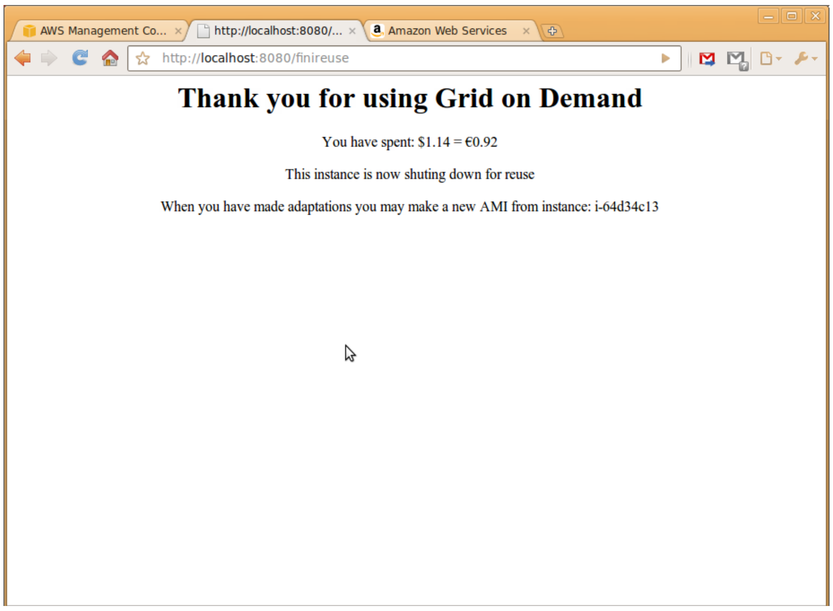
startup window 30

startup type

	mem	cores	CU's	price
●	7.5	2	4	$0.38
○	15	4	8	$0.76
○	17.1	2	6.5	$0.57
○	34.2	4	13	$1.34
○	68.4	8	26	$2.68
○	7	8	20	$1.16

Figure 34: Shutdown for reuse

[29] The content of the files containing the credentials are overwritten with random data before being removed.



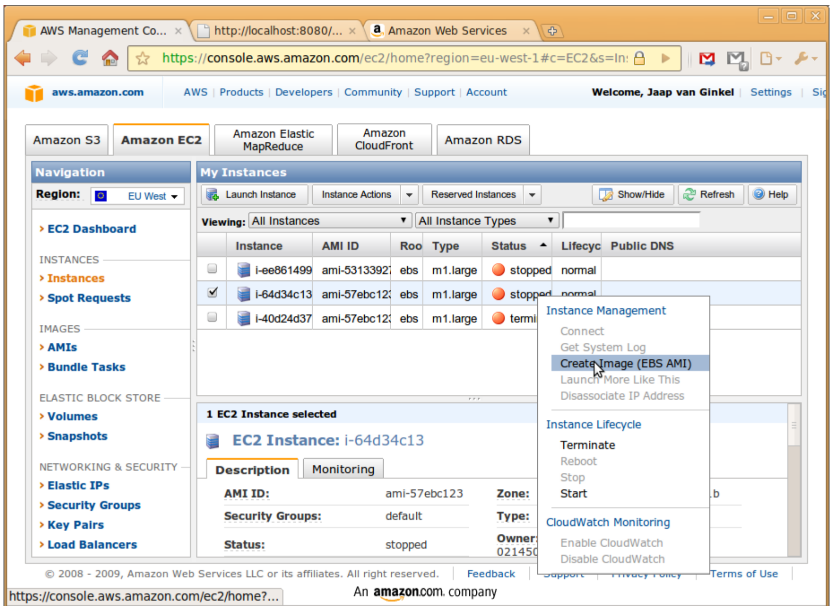
Thank you for using Grid on Demand

You have spent: $1.14 = €0.92

This instance is now shuting down for reuse

When you have made adaptations you may make a new AMI from instance: i-64d34c13

Figure 35: Thank you for using Grid on Demand



	Instance	AMI ID	Roo	Type	Status	Lifecyc	Public DNS
☐	i-ee861499	ami-5313392	ebs	m1.large	stopped	normal	
✓	i-64d34c13	ami-57ebc12	ebs	m1.large	stopped	normal	
☐	i-40d24d37	ami-57ebc12	ebs	m1.large	termi		

1 EC2 Instance selected

EC2 Instance: i-64d34c13

AMI ID: ami-57ebc123
Zone:
Security Groups: default
Type:
Status: stopped
Owner: 021450

© 2008 - 2009, Amazon Web Services LLC or its affiliates. All right reserved.

An amazon.com. company

Figure 36: Creating a new AMI based on the instance's volume

Glossary

Amazon Machine Image (AMI) A special type of virtual appliance which is used to instantiate (create) a virtual machine within the Amazon Elastic Compute Cloud (EC2)*. 10, 11, 24–26, 28, 31, 39, 51

Application Programming Interface (API) An interface implemented by a software component which enables it to interact with other software components.. 7, 12, 21, 23, 24, 31

Advanced School for Computing and Imaging (ASCI) Designed the DAS-3 distributed cluster.. 30

Certificate Authority (CA) An entity that issues digital certificates for use by other parties. It is an example of a trusted third party. CA's are characteristic of many Public Key Infrastructure (PKI) schemes*. 10, 11, 15–19, 21, 25–28, 36, 47, 51

Condor A specialized workload management system for compute-intensive jobs (Job Scheduler). The metaphor is vindicated in the 1993 Research Sampler Hunting for Wasted Computing Power:

> Like a vulture circling the desert, Condor scavenges for processing power that would otherwise be lost.

. 20, 21

Distributed ASCI Supercomputer 3 (DAS-3) A five-cluster wide-area distributed system designed by Advanced School for Computing and Imaging (ASCI).. 30, 32

The UvA part of the Distributed ASCI Supercomputer 3 (DAS-3-UvA) The University of Amsterdam (UvA) 32 node cluster that is part of Distributed ASCI Supercomputer 3 (DAS-3).. 30–34

Dynamic Host Configuration Protocol (DHCP) A computer networking protocol used by hosts (DHCP clients) to retrieve IP address assignments and other configuration information*. 26

Amazon Elastic Block Storage (EBS) Provides block level storage volumes for use with Amazon Elastic Compute Cloud (EC2) instances. Amazon EBS volumes are off-instance storage that persists independently from the life of an instance.. 10, 11, 24–26

Amazon Elastic Compute Cloud (EC2) Amazon's IaaS that provides resizable compute capacity in the cloud.. 2, 10, 21–24, 26, 37

End Entity Certificate (EEC) An X.509 Public Key Certificate issued to an end entity, such as a user or a service, by a Certificate Authority (CA).. 15–19, 22, 25–28, 36, 47

Amazon EC2 Elastic IP Addresses (EIP) Static IP addresses designed for dynamic cloud computing. An Elastic IP address is associated with your account, not a particular instance, and you control that address until you choose to explicitly release it.. 17, 23

Eucalyptus An open-source software infrastructure for the implementation of cloud computing on computer clusters which provides an interface that is compatible with the Amazon EC2 service*. Eucalyptus is an acronym for "Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems".. 24

* These descriptions are copied from Wikipedia in the second week of July 2010

Global Grid Forum (GGF) Organization established in 1998 to bring together developers, practitioners, and users of Grid technologies. In 2006, GGF became Open Grid Forum (OGF) after a merge with an industry group named Enterprise Grid Alliance (EGA).. 55

Globus Alliance An international association dedicated to developing fundamental technologies needed to build grid computing infrastructures.. 11, 15, 21, 22

Grid Resource Allocation Manager (GRAM) A software component of the Globus Toolkit that can locate, submit, monitor, and cancel jobs on Grid computing resources*. 12, 21, 26, 39

Grid File Transfer Protocol (GridFTP) An extension of the standard File Transfer Protocol (FTP) for use with Grid computing*. 12, 39

Grid Security Infrastructure (GSI) A specification for secret, tamper-proof, delegatable communication between software in a grid computing environment [14, 4]. GSI employs a number of different open standards to this end, such as: Transport Layer Security (TLS) [6], GSS-API [20], Public Key Infrastructure (PKI) and X.509 Certificates [15], but most notably X.509 Proxy Certificate (PC)s [25] which were originally developed to address the need for delegation in GSI. Originally developed for the Globus Toolkit (GT), GSI is now an Open Grid Forum (OGF) standard [27].. 12, 13, 18, 19, 36

Globus Toolkit (GT) The Globus Toolkit is an open source software toolkit used for building grids*. It is being developed by the Globus Alliance and many others all over the world. A growing number of projects and companies are using the Globus Toolkit to unlock the potential of grids for their cause.. 2, 10–13, 17, 20, 21, 25, 26, 30, 31, 36, 39

Hardware as a Service (HaaS) Cloud infrastructure services that delivers a physical managed computer infrastructure. Traditionally offered as an outsourcing service.. 5, 6

HyperText Transfer Protocol (HTTP) An Application Layer protocol for distributed, collaborative, hypermedia information systems*. 24

Infrastructure as a Service (IaaS) Cloud infrastructure services that delivers computer infrastructure, typically a platform virtualization environment as a service.. 5–8, 13, 17, 20–23

Internet Engineering Task Force (IETF) The IETF develops and promotes Internet standards, cooperating closely with the W3C and ISO/IEC standards bodies and dealing in particular with standards of the TCP/IP and Internet protocol suite*. 17

International Grid Trust Federation (IGTF) A body to establish common policies and guidelines between its Policy Management Authorities (PMA)'s members and to ensure compliance to this Federation Document amongst the participating PMAs.. 15, 27

Incubator Management Project (IMP) A Globus Alliance project that is responsible for helping new efforts to join the Globus Alliance.. 21

Job Scheduler A software application that is in charge of unattended background executions, commonly known for historical reasons as batch processing*. 20

Load Sharing Facility (LSF) A commercial computer software Job Scheduler sold by Platform Computing*. 20

* These descriptions are copied from Wikipedia in the second week of July 2010

Monitoring and Discovery System (MDS) A suite of web services to monitor and discover resources and services on Grids. This system allows users to discover what resources are considered part of a Virtual Organization (VO) and to monitor those resources. MDS services provide query and subscription interfaces to arbitrarily detailed resource data and a trigger interface that can be configured to take action when pre-configured trouble conditions are met.. 12

Machine Oriented Mini-server (MOM) A Portable Batch System (PBS) service provided by the `pbs_mom` command whose responsibility is to place jobs into execution as directed by the server, establish resource usage limits, monitor the job's usage, and notify the server when the job completes.. 20

Message Passing Interface (MPI) A specification for an API that allows many computers to communicate with one another*. 20

Nimbus An open source toolkit that allows you to turn your cluster into an Infrastructure-as-a-Service (IaaS) Cloud. Nimbus is developed by the Globus Alliance incubator project Virtual Workspaces. Nimbus is Latin for Cloud.. 21, 22, 24, 36

Organization for the Advancement of Structured Information Standards (OASIS) A not-for-profit consortium that drives the development, convergence and adoption of open standards for the global information society.. 56

Oracle Grid Engine (OGE) Previously known as Sun Grid Engine (SGE), previously known as CODINE (COmputing in DIstributed Networked Environments) or GRD (Global Resource Director), is an open source batch-queuing system, developed and supported by Sun Microsystems*. 20

Open Grid Forum (OGF) The community of users, developers, and vendors leading the global standardization effort for grid computing.. 11

Open Grid Services Architecture (OGSA) Describes an architecture for a service-oriented grid computing environment for business and scientific use, developed within the Global Grid Forum (GGF) [3]. OGSA was first suggested in [12].. 11

Open Science Grid Consortium (OSGC) An organization that administers a worldwide grid of technological resources called the Open Science Grid, which facilitates distributed computing for scientific research*. 21

Platform as a Service (PaaS) Cloud software services that delivers middleware, typically used by application developers.. 5, 6

Portable Batch System (PBS) The name of computer software that performs job scheduling*. 20, 21, 56

Proxy Certificate (PC) A certificate that is derived from, and signed by, a normal X.509 Public Key End Entity Certificate or by another Proxy Certificate for the purpose of providing restricted proxying and delegation within a Public Key Infrastructure (PKI) based authentication system [25].. 15–19, 27, 36, 39, 47

Public Key Infrastructure (PKI) A set of hardware, software, people, policies, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates*. 15, 53

Policy Management Authorities (PMA) Coordinates authentication for people and services. Manages authentication guidelines policies. Trust domain for research and academic grids.. 54

Amazon Simple Storage Service (S3) An online storage web service offered by Amazon Web Services*. 24

* These descriptions are copied from Wikipedia in the second week of July 2010

Software as a Service (SaaS) Cloud software services that deliver applications, typically accessed by a browser or Application Programming Interface (API).. 5, 6

Secure Copy (`scp`) Unix tool to copy files between hosts on a network. It uses Secure SHell (`ssh`) for data transfer.. 21

Software Development Kit (SDK) A set of development tools that allows for the creation of applications for a certain software package, software framework, hardware platform, computer system, video game console, operating system, or similar platform*. 12

System- and Network-Engineering (SNE) University of Amsterdam (UvA) master education in System and Network Engineering.. 2, 31, 36, 38

Simple Object Access Protocol (SOAP) A protocol specification for exchanging structured information in the implementation of Web Services in computer networks*. 23

Secure SHell (`ssh`) A program for logging into a remote machine and for executing commands on a remote machine. It provides secure encrypted communications between two untrusted hosts over an insecure network.. 27, 56

Transport Layer Security (TLS) Cryptographic protocols that provide security for communications over networks*. TLS is an Internet Engineering Task Force (IETF) standards track protocol, last updated in RFC 5246 [6], that was based on the earlier Secure Socket Layer (SSL) specifications developed by Netscape Corporation.. 15–17

Torque Resource Manager (Torque) An open source distributed resource manager providing control over batch jobs and distributed compute nodes. Its name stands for Terascale Open-Source Resource and QUEue Manager. It is a community effort based on the original Portable Batch System (PBS)*. 10, 20, 21, 25, 26, 28, 29, 31, 49

University of Amsterdam (UvA) A Dutch University located in Amsterdam.. 2, 30, 36, 38

Virtual Workspaces The Globus Alliance incubator project that develops Nimbus.. 21, 55

Virtual Machine (VM) A software implementation of a machine (i.e. a computer) that executes programs like a physical machine*. 10, 13, 21–25, 28

Virtual Organization (VO) A dynamic set of individual and/or institutions defined around a set of resource-sharing rules and conditions. All these virtual organizations share some commonality among them, including common concerns and requirements, but may vary in size, scope, duration, sociology, and structure*. 3–5, 12, 13

Web Services Description Language (WSDL) An Extensible Markup Language (XML)-based language that provides a model for describing Web services*. 21, 24

Web Services Resource Framework (WSRF) A generic framework for modeling and accessing persistent resources using Web services so that the definition and implementation of a service and the integration and management of multiple services is made easier [2]. WSRF is developed by Organization for the Advancement of Structured Information Standards (OASIS).. 21

Extensible Markup Language (XML) A set of rules for encoding documents in machine-readable form*. 23, 56

* These descriptions are copied from Wikipedia in the second week of July 2010

Acronyms

AMI Amazon Machine Image. 53
API Application Programming Interface. 53
ASCI Advanced School for Computing and Imaging. 53

CA Certificate Authority. 53

DAS-3 Distributed ASCI Supercomputer 3. 53
DAS-3-UvA The UvA part of the Distributed ASCI Supercomputer 3. 53
DHCP Dynamic Host Configuration Protocol. 53

EBS Amazon Elastic Block Storage. 53
EC2 Amazon Elastic Compute Cloud. 53
EEC End Entity Certificate. 53
EIP Amazon EC2 Elastic IP Addresses. 53

GGF Global Grid Forum. 54
GRAM Grid Resource Allocation Manager. 54
GridFTP Grid File Transfer Protocol. 54
GSI Grid Security Infrastructure. 54
GT Globus Toolkit. 54

HaaS Hardware as a Service. 54
HTTP HyperText Transfer Protocol. 54

IaaS Infrastructure as a Service. 54
IETF Internet Engineering Task Force. 54
IGTF International Grid Trust Federation. 54
IMP Incubator Management Project. 54

LSF Load Sharing Facility. 54

MDS Monitoring and Discovery System. 55
MOM Machine Oriented Mini-server. 55
MPI Message Passing Interface. 55

OASIS Organization for the Advancement of Structured Information Standards. 55
OGE Oracle Grid Engine. 55
OGF Open Grid Forum. 55
OGSA Open Grid Services Architecture. 55
OSGC Open Science Grid Consortium. 55

PaaS Platform as a Service. 55
PBS Portable Batch System. 55
PC Proxy Certificate. 55
PKI Public Key Infrastructure. 55
PMA Policy Management Authorities. 55

S3 Amazon Simple Storage Service. 55
SaaS Software as a Service. 56
scp Secure Copy. 56
SDK Software Development Kit. 56
SNE System- and Network-Engineering. 56
SOAP Simple Object Access Protocol. 56
ssh Secure SHell. 56

TLS Transport Layer Security. 56
Torque Torque Resource Manager. 56

UvA University of Amsterdam. 56

VM Virtual Machine. 56
VO Virtual Organization. 56

WSDL Web Services Description Language. 56
WSRF Web Services Resource Framework. 56

XML Extensible Markup Language. 56

References

[1] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, et al. Above the clouds: A berkeley view of cloud computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*, 2009.

[2] T. Banks. Web Services Resource Framework (WSRF) - Primer, December 2005. URL http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-01.pdf.

[3] Berry, Djaoui, Grimshaw, Horn, Maciel, Siebenlist, Subramaniam, Treadwell, Von Reich, and 24 J. 2006. The open grid services architecture, version 1.5, July 2006.

[4] R. Butler, V. Welch, D. Engert, I. Foster, S. Tuecke, J. Volmer, and C. Kesselman. A national-scale authentication infrastructure. *Computer*, 33(12):60–66, 2000. doi: 10.1109/2.889094. URL http://dx.doi.org/10.1109/2.889094.

[5] Nrenaissance Committee and National R. Council. *Realizing the Information Future: The Internet and Beyond.* National Academies Press, January 1994. ISBN 0309050448. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0309050448.

[6] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. URL http://www.ietf.org/rfc/rfc5246.txt. Updated by RFCs 5746, 5878.

[7] Remy Evard. An analysis of unix system configuration. In *LISA '97: Proceedings of the 11th USENIX conference on System administration*, pages 179–194, Berkeley, CA, USA, 1997. USENIX Association.

[8] I. Foster. What is the Grid? A Three Point Checklist. *Argonne National Laboratory & University of Chicago July*, 20, 2002.

[9] I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke. The physiology of the grid. *Grid computing: making the global infrastructure a reality*, pages 217–250, 2003.

[10] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10, 2008.

[11] Ian Foster. The anatomy of the grid: Enabling scalable virtual organizations. *Euro-Par 2001 Parallel Processing*, pages 1–4, 2001. doi: 10.1007/3-540-44681-8_1. URL http://dx.doi.org/10.1007/3-540-44681-8_1.

[12] Ian Foster. The physiology of the grid: An open grid services architecture for distributed systems integration, 2002.

[13] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11:115–128, 1996.

[14] Ian T. Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A security architecture for computational grids. In *ACM Conference on Computer and Communications Security*, pages 83–92, 1998. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.5657.

[15] R. Housley, W. Polk, W. Ford, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 3280 (Proposed Standard), April 2002. URL http://www.ietf.org/rfc/rfc3280.txt. Obsoleted by RFC 5280, updated by RFCs 4325, 4630.

[16] A. Savva D. Berry A. Djaoui A. Grimshaw B. Horn F. Maciel F. Siebenlist R. Subramaniam J. Treadwell J. Von Reich I. Foster, H. Kishimoto. The open grid services architecture, version 1.0., January .

[17] K. Keahey and T. Freeman. Contextualization: Providing one-click virtual clusters. In *IEEE Fourth International Conference on eScience, 2008. eScience'08*, pages 301–308, 2008.

[18] V. Korkhov, D. Vasyunin, A. Wibisono, V. Guevara-Masis, A. Belloum, C. de Laat, P. Adriaans, and LO Hertzberger. WS-VLAM: towards a scalable workflow system on the grid. In *Proceedings of the 2nd workshop on Workflows in support of large-scale science*, page 68. ACM, 2007.

[19] Stefanie Leimeister, Christoph Riedl, Markus Böhm, and Helmut Krcmar. The business perspective of cloud computing: Actors, roles, and value networks. In *Proceedings of 18th European Conference on Information Systems (ECIS 2010)*, Pretoria, South Africa, 2010. URL `http://home.in.tum.de/~riedlc/res/LeimeisterEtAl2010-preprint.pdf`.

[20] J. Linn. Generic Security Service Application Program Interface Version 2, Update 1. RFC 2743 (Proposed Standard), January 2000. URL `http://www.ietf.org/rfc/rfc2743.txt`. Updated by RFC 5554.

[21] P. Marshall, K. Keahey, and T. Freeman. Elastic Site: Using Clouds to Elastically Extend Site Resources. In *Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Melbourne, Australia*, 2010.

[22] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing. URL `"http://www.st.ewi.tudelft.nl/~iosup/ec2perf-sci-comp09cloudcomp.pdf"`.

[23] Simon Ostermann, Alexandru Iosup, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. A performance analysis of EC2 Cloud computing services for scientific computing. In *1st International Conference on Cloud Computing*. ICST Press, 2009. Accepted for publication.

[24] P. Saint-Andre and J. Hodges. Representation and verification of domain-based application server identity in certificates used with transport layer security. draft-saintandre-tls-server-id-check (Best Current Practice), June 2010. URL `https://datatracker.ietf.org/doc/draft-saintandre-tls-server-id-check/`.

[25] S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson. Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile. RFC 3820 (Proposed Standard), June 2004. URL `http://www.ietf.org/rfc/rfc3820.txt`.

[26] L.M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

[27] Von Welch. Grid Security Infrastructure Message Specification. GFD-I.078, May 2006. URL `http://www.ogf.org/documents/GFD.78.pdf`.

[28] J. Yu and R. Buyya. A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3(3):171–200, 2005.