REPORT ON IMPLEMENTATION OF USER LEVEL SLA FOR GLITE USING TYCOON

Document Filename:	**GLITE_TYCOON_1.6.odt**
Activity:	JRA1
Partner(s):	PDC/KTH, SICS
Lead Partner:	PDC/KTH
Document classification:	**PUBLIC**

Document review and moderation

	Name	Partner	Date	Signature
Released for moderation to				
Approved for delivery by				

Document Log

Version	Date	Summary of changes	Author
0.1	30/10/2006	Draft version	Lars Rasmusson, Niklas Wirström
1.0	24/11/2006	Extended all sections	Lars Rasmusson, Niklas Wirström
1.1	12/12/2006	Added figures, and changed design of “CE maager”	Lars Rasmusson, Niklas Wirström
1.2	19/01/2007	Added more details of virtual cluster creation, Future Work	Lars Rasmusson, Niklas Wirström
1.3	1/02/2007	Restructured and rewritten	Lars Rasmusson, Niklas Wirström
1.4	17/04/2007	Added more details of installation	Amir H. Payberah
1.5	28/05/2007	Modified installation	Amir H. Payberah
1.6	29/05/2007	Added some extra information as appendix	Amir H. Payberah

Contents

1. INTRODUCTION

This document describes the resulting system of integrating the gLite and Tycoon platforms. Section 2 describes the two separate sub systems, and the properties of integrated gLite-Tycoon system. Section 3 describes how the system is implemented, and how a gLite Computing Elements (CE) and Worker Nodes (WN) are set up on virtual machines to form *virtual clusters*. Section 4 describes how gLite-Tycoon is installed on a site.

1.1. MOTIVATION

The purpose of the BalticGrid JRA1 task is to provide user-level service level agreements, i.e. the possibility for users to acquire more (or less) resources in a well defined manner [5]. The resource consumption will be accounted for, to prevent some users from asking for so much resources that other users are starved. We use the Tycoon market-based Utility Computing infrastructure to create and manage a SLA-capable execution environment hosting unmodified gLite Computing Elements.

A market-based resource allocation system makes efficient use of its resources since it encourage users not only to choose resources with currently low demand, but also to choose time periods when demands are low. It also allocates more resources to more important tasks, if tasks are funded according to importance. Thus, the service level of a task or group of tasks is ultimately determined by its level of funding, relative to other tasks [2].

2. SYSTEM DESCRIPTION

Here we describe the Tycoon-gLite system. This is done by first briefly describing the current implementation of the Tycoon system, and then (also briefly) describe the functionality of the gLite system. Last in this section we describe the integrated system.

2.1. TYCOON

A detailed description of the Tycoon system can be found in [1].

Tycoon is a market-based system that manages computing resources like CPU, RAM, disk, network bandwidth, etc., for users that compete for the resources. The aim of Tycoon is to provide users with an intuitive and extremely simple way to specify resource demand, namely through a single variable, the "spending-rate".

The Tycoon system consists of a Bank, Auctioneers, a Service Location Server (SLS) and Agents. Auctioneers are responsible for allocating and to do accounting for resources on the host where they are installed. The Auctioneers publish themselves in an SLS. Agents, which each acts on behalf of a resource user, queries the SLS for information about prices and location of resources. After selecting the economically most efficient set of Auctioneers, an Agent uses the Bank to transfer money from its user's bank account to that of the Auctioneer.

Tycoon enables users to create private Virtual Machines (VM), and assigns resource shares to the VM's, proportionally to the spending-rate. Tycoon also contains a secure banking infrastructure that provides authenticated money transfer between the users and the resource providers.

The current implementation of Tycoon uses Xen as the virtualization layer. A detailed description of Xen can be found in [3]. Xen can dynamically create and reallocate resources between virtual machines. One of the VMs that run on top of Xen is a *privileged domain* that has access to the physical devices (disk, network cards, etc.). It is called *domain0*, and is the domain in which the Tycoon Auctioneer runs. Many users run concurrently on the same physical machine, isolated from each other, in their individual virtual machines.

The privileged domain acts as a NAT firewall for the user VMs. Each Tycoon VM has at least one port which is reachable from the outside. It is forwarded with Linux *iptables* to the VM's SSH port.

Users can bid for additional TCP ports from Domain0's host address. Such externally visible ports make incoming external communication possible, and avoids the overhead of setting up a VPN.

It is also possible for the Auctioneer to possess a pool of IP addresses for which the users may bid. In this way a VM can be assigned a public IP address.

Figure 1 shows the steps involved in a Tycoon host account creation: 1) An Tycoon Agent queries an SLS for information about Auctioneers and prices. 2) The SLS responds with this information, and the Agent selects a set of Auctioneers. For each Auctioneer in the set, the Agent then does the following: 3) The Agent contacts the Bank, requesting a money transfer from its user's account to that of the Auctioneer. 4) The Bank sends back a signed receipt to the Agent. 5) The Agent forwards the receipt to the Auctioneer which then creates an Tycoon account and a VM. 6) When the VM has booted, the user can log in to it and start to use it.

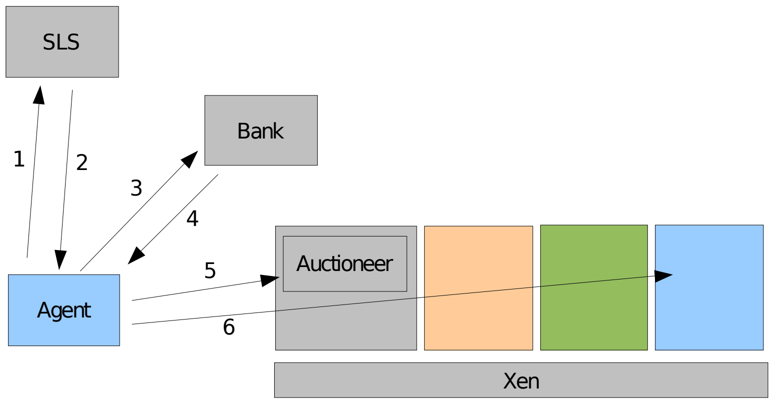

Fig. 1: Tycoon host account creation.

2.2. GLITE

The components of the gLite system can be divided into two categories: Site level components and Grid level components.

Here and in the rest of this report, we refer to a *site* as subsystem of gLite which provides job and data management systems, and information and monitoring services regarding these. The job management system is comprised of a Computing Element (CE) and a number of Worker Nodes (WN). The CE acts as an interface to the Grid level, and is responsible for delegating jobs to the WNs which performs the actual work. Each CE possesses a *host certificate* for authentication.

The data management system is responsible for storing data, and to map globally unique identifiers to local file names. The information service is responsible for publishing information of the state of, for example, CEs or jobs.

On the Grid level, components responsible for Grid security, information services, and workload and data management systems are found. The information service on this level is responsible for binding information publishers at the site level and information consumers such as a user querying the status of a job. The data management system keeps track of where particular files are stored, and the workload management system (WMS) is responsible for matching jobs with available CEs.

A Virtual Organization Management Services (VOMS) server constitute the Grid security. A VOMS server acts as a repository for information of user authorization. Components such as CEs and WMSs, use a VOMS sever's SOAP interface to receive such information.

Further information of the gLite system can be found in [4].

2.3. THE TYCOON-GLITE INTEGRATION

The BalticGrid is a production environment as well as a research environment. To avoid disturbing the users of the BalticGrid unnecessarily, we provide a heterogeneous gLite system, in which some compute nodes can run the unmodified gLite software, and other nodes can run gLite-prepared Tycoon Auctioneers.

The gLite WMS is used for submitting jobs and Tycoon SLSs are used for discovering resources for creation of virtual clusters. Each site runs an SLS for each LAN on the site that contains gLite-prepared Tycoon Auctioneers. All Auctioneers on the same LAN registers in the same SLS.

VO managers can request the Auctioneers to create several VMs that together constitute a virtual gLite compute cluster. In each virtual cluster there is one CE and an arbitrary number of WNs. Only members of the VO owning a virtual cluster are able to submit jobs to it.

The CE belonging to virtual clusters register with the already existing BalticGrid infrastructure, and look just like regular CEs to the BalticGrid users. A user that submits a job to the BalticGrid can either let the gLite Workload Management System (WMS) pick a suitable computing element for its job, or specify a CE belonging to one of its VO's virtual clusters.

The virtual clusters use TORQUE as the Local Resource Management System (LRMS).

The system is also depicted in Figure 2. The figure shows a site with both ordinary gLite nodes and gLite-prepared Tycoon Auctioneer nodes, all using the same WMS. All auctioneers in the picture are on the same LAN, and registers therefore in the same SLS. Each VO has a VOMS server which its CE uses to retrieve VO-specific information.

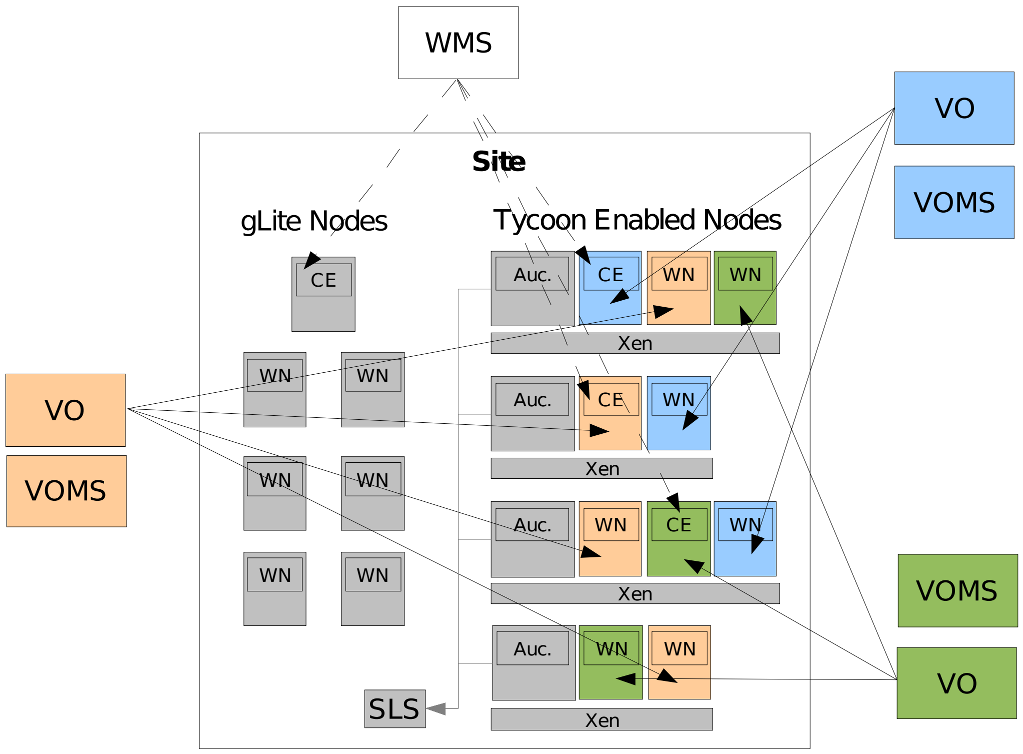

Fig. 2: Three different VOs (orange, blue and green) and their virtual clusters on a specific site.

2.3.1. Example Scenario

The following is a description of an example scenario, where a VO manager creates a virtual cluster on a Tycoon enabled gLite site. For further information about the commands named below, see Section 3.2

1. The VO manager has previously been allocated a budget on a Tycoon-managed bank account. The VO manager also has a list of SLSs, each corresponding to a LAN where gLite-prepared Tycoon Auctioneers are installed.
2. The VO manager issues the following command to get information from the different SLSs:

```
multi_wn_agent list <sls_list> <budget> <deadline> <max_nodes>
```

   Here, *<sls_list>* is the list of SLSs, *<budget>* is the amount of money to be spent, *<deadline>* is the amount of seconds over which the budget is to be spent, and *<max_nodes>* is the maximal number of nodes wanted. This outputs a list of Auctioneers for each SLS in *<sls_list>*.
3. To create a CE, the VO manager chooses an Auctioneer from one of the lists output by the *multi_wn_agent* list command and issues the following command:

```
ce_agent create <ce_name> <auctioneer> <budget> <ce_ip> <ce_netmask>
```

4. Here, *<ce_name>* is the name of the VM to be created, *<auctioneer>* is the IP address to the Auctioneer where the VM is to be created, *<budget>* is the amount of money that is to be transferred from the VO managers bank account to the Auctioneer's ditto. *<ce_ip>* and *<ce_netmask>* are the VMs public IP address and netmask, respectively.
5. The VO manager sets the environmental variable CE_SLS to point to the SLS corresponding to the chosen Auctioneer.
6. To create worker nodes, the VO manager issues the following command:

```
multi_wn_agent create <ce_name> <wn_pref> <budget> <deadline> <max_nodes>
```

Here, *<wn_pref>* is a tag that precedes the name of each WN to be created. This command outputs a list of Auctioneer - VM name pairs, which can be used when rebalancing the bids.

7. Users belonging to the specific VO can at this point start submitting jobs to the virtual cluster. As usual, the user first creates a VO Management System (VOMS) Proxy with the followng command:

```
voms-proxy-init -voms <VOMS_SERVER>
```

where *<VOMS_SERVER>* is a VOMS server that holds information of the VO to which the virtual cluster and the user belongs. Now, the user can submit a job by issuing:

```
glite-job-submit -r <CE_RESOURCE> <JDL>
```

where *<CE_RESOURCE>* specifies the CE's IP address, its Gatekeeper port number and the desired service; and *<JDL>* is a normal Job Description Language file.

8. To reblance the bids over the set of existing WNs, the VO manager issues the following command:

```
multi_wn_agent rebalance <rate> <info_list>
```

where <rate> is the spending rate and <info_list> is the list output in step 5.

2.3.2. Funding of Resources

The first resource allocation is the allocation of money to the Virtual Organizations (VO). The amount of money allocated can be different for each VO, and is determined by a human committee.

The money is used by a VO to purchase resources from the Tycoon-managed machines, which creates a virtual cluster for the virtual organization. The amount of resources allocated to the virtual cluster is determined by the current demand from the other virtual organizations in the BalticGrid. When the users in the VO need more resources, they will ask the VO manager to increase the spending rate. The rate will be decreased when the demand is low again.

A VO manager uses a software agent that gets as input a spending rate, measured in EUR/second. The agent is used to select optimal Auctioneers from a given SLS, create and configure CE and WN nodes, and to rebalance the VO's total funds over the already created VMs, on certain time intervals. The gLite-Tycoon system itself does not provide information of on which WN a specific job is being run. Therefore all jobs executed on the same virtual cluster have the same probability to benefit from an incrementation of the spending rate. This means that a low priority job actually may have greater computing power than a high priority job running on the same virtual cluster. Such information can, however, be manually extracted from the system, in which case it can be used to manually allocate more or less money for a specific WN.

The VO manager can also use the Agent to, at any time, create new nodes or delete existing ones. When the Agent selects Auctioneers or rebalances the total funds over an already existing set of machines, it is done so that the marginal price per selected resource is the same (and sum up to the spending rate). In the case of selection, also, all the other machines are more expensive. This is referred to as the *best response* algorithm [1].

In the current implementation the number of nodes to run on is chosen as the number that maximizes the amount of resources per EUR spent, but in some cases, the VO may want to enforce a bigger or smaller number of nodes in the virtual cluster. This can be done by manually creating or deleting nodes.

3. IMPLEMENTATION

Here we explain how the system is implemented and plugged in to the existing gLite infrastructure, and the way CEs and WNs are created to form a virtual cluster.

3.1. GENERAL TYCOON ACCOUNT CREATION

When a Tycoon Agent creates an account at an Auctioneer, it first makes a bank transfer from its user's bank account to that of the owner of the Auctioneer. The bank returns a signed receipt to the Agent. The agent sends a *create request* to the Auctioneer, attaching the receipt and options regarding the configuration of the user's Tycoon account. These options include user name, public keys used for future log-ins, etc. It is also possible to specify the path to a file system to be used by the new VM.

The Auctioneer now creates a new Tycoon user account, and requests Xen to create a new VM. The user's public key is made an *authorized key* of the root account of the VM.

Xen assigns a locally unique number to each VM. We refer to this number as *domain_ID*.

3.1.1. Network Configuration

When a new VM is created, Xen creates two virtual network interfaces. One in the privileged domain, and one in the new VM. Xen then decides MAC addresses for these interfaces.

The creation script can be called with an argument specifying a public IP address for the VM's virtual interface. If no such address is specified, the interface will be assigned an address on the form *10.X.Y.Z*, where the four leftmost bits of *X* are specific to the particular physical host. The remaining bit string (i.e. the four rightmost bits of *X*, and the bits of *Y* and *Z*) is the binary representation of *domain_ID*2*.

The creation script assumes that the privileged domain has access to a local DNS server in order to add and remove entries as VMs are created an deleted.

It is also assumed that the privileged domain runs a DHCP server for its VMs.

1. The back-end interface is assigned an IP address according to the procedure described above.
2. If a public IP address was specified in the arguments, that address is assigned to the VM's virtual network interface. Otherwise it is assigned an address on the form *10.X.Y.Z* as described above.
3. If no public address was specified, an iptables rule specifying to *masquerade* all IP packets with source address matching that of the VM's network interface, and that are not destined to hosts with an IP address matching *10.0.0.0/8*.

If a public address was specified, an iptables rule specifying to *accept* all IP packages destined to this address. Also, the privileged domain is set up to do proxy-ARPing for the VM.

4. An entry for the VM's host name is created in the */etc/dhcpd.conf* file in the privileged domain. This overwrites any already existing entry with that name.
5. The DHCP Server in the privileged domain is reloaded with the new configuration.
6. The local DNS is configured to handle the new host. This is done using the *nsupdate* command.

3.1.2. Computing Elements and IP Addresses

In Tycoon, each user domain is NATed by default. The authentication in gLite unfortunately uses reverse DNS for host authentication, which is problematic for NATed compute clusters. The reason is that the host name is entered in the certificate subject, and the communicating partner verifies the host name of a caller with reverse DNS. So all CEs behind a NAT will appear to have the same host name. Another problem with using NAT is that a large number of ports have to be forwarded to each CE. If some ports are not reconfigurable, they will not allow two virtual CEs share the external IP address of the same physical machine.

For this reason, CEs have public IP addresses and host names published in a DNS. Because of this, each CE must have a host certificate. How these are provided can be implemented in the following ways:

1. The site is responsible for providing the host certificates to the virtual CEs. Since a host certificate with a Distinguished Name (DN) of ce.mysite.se, can be used by hosts named ce-1.mysite.se, ce-2.mysite.se, etc., it is sufficient to have a single certificate that is used for all CEs on the site.

   Since the same certificates then are used by multiple CEs belonging to different VOs, the certificate's private key must not be readable to the VO manager. Therefore, only the site administrator, and not the VO manager can be allowed to log in as root on the CE. In this way the virtual CEs have the same level of credential as an ordinary CE belonging to the same site.
2. The VO is responsible for providing the host certificates for its CE. In this way, the VO manager can be allowed to administrate the CE. However, this means that each VO needs to, either, have a separate host certificate for each site in the Grid and let the site configure the proper DNS entries (both direct and reversed), or to choose a certificate with a DN from its own network domain, and configure a DNS to point direct DNS lookups of that name to the IP address (belonging to the site) of the CE. In the latter case the site must configure a DNS to point reversed DNS lookups to the host name.

We have chosen the first strategy for providing host certificates for CEs. However, this means that the Tycoon Auctioneer software needs to be slightly modified.

3.1.3. Worker Nodes and IP Addresses

WNs do not need public addresses for communication between each other or with a CE. Since all nodes resides on the same LAN, the following strategy can be used:

When the VM is created, it is equipped with a virtual network interface. In the privileged domain, another virtual network interface which is connected to that of the VM, is created.

All virtual interfaces on a particular physical machine, are assigned IP addresses from a predefined range, specific to that machine. In order to allow VMs on one physical machine to communicate with VMs on another, each privileged domain have a routing entry for each of the predefined address ranges specifying the associated privileged domain as the gateway.

3.1.4. File System Configuration

Each VM has its own file system. We use LVM (Logical Volume Management) to create copy-on-write *snapshots* of a pre-installed file system. An LVM snapshot can be created within seconds, which makes fast VM creation possible.

Before the VM is booted, the snapshot is mounted and the user's public SSH key is copied to it.

3.2. VIRTUAL CLUSTER CREATION

To create a virtual cluster, the VO manager creates a VM containing a CE and several others containing WNs. When the VMs are booted they use a filesystem where the gLite software has been preinstalled, but not yet configured. Configuration of the VMs include specifying the CE's IP address to the WNs and vice versa, and settings regarding restricting the virtual cluster to only be used by one VO. This will be done automatically by a software agents, without the interaction of the VO managers or BalticGrid administrators. Below, we give a more detailed explanation of how the virtual clusters are created. The scripts and commands in the following sections are prototypes.

3.2.1. Selecting a Site for Virtual Cluster Creation

A virtual cluster can be created on any site which have installed gLite-prepared Tycoon Auctioneers and the other requirements found in Section 4.

It is assumed that each VO manager possesses a list of SLSs, where each SLS corresponds to a LAN with gLite-prepared Tycoon Auctioneers. The VO manager can then use the following command to list the *<max_nodes>* most economical machines for each SLS in the list:

```
multi_wn_agent list <sls_list> <budget> <deadline> <max_nodes>
```

Here, <*sls_list*>, is the list of SLSs, <*budget*> is the amount of money to be spent, <*deadline*> is the amount of seconds over which the budget is to be spent, and <*max_nodes*> is the maximal number of nodes wanted. The command runs the *best response* algorithm [1] for each SLS in <*sls_list*>, and outputs a list of Auctioneers. The list is grouped by LAN (corresponding to an SLS), and specifies the amount of money that should be spent on each Auctioneer and the resulting utility value of doing so, given that the VO manager chooses to create the virtual cluster on that particular LAN.

By comparing the utility values of the Auctioneers of different LANs, the VO manager can select an economically good set of machines for creation of a virtual cluster. When this choice has been made, the VO manager first picks a machine on which he/she creates a CE, and then WNs can be created.

3.2.2. CE Creation

To create a CE on a specific Auctioneer the following command can be used:

```
ce_agent create <ce_name> <auctioneer> <budget> <ce_ip> <ce_netmask>
```

Here, <*ce_name*> is the host name of the VM to be created, <*auctioneer*> is the IP address or host name of the Auctioneer where the VM is to be created, <*budget*> is the amount of money that is to be transferred from the VO managers bank account to the Auctioneer's ditto. <*ce_ip*> and <*ce_netmask*> are the VMs public IP address and netmask, respectively. These arguments should in future versions of the system be decided by the site instead of the VO manager.

The *ce_agent* also needs some additional parameters that can either be set as environmental variables or edited directly in the *ce_agent* script.

The Auctioneer uses a special creation script to set up the VM as a CE. Instead of making the VM's root account accessible to the VO manager, the script creates a special user account (*voadmin*) on the VM for which the VO manager's public key is made an *authorized key*. In order to let a VO manager configure the CE, and add or delete WN's to the Local Resource Management System (LRMS), this user is made able to execute special scripts with root permissions, using the *sudo* command. In this way the host certificate's private key can remain private to the site administrator.

When the VM's filesystem has been set up, the VM is booted and it can be configured. The Agent then logs in to the machine (as *voadmin*) and runs a configuration script with root privileges. This script uses the gLite configuration tool and the gLite site configuration file to configure the CE. Values in the site configuration file has partly been specified by the site administrator when setting up the Auctioneer, but some values are VO specific and must be set up at this point: A VOMS server needs to be specified in the *VO_<VONAME>_VOMS_SERVERS* variable in the site configuration file. Here, <*VONAME*> is the name of the VO. Additional VO specific variables that need to be specified includes user- and group settings for local UNIX accounts.

All VO specific settings will be read from files by the VO's Agent and handed over to the configuration script.

The gLite configuration tool uses the source command to read configuration parameters from a file. No special check is made which results in that any command in the file may be executed. Therefore the file supplied by the VO manager cannot simply be appended to the site manager's configuration file since it may contain malicious code. We solve this in the following way:

1. When a VO manager creates a CE, he/she provides a file containing the VO specific configuration parameters.
2. A script extracts lines starting with a recognized variable name and rewrites it to have the following format:
   <var_name>=$(echo '<uu_encoded_value>' | uudecode).
   Here, *<var_name>* is the variable name *<uu_encoded_value>* is the 64 bit *uu encoded* version of the value specified by the VO manager.

This ensures that only variable assignment is carried out when the configuration file is sourced.

3.2.3. WN Creation

To create a WN the following command can be used:

```
wn_agent create <wn_name> <auctioneer> <ce_name> <budget>
```

Here, *<wn_name>* is the name of the WN, *<auctioneer>* is the IP address of the Auctioneer where the VM is to be created, *<ce_name>* is the name of the CE that should be configured to use the WN, and *<budget>* is the amount of money that is to be transferred from the VO managers bank account to the Auctioneer's ditto.

The Auctioneer creates a VM in the normal Tycoon manner and boots it. When the VM is booted, the agent copies the site configuration file from the CE to it, and logs in to the VM as root and use the gLite configuration tool to configure it as a WN. The agent also adds the new WN to the CE's WN list.
However, the LRMS on the CE needs to be reconfigured before the new settings can be used. This is done with the following command:

```
wn_agent update <ce_name>.
```

For efficiency, multiple WNs can be created and configured before this command is called.

To set up multiple WNs, it is convenient to use the following command:

```
multi_wn_agent create <ce_name> <wn_pref> <budget> <deadline> <max_nodes>
```

This command runs the *wn_agent create* command for the most economical set of nodes on the LAN where the CE is created. *<wn_pref>* is a tag that is to precede the name of each WN to be created, *<deadline>* is the length of the period over which the budget should be spent, and *<max_nodes>* is the maximum number of nodes that are to be created. When WNs have been created and configured the script automatically runs the *wn_agent update <ce_name>* command.
The command outputs a list of Auctioneer - VM name pairs, which can be used when rebalancing the bids.
Using this command can cause a WN to be created on the same physical machine as the CE of the same virtual cluster. This would result in that the VO competes with itself for the same resources.

3.2.4. Funding a Virtual Cluster

Funding nodes of a virtual cluster is done in the usual Tycoon manner. That is, the Agent transfers money from its user's account to the account of the owner of the Auctioneer. The bank sends back a signed receipt to the Agent. The Agent sends the receipt along with a *funding request* to the Auctioneer, specifying which account to be funded. The Auctioneer adjusts the user's local account balance according to the funding amount.

3.2.5. Rebalancing Bids of a Virtual Cluster

To rebalance the bids, the following command can be used:

```
multi_wn_agent rebalance <rate> <wn_info>
```

Here, *<rate>* is the spending rate (i.e. budget / deadline) and *<wn_info>* is the list of Auctioneer - VM name pairs that was output by the *multi_wn_agent create* command. The command calculates the total amount of money remaining on the accounts of the VMs specified in the *<wn_info>* list, and rebalance the money over these accounts so that the most economical configuration is obtained.

4. INSTALLATION

This section describes how to install the gLite-Tycoon system on a site. The installation regards the installation of the virtualization and Tycoon software, the preparation of a filesystem image to be used by the VMs acting as CEs or WNs.

4.1. INSTALLING TYCOON-GLITE PLATFORM

To enable a site to create virtual clusters and CEs, the Tycoon software must be installed on a set of the physical nodes. Each site also needs a DNS at its disposal, and an SLS for each LAN containing gLite-prepared Tycoon Auctioneers.

The following must be done on each of the physical cluster nodes that will host virtual cluster nodes:

1. Install the Xen virtualization software.
2. Install a Fedora Core root filesystem.
3. Install NTP.
4. Install the gLite-prepared Tycoon Auctioneer software.
5. Set up routing tables.
6. Install keys for permission to modify the local DNS.
7. Install a DHCP server.
8. Install ScientificLinux file system images for use by the Tycoon-managed CEs and WNs. These images should have the gLite software pre-installed so that only the gLite configuration step is necessary when creating new virtual cluster nodes.

In the remaining part of this section it will be explained how to install Xen, Tycoon-gLite and also how to prepare Tycoon-gLite filesystem image.

4.2. XEN

To enable the machine to support Tycoon Auctioneer, Xen should be installed. In this section installing and working with Xen is explained.

4.2.1. Installing Xen

Installing Xen can be done in two different ways: from source code or from its RPM. To install Xen from its RPM, the following command can be used.

```
# yum install xen kernel-xen0 kernel-xenU
```

This command installs the required packages. *xen* is the Xen OS kernel; *kernel-xen0* is the Xen-enabled host system kernel (domain 0) and *kernel-xenU* is the Xen guest system kernel.
After installing these packages, grub configuration should be changed to support the new installed Xen kernel. The grub.conf should be consist of some lines as follow which defines the Xen and Xen0 kernel path:

```
title Fedora Core (2.6.19-1.2288.fc5xen0)
        root (hd0,0)
        kernel /boot/xen.gz-2.6.19-1.2288.fc5
        module /boot/vmlinuz-2.6.19-1.2288.fc5xen0 ro root=LABEL=/1 rhgb quiet selinux=0
        module /boot/initrd-2.6.19-1.2288.fc5xen0.img
```

Once the system is being rebooted with the xen0 GRUB option, it will start with a Xen-enabled kernel. After that, Xen service should be enabled. Enabling Xen can be checked with following command:

```
# /usr/sbin/xm list
```

Now Xen is enabled and can manage some guest OS concurrently.

4.2.2. Creating Xen filesystem image

To create Xen filesystem image, at first, the required OS should be installed on one partition. But if an empty partition is not available, the best solution is to use QEMU. QEMU is a generic and open source machine emulator and virtualizer. With QEMU it is possible to install the OS on one image file.
To install the required OS with QEMU, at first one image file should be created:

```
# qemu-img create qemu-image.img 1G
```

This command creates an image file with size of 1G. After creating this image file, it is possible to install the OS on it.

```
# qemu -m 128 -hda qemu-image.img -cdrom /dev/cdrom -boot d
```

This command installs the OS from cdrom on the created image file. It considers 128M for RAM.

The next step after installing the OS is converting its format from QEMU to Xen image format. Following shows the steps of conversion:

1. Make a couple directories to mount the images.

```
# mkdir /mnt/loop1; mkdir /mnt/loop2
```

2. Mount the QEMU image to loop1 (in the following command should be calculated).

```
# mount -o loop,offset=##### qemu-image.img /mnt/loop1
```

To calculate ##### at first use *fdisk* command to find out the start sector of root filesystem on image file and then multiple it at 512. For example:

```
# fdisk -lu qemu-image.img
You must set cylinders.
You can do this from the extra functions menu.

Disk qemu-image.img: 0 MB, 0 bytes
255 heads, 63 sectors/track, 0 cylinders, total 0 sectors
Units = sectors of 1 * 512 = 512 bytes

Device Boot          Start      End        Blocks    Id   System
qemu-image.img1      63         257039     128488+   82   Linux swap / Solaris
qemu-image.img2  *   257040     2088449    915705    83   Linux
Partition 2 has different physical/logical beginnings (non-Linux?):
phys=(127, 16, 1) logical=(16, 0, 1)

# mount -o loop,offset=131604480 qemu-image.img /mnt/loop1
```

In this sample offset is equal 512 * 257040 = 131604480, so the mount command should be:

```
# mount -o loop,offset= 131604480 qemu-image.img /mnt/loop1
```

3. Create a new sparse image file and format it (The seek number is how many MB to make the file.)

```
# dd if=/dev/zero of=xen-image.img bs=1M count=1 seek=1024
# mke2fs -F -j xen-image.img
```

4. Mount the spare image.

```
# mount -o loop xen-image.img /mnt/loop2
```

5. Copy everything over from QEME image to Xen image.

```
# cp -ra /mnt/loop1 /mnt/loop2
```

6. Create essential devices on Xen image.

```
# for i in console null zero; \
do \
/sbin/MAKEDEV -d /mnt/loop2/dev -x $i; \
done
```

7. Change Xen image fstab.

```
/dev/sda1       /           ext3        defaults            1 1
none            /dev/pts    devpts      gid=5,mode=620      0 0
none            /dev/shm    tmpfs               defaults            0 0
none            /proc       proc        defaults            0 0
none            /sys        sysfs       defaults            0 0
```

8. Disable TLS on Xen image.

```
# echo "hwcap 0 nosegneg" >> /mnt/bop2/etc/ld.so.conf
```

9. The last step is to create xenU kernel for Xen image. It can be done with yum as said before (yum install kernel-xenU) or can be done from source file. Using source file has advantage of configuring kernel according to the situation. To compile the Xen from source after downloading it, do the following steps:

```
# cd xen-<xen-version>
# make world
# make install
# ./install.sh
# cd linux-<kernel-version>-xenU
# make menuconfig
# make
# make modules_install
# cp vmlinuz /boot/vmlinuz-<kernel-version>-xenU
# cp -ra /lib/modules/<kernel-version> /mnt/loop2/lib/modules/
```

10. Finally, just in case of using Fedora RPM's to install, it is required to add the following line to the /etc/modprobe.conf file in Xen image file.

```
alias eth0 xennet
```

To make changing image kernel easily, it's better to put kernel modules in separate image file and then mount it separately, instead of put it in Xen image file (But this step is optional).

```
# dd if=/dev/zero of=modules bs=1M count=100
# mke2fs -F -j modules
# mount modules /mnt/modules
# cp -ax /lib/modules/<linux xenU version> /mnt/modules
# umount /mnt/modules
```

add this line to Xen image etc/fstab

```
/dev/sda2 /lib/modules ext2 defaults 0 1
```

4.2.3. Configuring and working with Xen

Tycoon-gLite auctioneer manages the guest OS without interference of others, but it would be useful to know how Xen manages its guest OSes.
Xen creates a configuration file for each guest OS under /etc/xen. The sample configuration file for Scientific Linux 3 can be as follow:

```
kernel = "/boot/vmlinuz-<kernel-version>-xenU
memory = 256
extra = "fastboot ro selinux=0"
ramdisk = "/boot/initrd-<kernel-version>-xenU.img"
disk = ['file:/root/xen-image.img,sda1,w', 'file:/root/swap,hda5,w']
hostname = "sl"
name = "sl"
root = "/dev/sda1 ro"
```

In this file, *kernel* refers to Xen enabled kernel of guest OS (xenU). The other important parameter is *disk* which refers to the filesystem image. This image can be a image file or an installed OS on disk. If modules are installed on separate image file, the disk configuration can be like this:

```
disk = ['file:/root/xen-image.img,sda1,w', 'file:/root/modules,sd2,r']
```

If the guest OS was installed on disk the disk parameter should be change as follow:

```
disk = ['phy:/dev/hda5,sda1,w', 'phy:/dev/hda6,sda2,w']
```

On some machines the root filesystem check will bring domain 0 into single user mode. The *fastboot* command line option will make SL skip this check.
The *root* parameter also points to the virtual partition which is mounted as root on machine (it is defined in /etc/fstab in filesystem image).
The only thing to test the guest OS is to boot it. This can be done by executing following command:

```
xm create -c /etc/xen/<your-config-file>
```

4.3. TYCOON-GLITE

In this section the Tycoon-gLite structure and also installing and working with it is explained.

4.3.1. Tycoon-gLite source code

Tycoon consists of four types of components:

- **Clients**: Client programs (also called agents) help users set up and manage accounts, find resources, and manage resources.

  The code which implements this part of tycoon is located under src/Tycoon path of Tycoon-gLite. These codes are implemented by Python script.

- **Auctioneers**: An auctioneer manages the resources on single providing host. Users use their agent to contact auctioneers to query the availability and current prices of resources, bid on resources, etc. To perform most operations on a providing host, a user must first set up a host account on that host. The auctioneer collects bids and manages the virtualization system to ensure that users receive the resources that they bid on.

  Auctioneer codes are in src/Tycoon and src/Tycoon/Virtualization. Auctioneer also is implemented by Python.

- **Service Location Service (SLS)**: The SLS keeps track of which auctioneers are up and what their status is. There is an Internet-wide SLS available at tycoon-sls.hpl.hp.com.

  SLS also is implemented by Python and its code can be found under src/Tycoon.

- **Bank**: The bank keeps track of the amount of currency that different users have in their bank accounts. There is an Internet-wide bank available at tycoon-bank.hpl.hp.com.

  Bank is the only part which implemented by Java and placed under src/bank. It is recommended to use the Internet-wide bank, but for Tycoon-gLite the local bank should be installed.

Installing Tycoon can be done in two different ways: with the RPMs which can be downloaded from the hp lab (http://tycoon.hpl.hp.com/~tycoon/dl/fedora/) or from its source which can be cloned from the repositories.

```
# hg clone http://tycoon.hpl.hp.com/cgi-bin/hgwebdir.cgi/tycoon/KL
# hg clone http://tycoon.hpl.hp.com/cgi-bin/hgwebdir.cgi/tycoon/tycoon
```

To work with source code, two different packages should be cloned: tycoon and KL. tycoon consists of codes for the clients, auctioneer, bank, and service location service, and KL is library code for tycoon.

To test and work with source code, it is required to make a soft link from *KL* in Tycoon source path (suppose *tycoon-glite* is places in ~/tycoon-repo and *KL* is in ~/tycoon-repo):

```
# cd ~/tycoon-repo/tycoon/src
# ln -s ../../KL KL
```

Tycoon has some dependencies which should be installed before working with Tycoon-gLite:

```
# yum -y install \
        mercurial \
        python-twisted.i386 \
        python-crypto \
        atlas \
        glpk-devel \
        pyOpenSSL \
        strace \
        sharutils
# wget -q http://tycoon.hpl.hp.com/\~tycoon/dl/fedora/5/i386/cvxopt-0.7.1-1.i386.rpm
# rpm -ivh cvxopt-0.7.1-1.i386.rpm
```

In above script, cvxopt is downloaded for Fedora core 5. Notice to use correct version for different distributions. Besides these, Java SDK also should be installed.
After installing the packages, some environment variables should be set. *PYTHONPATH* should set to the location of Tycoon-gLite source code and also *JDKDIR* should point to the location where Java SDK is installed.

```
# export PYTHONPATH=~/tycoon-repo/tycoon/src
# export JDKDIR=/usr/java/jdk1.5.0_08
```

The easiest way to copy Tycoon-gLite to filesystem is to create RPMs from source file and install them on filesystem. To do this, after going to the Tycoon-gLite execute the commands:

```
# make rpm
# make sub_rpms
# make kl_rpm
```

The first *make* creates t*ycoon-<version>.noarch.rpm*. The second one creates *tycoon_aucd-<version>.noarch.rpm*, *tycoon_aucd_plnm-<version>.noarch.rpm*, *tycoon_aucd_xen<version>.noarch.rpm* and *tycoon_client-<version>.noarch.rpm*, and the last make *creates KL-<version>.noarch.rpm.*

4.3.2. Installing and Configuring Tycoon Client

The first step on working with Tycoon is installing client. Installing client can be done through its RPM from hp lab:

```
# yum -y -c http://tycoon.hpl.hp.com/~tycoon/dl/yum/tycoon.repo install tycoon_client
```

This command by default installs Tycoon on Fedora Core 4. To install it on other version of Linux, we can use its source or find its prepared RPM files in the following address: http://tycoon.hpl.hp.com/~tycoon/dl/fedora.

The most important steps on working with Tycoon is generating public and private keys and creating an account on bank with generated public key. Public key is used for logging into hosts and performing bank operations. To generate it, use the *ssh-keygen* command:

```
# ssh-keygen -t dsa
# cat .ssh/id_dsa.pub >> .ssh/authorized_keys
# chmod 600 .ssh/authorized_keys
```

ssh-keygen -t dsa creates DSA public and private keys under *.ssh/id_dsa.pub* and *.ssh/id_dsa* files. The public key should be append to *.ssh/authorized_keys* for SSH connections.
The next step is to setup Tycoon configuration for user with its keys.

```
# tycoon user setup amir@kth.se amir .ssh/id_dsa.pub .ssh/id_dsa
```

This command creates a folder with name of *amir@kth.se* under *~/.tycoon* and copies public and private keys on it with name of *bank_account_public_key* and *bank_account_private_key*.
The next step is creating an account on bank. This can be done in two different ways: creating account on public bank or do it on private one. These two are explained on section 4.3.4.
Tycoon uses different ports for different parts of it works. In general three important ports are available which should be opened on firewall for different reasons:

- SLS: Uses TCP port 25955 as outgoing
- Bank: Uses TCP port 8899 as outgoing
- Auctioneer: Uses TCP port 24571 as outgoing

4.3.3. Installing and Configuring Tycoon Auctioneer

An auctioneer manages the resources on single providing host. The auctioneer uses Xen as virtual machine to manages resources for different user. To do this the auctioneer should be installed on domain0 of Xen and then it can manage other domains for each user request.

To install auctioneer, again using yum can be helpful:

```
# yum -y -c http://tycoon.hpl.hp.com/~tycoon/dl/yum/tycoon.repo install tycoon_aucd_xen3
```

This command installs auctioneer and also Xen kernel on Fedora Core 4. To install it on other version of Linux, we can use its source or find its prepared RPM files in the following address: http://tycoon.hpl.hp.com/~tycoon/dl/fedora.

To work with auctioneer, at first the system should be booted with Xen kernel. After installing auctioneer, it creates a script as */etc/rc.d/init.d/tycoon_aucd* which starts auctioneer. Auctioneer listens on TCP port 24571 for requests.

One point which should consider while installing auctioneer with *yum* is that, it installs *tycoon_file_system_xen3_fc4* filesystem image. This is the filesystem image which auctioneer uses for each request. For Tycoon-gLite the filesystem image should be created separately and replaced with this one (section 4.4).

Auctioneer requires some configuration to works well:

1. Changing the firewall setting:

```
# iptables -A INPUT -s \! 127.0.0.1 -p tcp --dport 8001:8002 -j REJECT
# iptables -A INPUT -s \! 127.0.0.1 -p tcp --dport 9601:9699 -j REJECT
# service iptables save
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

2. Copy the owner's bank key to */etc/tycoon* with the following names: *amir@kth.se_bank_account_public_key*and *amir@kth.se_bank_account_private_key*. These keys are used by the auctioneer to make deposits into and withdraws from the owner's bank account.
3. Copy the owner's bank key to */etc/tycoon/admin_public_key*. This key is used to authenticate remote administrative commands to the auctioneer.
4. Add or change the *UserName* option in */etc/tycoon/tycoon_aucd.conf* to set the autioneer's owner.

```
UserName = "amir@kth.se".
```

5. Now it's possible to start its service as follow:

```
/etc/init.d/tycoon_aucd start
```

After installing, configuring auctioneer and creating its filesystem image, the next step is to configure auctioneer to use the new filesystem image. As said before the auctioneer uses *tycoon_file_system_xen3_fc4* as its filesystem image by default. This filesystem and related kernel is placed in */var/lib/tycoon/aucd/Xen3/lib*, and the auctioneer uses a configuration file which its name is the name of user account in Tycoon and it is placed in */var/lib/tycoon/aucd/Xen3/accounts*. For example if there is an account in Tycoon with name of *amir*, we can find *amir.conf* configuration file with the following format:

```
kernel = "/home/amir/vmlinuz-default"
disk   = ['file:/home/amir/default.ext3,sda1,w']
name   = "amir"
root   = "/dev/sda1 ro"
memory = 856
vif    = [ "mac=aa:00:00:77:66:d4, ip=10.106.212.209/30", "mac=aa:00:00:2b:ca:15" ]
ip     = "10.106.212.210"
netmask = "255.255.255.252"
gateway = "10.106.212.209"
hostname = "amir-boogieman.pdc.kth.se"
```

To replace the default kernel and filesystem image it is enough to change the kernel and disk parameter in the configuration file and replace their value to new kernel and filesystem image path.

4.3.4. Installing and Configuring Tycoon Bank

Creating bank account can be done in two ways: using public bank or private one. Creating an account on public bank can be done by registering online from *http://tycoon.hpl.hp.com/wiki/TycoonAccountForm* address. It requires public key and email address of the user to open an account with 100 credit. After opening an account, they inform with a reply message. To check the credit the following command can be used:

```
# tycoon bank get_balance
```

This command should return 100 as the base credit which was created for user.
It is recommended to work with the Internet wide bank at tycoon-bank.hpl.hp.com. But it is also possible to install a private bank. To install we should use the source of Tycoon. The following step then should be done to install private bank:

1. Set the *PYTHONPATH* and *JDKDIR* environment variables:

```
# export PYTHONPATH=~/tycoon-repo/tycoon/src
# export JDKDIR=/usr/java/jdk1.5.0_08
```

2. Original Tycoon save banks data under */var/lib/tycoon/tycoon_bank* but Tycoon-gLite use */root/tycoon/tycoon_bank*. So this folder should be created:

```
mkdir -p /root/tycoon/tycoon_bank
```

3. Then build and start the bank:

```
TZ=UTC make -C ~/tycoon-setup/tycoon/src/bank
```

4. This command compiles the bank code and starts its service. It creates a file *acounts.data* which saves accounts on bank and also it creates two other files *privatekeys.dat* and *publickeys.dat* which consists of bank private and public keys. The bank uses a magic user *__Bank*, that can create accounts. But bank itself does not create *__Bank* folder under *~/.tycoon*, so create *~/.tycoon/__Bank* and copy *privatekeys.dat* and *publickeys.dat* under it and rename them to *bank_account_priavate_key* and *bank_account_public_key*. But notice to remove *comment* and also *__Bank=* form these files. Also copy the *bank_account_public_key* to */etc/tycoon* with new name of *bank_public_key*.
5. The next step is to change the */etc/tycoon/tycoon.conf* and change *BankURLList* and *BankPublicKeyFileName* from their default value to the following:

```
BankPublicKeyFileName = "/etc/tycoon/bank_public_key"
BankURLList = ("http://<your IP address>:8899",)
```

6. The configuration file */etc/tycoon/tycoon.conf* is installed by installing tycoon RPM. Making Tycoon RPM as explained in 4.3.1 is created as follow:

```
# make rpm
# rpm -ivh tycoon-<version>.noarch.rpm
```

7. To create an account on installed bank use the following command:

```
# tycoon -oBankAccountName=__Bank bank create_account amir@kth.se 1000
~/.tycoon/amir\@kth.se/bank_account_public_key
```

This command open an account on private bank with 1000 credit for Amir.

4.3.5. Installing and Configuring Tycoon SLS

As explained before, SLS keeps track of which auctioneers are up and what their status is. The easiest way to install Tycoon SLS is to use its RPM which can be found at:

http://tycoon.hpl.hp.com/~tycoon/dl/fedora/3/i386/tycoon_sls-0.5.0p53-1.noarch.rpm.

After installing this package a script can be found as */etc/rc.d/init.d/tycoon_sls*. This script starts the SLS service with its default port 25955.

SLS was implemented by Python and its code can be found at ~/tycoon-repo/src/Tycoon/ServiceLocationService.py.

There is a global SLS at tycoon-sls.hpl.hp.com which Tycoon by default is configured to use it. After installing the private SLS, the Tycoon configuration file (*/etc/tycoon/tycoon.conf*) should be modified to support it. To do this the following lines should be changed on t*ycoon.conf*:

```
SLSHostName = "<SLS machine IP address>"
SLSPortNumber = "25955"
```

In Tycoon-gLite it is required to install the private SLS for each site.

4.4. CREATING GLITE FILESYSTEM IMAGE

A single filesystem image is used for both CEs and WNs. The image contains a ScientificLinux (SL) filesystem with the gLite CE and WN components pre-installed (but unconfigured). The procedure to install these components are the same as that of ordinary CEs and WNs with the exception that VO specific parameters are just assigned dummy values. These parameters are assigned their real values by the user (i.e. the VO manager) when the virtual cluster is created.

To do this at first install SL on QEMU or real partition, create a filesystem image (using dd command. More detail about how dd can be found in 4.2.2), and mount that image. In the remaining part of this section, it is considered that filesystem image is mounted under /mnt. The steps are as follow:

1. The first step after installing SL is installing gLite basic requirements.

   At first, Java SDK 1.4.2_04 (or higher) should be installed. gLite finds it with variable *JAVA_LOCATION* in configuration file *site-info.def* (this file will be installed by yaim).

   gLite uses *yaim* as a configuration tool. The latest version of it can be found at http://grid-deployment.web.cern.ch/grid-depbyment/gis/yaim. The *yaim* configuration values are saved in a configuration file as key-value pairs. This file is shared among all the different node types (an example of configuration file is the file */opt/glite/yaim/examples/site-info.def).*

2. *Yaim* uses *apt* or *yum* to install other packages of gLite, (which can be selected from site-info.def file). It uses *apt* by default. To use *apt,* the address of repositories should be defined for it through *LCG_REPOSITORY* and *CA_REPOSITORY* in configure files.

```
LCG_REPOSITORY="rpm http://glitesoft.cern.ch/EGEE/gLite/APT/R3.0/ rhel30 externals
Release3.0 updates"
```

With using SL 3 add the following list in /etc/apt/sources.list.d/cern.list file:

```
rpm http://linuxsoft.cern.ch  cern/slc30X/i386/apt  os updates extras
rpm-src http://linuxsoft.cern.ch  cern/slc30X/i386/apt  os updates extras
```

But if using SL 4 the addresses should be change to:

```
rpm http://linuxsoft.cern.ch  cern/slc4X/i386/apt  os updates extras
rpm-src http://linuxsoft.cern.ch  cern/slc4X/i386/apt  os updates extras
```

And change the configuration of /etc/apt/preferences as follow

```
Package: *
Pin: release o=linux.cern.ch
Pin-Priority: 980
```

3. *Yaim* uses the *install_node* script to install the required nodes, which the format of using it, is like:

```
/opt/glite/yaim/scripts/install_node <site-configuration-file> <meta-package> [ <meta-package>
... ]
```

   In Tycoon-gLite the filesystem consists of CE and WN without configuration. So to install them on filesystem image, the yaim *install_node* command can help:

```
# chroot /mnt /opt/glite/yaim/scripts/install_node site-info.def glite-CE TORQUE_server
# chroot /mnt /opt/glite/yaim/scripts/install_node site-info.def WN_torque
```

   Notice that only one of the above commands should be executed on one filesystem image. It means that one filesystem can be created to support CE or support WN.

4. Copy the root public key to filesystem image under */root/.ssh/authorized_keys* to be able to do SSH to filesystem image.

```
# chroot /mnt mkdir -p -m 0700 /root/.ssh
# echo `cat /root/.ssh/id_dsa.pub` > /mnt/root/.ssh/authorized_keys
```

5. The next step is creating *voadmin* user on image filesystem and copying some prepared keys and certificate to it.

```
# chroot /mnt mount /proc
# chroot /mnt useradd voadmin
# chroot /mnt umount /proc
# chroot /mnt mkdir -p -m 0700 /home/voadmin/.ssh
# chroot /mnt chown voadmin /home/voadmin/.ssh
# echo "$SSHKEY" > /mnt/home/voadmin/.ssh/authorized_keys
# chmod 644 /mnt/home/voadmin/.ssh/authorized_keys
# chroot /mnt chown voadmin /home/voadmin/.ssh/authorized_keys
```

6. Because voadmin should have enough permission to configure system, the required permission should be added to it as sudo:

```
# echo "voadmin ALL=(root) NOPASSWD: /root/config_wn_set # voadmin" >> /mnt/etc/sudoers
# echo "voadmin    ALL=(root) NOPASSWD: /root/install_ce      # voadmin" >> /mnt/etc/sudoers
```

7. Then the certificates should be copied to filesystem image. These certificates are used to configure CE or WN.

```
# mkdir -p /mnt/etc/grid-security/
# cp $HOSTCERTDIR/*.pem /mnt/etc/grid-security/
# chmod 400  /mnt/etc/grid-security/hostkey.pem
```

8. If the filesystem is CE filesystem the *install_ce* should be copied to filesystem and if it is WN filesystem, the *install_wn* should be copied. So depending the type of filesystem, one of the following command should be executed:

```
# cp -f ~/root/tycoon-repo/tycoon/packaging/glite/scripts/install_ce /mnt/root/
# cp -f ~/root/tycoon-repo/tycoon/packaging/glite/scripts/install_wn /mnt/root/
```

Now filesystem image is ready to use.

5. FUTURE WORK

The following paragraphs explain weaknesses in the current implementation that should be addressed in future work.

The requirement that all nodes in a virtual cluster should reside on the same LAN is a great limitation. There are several ways to relax this requirement. One is to let the VO managers bid for additional “external” ports in the privileged domain, that are forwarded to the necessary ports of the VM when creating a WN, and then configure the CE to use the external ports and the IP address of the Auctioneer where the WN is installed.

Another solution is to set up a Virtual Private Network (VPN) between the CE and the WNs.

The fact that the VO managers have to manually specify host names, IP addresses and netmasks when creating new CEs, and host names when creating WNs gives rise to the following problems: 1) There is a risk that a new node receives the same name as an already existing node. Only a simple, DNS-based check is made to determine if a name already is in use or not. 2) The system gets unnecessarily complicated since VO managers need to receive information of unused IP addresses. 3) The IP address is actually a resource, but is not treated as such.

One solution to this is to let the VO managers bid on IP addresses and to let the Auctioneers decide host names.

On the other hand, if the Local Resource Management System (LRMS) could be configured to select a queue for a job depending on the VO the submitting user belongs to, it would be enough with one common CE to be used by all VOs. This is, however, to our knowledge not possible.

The requirement that there must be one SLS per LAN containing gLite-prepared Tycoon Auctioneers is a work-around for the problem of finding a set of Auctioneers that all reside on the same LAN. This is clearly not necessary if the “same-LAN-requirement” is relaxed, but it can also be removed adding a netmask field in the SLS information.

gLite uses DNS lookups (both reversed and direct) for host authentication. This makes it difficult to set up a NATed CE. We do not see the necessity of these lookups, and suggest that the possession of the private key corresponding to a host certificate should be sufficient as authentication.

The work-around for ensuring that no malicious code is executed when the configuration file is *sourced*, makes no guarantees for the code not to be executed at a later stage. A shell command like $(<var_name>), where <var_name> is a variable name, would execute the string contained in the value of the variable. We have not been able to find such code in the gLite system, but we think that alternative ways for specifying VO specific parameter values should be investigated.

6. SUMMARY

In this document we have explained how Tycoon will be combined with gLite to provide user-level service level agreements. In the following short paragraphs we repeat the important conclusions of this document.

We have identified the components comprising the system: Each site that wants to host Tycoon enabled gLite Nodes, must have a local DNS, a number of computers, each running a gLite-prepared Tycoon Auctioneer in the privileged domain of a Xen virtualization hypervisor. Each site also needs an SLS for each LAN that contains gLite-prepared Tycoon Auctioneers.

Each VO manager has a VOMS server and a software agent that is used to create and configure the virtual cluster nodes.

No modifications have been made to the gLite software components, but the Tycoon Auctioneer has been modified to use an interchangeable script for VM creation. The system can co-exist with existing gLite infrastructure, and users can choose whether to submit jobs in the *usual* way, or using a specific Tycoon managed virtual cluster.

A virtual cluster belongs to a VO, and is restricted for use by the members of that VO only. The VO manager's Agent does not have information of the priority of individual jobs, which leads to that all jobs executing on a virtual cluster at a given time, will have the same probability to benefit from an incrementation of the spending rate.

Nodes of a virtual cluster all reside on the same LAN. This restriction facilitates the communication between the CE and its WNs, and between different WNs. CEs must have public IP addresses.

7. APPENDIX A – HOW TO UPGRADE TO KERNEL 2.6

One of the problem of using ScientificLinux version 3 (which is the best distribution for gLite) is that it uses the old version of kernel. Because in this project we used Xen 3, we have to upgrade its kernel. Installing new kernel for guest domain on Xen was explained a little in 4.2.2. Here upgrading the kernel is explained in more detail.

At first obtain the current 2.6 kernel. In our case which we use Xen 3, if we download and *make* Xen 3, it, itself downloads the proper version of kernel during making.

Some packages of the filesystem should upgrade to support new kernel. One of the most important packages which should replaced is *module-init-tools*. Kernel 2.4 uses *modutils* to manage its modules while kernel 2.6 uses *module-init-tools*. Its latest version can be downloaded from the following address:

ftp://ftp.kernel.org/pub/linux/kernel/people/rusty/modules

The other package which should be upgraded is filesystem management package. If the filesystem is *ext2* or *ext3* the name of package is *e2fsprogs*. If you are running *reiserfs*, the name of package is *reiserfsprogs*.

8. APPENDIX B - USEFUL SCRIPTS

Attached to this document some useful scripts can be found. In this appendix, some of these scripts are explained:

8.1. CE_AGENT

The format of using ce_agent is as follow:

```
ce_agent create <ce_name> <auctioneer> <budget> <ce_ip> <ce_netmask>
ce_agent delete <ce_name> <auctioneer>
```

Here, *<ce_name>* is the name of the VM to be created, *<auctioneer>* is the IP address to the Auctioneer where the VM is to be created, *<budget>* is the amount of money that is to be transferred from the VO managers bank account to the Auctioneer's ditto. *<ce_ip>* and *<ce_netmask>* are the VMs public IP address and netmask, respectively.

This script contains two main functions: create_ce and delete_ce. The create_ce is used to create virtual machine on auctioneer machine. This function after creating virtual machine on auctioneer machine, connects to it with SSH with voadmin username and then copies the gLite required files (such as user.conf, group.conf, wn-list.conf and bg_vo_info.conf) to the virtual machine. After that it calls install_ce script. This script will be explained later (8.4).

The delete_ce deletes that virtual machine.

8.2. WN_AGENT

The format of using ce_agent is as follow:

```
wn_agent create <wn_name> <auctioneer> <ce_ip> <boudget>
wn_agent delete <wn_name> <auctioneer> <ce_ip>
wn_agent update <wn_name>
```

Here, *<wn_name>* is the name of the VM to be created, *<auctioneer>* is the IP address to the Auctioneer where the VM is to be created, <ce_ip> is the the IP address of CE and *<boudget>* is the amount of money that is to be transferred from the VO managers bank account to the Auctioneer's ditto.

This script has three methods: create_wn, delete_wn and update_ce. The create_wn uses tycoon command and creates an account on auctioneer host. Then the same as create_ce, it copies the gLite required files (8.1). After doing this, it executes the install_wn script (8.5).

Remember to update the CE and let it know about new WN. This can be done with *wn_agent update* which it calls update_ce function. The delete_wn also deletes the WN from auctioneer machine.

8.3. MULTI_WN_AGENT

This script has many functions, but the most importance usage of this script is creating multiple WN simultaneously and then update the CE about the new WNs. The *create* function of this script responsible for doing this. This function first finds the list of SLSs, then uses this list to find the best auctioneers, and then creates WN on them.
The format of using this script to find the list of SLS is as follow:

```
multi_wn_agent list <sls_list> <budget> <deadline> <max_nodes>
```

Here, *<sls_list>* is the list of SLSs, *<budget>* is the amount of money to be spent, *<deadline>* is the amount of seconds over which the budget is to be spent, and *<max_nodes>* is the maximum number of nodes wanted. This outputs a list of Auctioneers for each SLS in *<sls_list>*.
The format of this script to create worker nodes is as follow:

```
multi_wn_agent create <ce_name> <wn_pref> <budget> <deadline> <max_nodes>
```

Here, *<ce_name>* is the name of related CE, *<wn_pref>* is a tag that precedes the name of each WN to be created and *<max_nodes>* is the number of WNs. This script, uses wn_agent (8.2) to create WNs.

8.4. INSTALL_CE

This script is used to configure the CE on virtual machine. This script uses *yaim* to configure the CE. It calls *configure_node* command of yaim.

8.5. INSTALL_WN

Install_wn is used to configure the WN on virtual machine. This script uses *yaim* to configure the WN. It calls *configure_node* command of yaim.

8.6. CE_FS

This script consists of the steps which should be done to create the CE filesystem image. These steps are explained in 4.4.

8.7. WN_FS

This script also consists of the steps to create WN filesystem image.

9. REFERENCES

[1] Kevin Lai, Lars Rasmusson, Eytan Adar, Stephen Sorkin, Li Zhang and Bernardo A. Huberman, **Tycoon: an Implemention of a Distributed Market-Based Resource Allocation System**, Proceedings of the ACM Conference on Electronic Commerce (2004)

[2] Kevin Lai, **Markets are Dead, Long Live Markets**, HP Labs, Palo Alto, CA, USA (2005)

[3] Dragovic, B., Fraser, K., Hand, S., Harris,T., Ho, A., Pratt, I., Warfield, A., Barham, P., and Neugebauer, R. **Xen and the Art of Virtualization**. In Proceedings of the ACM Symposiumon Operating Systems Principles (2003).

[4] **gLite Installation and Configuration Guide v. 3.0 (rev.2)**, http://glite.web.cern.ch/glite/packages/R3.0/R20060502/doc/installation_guide_3.0-2.pdf, May 8 (2006)

[5] BalticGrid, EU FP6, Contract 026715, Annex I - "**Description of Work**", Sep. 15 (2005)