GridFactory

A virtualized compute engine for distributed clusters

Technical Design Report

Frederik Orellana
Niels Bohr Institute
Copenhagen University
January 2008

Contents

1 Introduction

This paper describes the design of GridFactory - a new hierarchical, pull-based batch system with built-in virtualization capabilities. First we will briefly discuss why and how we believe such a system should be created. Chapter 2 details the proposed architecture; chapter 3 describes the interfaces of the various services; chapter 4 describes a first prototype implementation of this architecture; finally, chapter 5 contains a short summary and outlook.

1.1 The case for a batch system with built-in virtualization and software provisioning

Since the CPU of a typical desktop machine spends most of its time idle and desktop machines abound, it is a natural idea to try and exploit this apparently cheap computing power for number-crunching scientific or commercial purposes, for example by using the free CPU cycles of desktop PC's when users are not working on the PCs, e.g. at night. Such "cycle scavenging" is possible with middleware like BOINC and United Devices. Since most of the desktop machines in the world are running Microsoft Windows, the compute jobs run via such middleware are bound to be Windows jobs.

In the scientific community, most computing jobs run on Linux or UNIX clusters via traditional batch systems like PBS, LSF and SUN Grid Engine. These jobs typically involve binaries compiled for the specific architecture and operating system (OS) of a given cluster. This means that a given large computing task is typically locked to a given cluster.

One objective of GridFactory is to eliminate the two kinds of lock-in described above, by means of virtualization and a software provisioning system. The idea is to introduce a hierarchy of (SW) software catalogues and SW repositories and have worker nodes (WN's) get software from their local SW repository. Here, a virtual machine is considered a piece of software like any other. In this way, a selection of OS's is made available on any given resource and thereby, effectively, the amount of resources available to a given job is extended. An obvious example is to run Linux jobs inside Linux virtual machines (VM's) running on Windows PC's.

1.2 The case for a pull back-end for grid systems

Today's production grids are almost exclusively built of Linux clusters[1] with traditional batch systems as back-ends. Such clusters typically consist of dedicated WN's on a protected subnet. Integrating unpredictable off-site resources is not possible in any feasible way. For example, to use a desktop Windows PC as WN would require manually installing a Linux VM on the PC and running a traditional batch daemon inside this VM. However, Since traditional batch systems are push systems, this would require an open firewall between the batch system head-node and the desktop PC – something which is not allowed by most security-aware network administrators and certainly not recommended. Moreover, it would give the compute job access to the local-area network of the desktop PC – not recommended either. If we would none-the-less choose to follow such a path, we would still need a daemon running on the desktop PC, capable of starting, stopping and pausing a virtual machine. In conclusion, we don't consider this a feasible path.

Instead, GridFactory will employ a pull architecture: all network connections between a WN and any servers will be initiated by the WN - specifically by a "job control daemon" running on the WN. This daemon is also responsible for starting and stopping virtual machines.

[1]In the case of the European EGEE grid, all clusters even run exactly the same Linux distribution.

GridFactory will offer both a command-line interface, a Java API and possible other language bindings, and can thus be integrated with grid systems like any other batch system.

We believe that integrating computing resources across organizational and networking boundaries will allow better load balancing and reduced overall maintenance costs for grid systems.

1.3 The case for a stand-alone pull grid system

Even with a pull back-end, using current grid systems for sharing sets of computing resources across organizational boundaries is cumbersome:

- grid middleware stacks typically run only on a limited selection of OS's
- because of the push nature of the systems, firewalls need to be reconfigured (to allow the outside world access to the local grid servers)
- grid middleware is typically non-trivial to install and configure

GridFactory aims to allow the impromptu sharing of a set of computing resources across organizational boundaries. To achieve this, GridFactory will:

- be self-sufficient, i.e. not depend on a shared file system or shared user accounts
- be extremely easy to install, both on the client and server side
- allow a hierarchy of servers that can pull jobs from higher levels – i.e. not depend on any traditional grid middleware

We believe supporting such on-the-fly creation of virtual organizations (VO's) addresses a real need of today's scientific collaborations.

1.4 Design criteria

For the architecture described in this document the following design criteria were employed:

- **Independence**: the new batch system will not depend on a specific grid system. It should offer well documented interfaces that can be used easily by existing grid system as well as by any future systems.
- **Minimal effort**: Whenever possible, standard protocols, servers and clients should be used. In particular, we will not write a new multi-threaded server from scratch, but rather make the job queue available to the WN's via some standard protocol and server (e.g. https/Apache, mysql/MySQL server).
- **Minimal intrusion on workstations**: A pull architecture is mandatory, since we cannot expect workstations to run any services requiring inbound connectivity. Moreover, integrating a workstation as a WN will consist in simply starting up a small application on the WN. This application will be a self-contained binary, a Java program or an applet and will be installed either via a standard installer or simply by copying a single directory to the hard disk.
- **Security:** The host machine will be protected, i.e. jobs will not be allowed any access to the host machine, in particular the network access of the VM will be restricted to inbound access from the host machine.

  Sensitive data will be protected, i.e. not all host machines will be allowed to run all jobs and access all input files. Host machines should be identified by an X.509 certificate, i.e. the process running on the host machine, picking up a job should be authenticated by an X.509 certificate – typically a certificate belonging to the desktop user who started this process. Such a certificate should belong to a VO trusted by the user who originally submitted the job.

2 Architecture

There are three ways of using GridFactory:

1. as a stand-alone batch system
2. as back-end system on a grid site
3. to connect single and groups of computing resources in a hierarchy

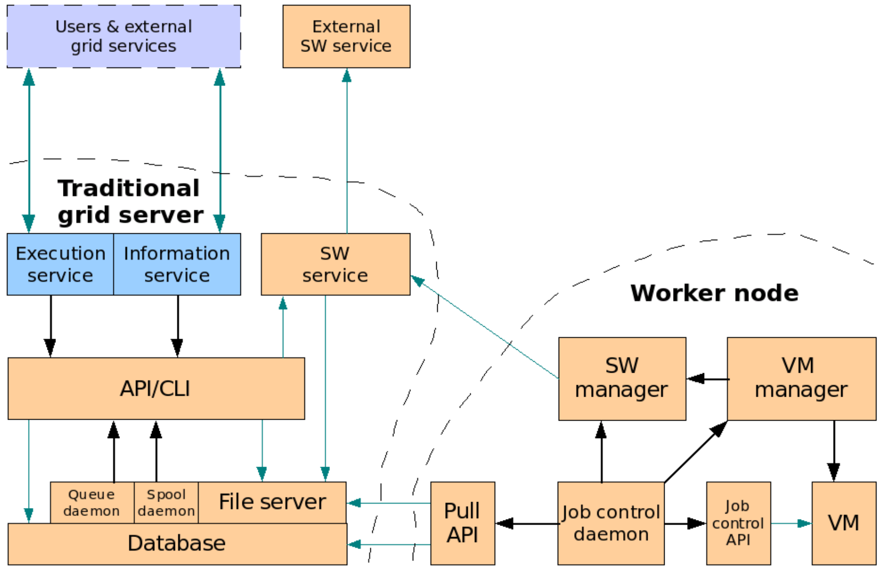

Illustration 1: GridFactory deployed as back-end for a traditional grid system. The thick lines indicate use of an API. The thin lines indicate network communication. The direction of the arrows indicate from the where the network communication is initiated.

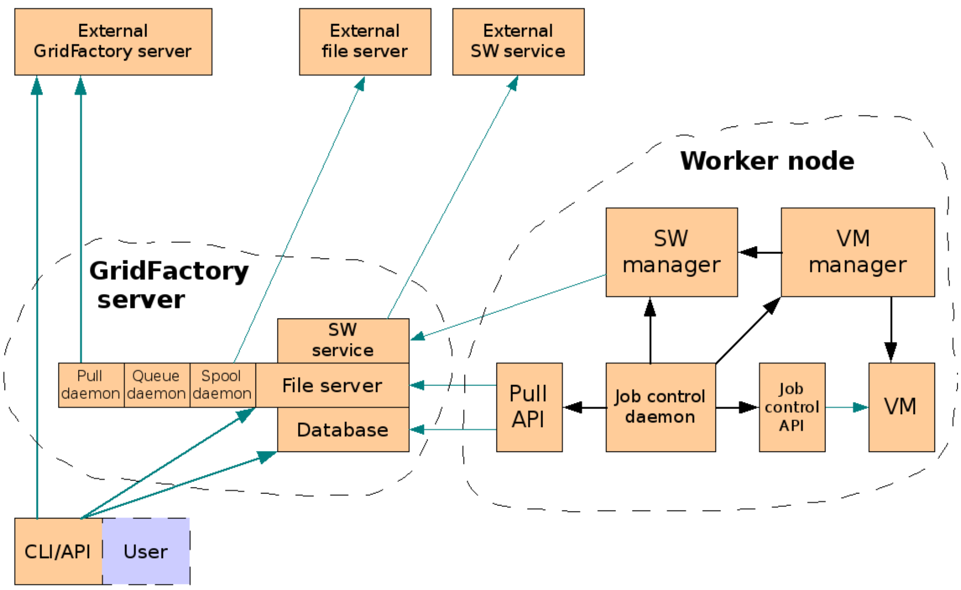

Illustration 2: GridFactory deployed as a stand-alone system. The thick lines indicate use of an API. The thin lines indicate network communication. The direction of the arrows indicate from the where the network communication is initiated

Case 2 is depicted in illustration 1, case 1 and 3 are depicted in illustration 2.

The architecture includes a service on the GridFactory server, making a queue of jobs available to job control daemons running on the WN's, i.e. on the machines that will host a VM. Such a daemon is capable of pulling job descriptions from the GridFactory server and running the corresponding compute jobs inside a VM. To achieve this, the daemon makes use of of a “VM manager” and a “SW manager” that pulls software from a SW service - that can optionally pull SW from an external SW service.

A WN is identified by an X.509 certificate - used by the job control daemon to authenticate against the services it accesses. This certificate can be the grid certificate of the desktop user starting the job control daemon. To make it easy to integrate resources, we consider also implementing a lower-security option, either without X.509 certificates or with certificates issued on the fly.

The communication of the job control daemon with the front-end is mediated by the “pull API” library. In a first implementation, this library resides on the WN. Later, the stippled line on illustration 1 can be moved one step to the right, i.e. a web service utilizing the “pull API” library put in front of the file server + database and the job control daemon.

The components of GridFactory will now be discussed in turn.

2.1 Job database

The database serves the following purposes:

- allows the queue daemon to announce jobs ready for execution
- allows WN's to search for jobs matching their system: OS, memory, installed SW packages, VO membership etc.
- allows a WN to announce its willingness to run a given job
- allows the queue daemon to permit a given WN to run a given job
- allows a WN to pick up all information necessary to run the job
- allow a WN to pick up job-control information (kill, upload stdout, ...)
- allows a WN to update status information for the job

Thus, there must be well-defined interfaces for users, the batch system and the WN daemon to access the database (see chapter 3).

2.2 Software catalogue

This service allows the spool daemon to decide whether or not fail a job depending on whether or not the requested SW is available. Assuming the SW is available, it also allows the WN's to provision it. As already mentioned, a virtual machine providing an OS is in this connection considered a piece of software like any other.

2.3 File server

This service allows WN's to download input files[2] from - and upload output files back to the front-end. The file transfers are authorized and authenticated via the certificate of the WN daemon, i.e. the file server provides access control based on X.509 certificates and VO membership. The WN daemon picks up the concrete URLs in the job database. There must be well-defined interfaces for the queue daemon to make a given file available for download and for the WN daemon to download and upload files from/to a given URL – see chapter 3 for these.

---

[2]These files are already downloaded to the front-end by a grid service or the spool daemon – depending on the mode of deployment of GridFactory – see chapter 1.

2.4 Spool daemon

The spool daemon regularly scans a spool area on disk and takes actions when certain situations arise:

Situation	Action taken
A job description file plus input files are placed in a new directory in the spool directory.	Any input files specified in the job description are downloaded. A job record is created in the database.
In the spool directory, a job directory is found without a corresponding entry in the job database	The job directory is deleted.

Table 1: Tasks of the spool daemon.

2.5 Queue daemon

The queue daemon regularly scans the database and takes actions when certain situations arise:

Situation	Action taken
A job is tagged as requested by a WN daemon.	If authorization of the WN goes well, the input files are made readable and the job is tagged as prepared. Otherwise, the job status set back to ready.
A job is tagged as running.	The WN is allowed write access to an upload location (for output files).
A job is tagged as executed.	If the output files have been uploaded the job is tagged as done. Otherwise it is tagged as failed.
A running job does not have its time stamp (a field in the database) updated for more than a certain amount of minutes.	The job is considered lost (the desktop user could have killed it, the machine could have crashed, ...) and the status set to ready. If this happens repeatedly, the status is set to failed.

Table 2: Tasks of the queue daemon.

The authentication and authorization (to run a given job) of a WN depends on:

- whether or not the WN certificate is is issued by a certificate authority trusted by the front-end
- VO membership: a job can be tagged as belonging to a given VO. Only WNs with certificates belonging to this VO will be allowed to download the input files of such a job

2.6 Pull daemon

The pull daemon pulls jobs from a remote GridFactory system – appearing to this remote system like a job control daemon, and runs them on the local GridFactory system. It will stop pulling when a configurable number of queued jobs is reached.

2.7 Job control daemon

On each WN a daemon will run which regularly scans the job database for eligible jobs, selects a job, starts a VM and runs the job *inside* this machine. Since the WN may be a desktop workstation, the daemon must be able to monitor if this workstation is in use, and if so, stop pulling jobs and suspend/pause running jobs. Once the workstation has been idle for a given time, jobs should be resumed.

The daemon must thus be capable of:

- communicating with the database
- starting the VM
- stopping the VM
- executing commands within the VM
- communicating with a relaying Janitor service
- keeping track of cached or installed SW packages and input files

In order to provide this functionality the daemon will make use of two software components:

- a **virtual machine manager**, capable of starting, stopping and configuring a VM, using an OS disk image and running jobs inside this VM
- a **software manager**, capable of getting and caching SW packages from the SW service

The daemon and its associated software components will run on various flavours of Windows and Linux and a GUI will be provided to start, stop and configure the daemon. The daemon will also give some graphical feedback on:

- when jobs are being picked up and exiting
- the CPU and memory consumption of the VM
- if possible: the nature and progress of running jobs[3]

The daemon will regularly scan the database and take actions when certain situations arise:

[3]This would require an API to be provided for jobs to inform about their status – *and* job submitters to use this API. This can hardly be expected of the casual user, but production managers running large scale productions should get some motivation from the goodwill and larger uptake such feedback is likely to produce.

Situation	Action taken
A job in state ready is found that can run on the VM operating system of the WN.	The public key/certificate is written in the job record. The job is tagged as requested.
A job with my public key is tagged as assigned.	Input files are downloaded and the job is started. If this goes well, the job is tagged as running.
The WN daemon fails repeatedly in getting a requested software package.	The job is set back as ready – and it is remembered not to try running it again.
A running job no longer has any record in the database.	The job is killed.
A job finishes.	Output files are attempted uploaded and the job is tagged as executed.
A desktop user starts working.	The job is suspended and tagged as paused.
A desktop user stops working.	The job is resumed and tagged as running.
A job exceeds its time limit.	The job is killed, a message is added to its stderr, the stdout and stderr are uploaded and the job is tagged as failed.

Table 3: Description of service of the job control daemon.

3 Interfaces

The interfaces described in this section are meant to provide a general understanding of the system. Only the methods important for this are listed. For full details, the javadoc of the sources should be consulted.

3.1 Batch system API

Class LRMS

Method Summary

JobInfo[] getJobs(String dbHost, String[] ids, boolean onlyRunning)	Get job records from a given database host.
int getJobStatus(String host, String id)	Get job status from a job database.
String[] getOutput(String host, String jobId)	Get the stdout/stderr of a running job.
void kill(String dbHost, String[] ids)	Request a given database host to have jobs killed.
void killRunningJobs(String host, String dn)	Kill all running jobs submitted with a certificate of a given distinguished name.
String submit(String id, String executableScript, String[] values, String submitUrl, boolean scriptOverValues, String certificate)	Submits a job.

3.2 *Batch system command-line interface*

3.2.1 psub

NAME

psub – submit a job

SYNOPSIS

psub **[options] [job_script]**

DESCRIPTION

psub is used to submit a job.

OPTIONS

-h

print help

-d level

print debug information – levels 0-3, 0 is off

-n name

use name to label job

-i f1 f2 ...

copy files from URLs f1, f1, ... to the run directory on the worker node

-e ex1 ex2 ...

make files ex1, ex2, ... executable

-t secs

request computing time equivalent to running at most secs seconds on a 2.8 GHz Intel Pentium 4 processor

-m mem

request number of megabytes mem of RAM

-o f1 f2 ...

specify that the job will produce output files with names f1, f2 in the run directory

-r rte1 rte2 ...

require runtime environments rte1, rte2, ...

-c cert

for remote submission: use X.509 certificate cert to authenticate

-k pkey

for remote submission: use private key pkey to authenticate

-p keypass

for remote submission: use password keypass to decrypt the private key

-a cadir

for remote submission: trust certificate authorities corresponding to certificates in the directory cadir

-s dn

when X.509 authentication is not used: use dn to identify yourself

-f pass

for remote submission: if no x509 credentials are given, use pass to authenticate with the remote server

-v vo1 vo2 ...

allow only virtual organizations vo1, vo2, ... to run job

-z

require that the job should run in its own virtual machine

-b host

submit the job to remote host

-j

enable bulk submission mode - multiple script files can be specified, options given in the files take precedence over command-line options

3.2.2 pstat

NAME

pstat – get status of jobs

SYNOPSIS

pstat [options] [job_ids]

DESCRIPTION

pstat is used to query the status of submitted jobs.

OPTIONS

-h

print help

-d level

print debug information – levels 0-3, 0 is off

-s dn

consider only jobs belonging to user with certificate subject dn

-f pass

for remote submission: if no x509 credentials are given, use pass to authenticate with the remote server

-v vo

consider only jobs allowed only to virtual organization vo

-q

consider only running jobs

-b host

query all jobs on the given host - if not given, the host name is taken from the identifier

-c cert

for remote jobs: use X.509 certificate cert to authenticate

-k pkey

for remote jobs: use private key pkey to authenticate

-p keypass

for remote jobs: use password keypass to decrypt the private key

-a cadir

for remote jobs: trust certificate authorities corresponding to certificates in the directory cadir

3.2.3 pcat

NAME

pcat – print standard out and standard error of a running job

SYNOPSIS

pcat [options] [job_id]

DESCRIPTION

pcat is used to view the standard out and/or standard error of a running job.

OPTIONS

-h

print help

-d level

print debug information – levels 0-3, 0 is off

-o

print stdout

-e

print stderr

-c cert

for remote jobs: use X.509 certificate cert to authenticate

-k pkey

for remote jobs: use private key pkey to authenticate

-p keypass

for remote jobs: use password keypass to decrypt the private key

-a cadir

for remote jobs: trust certificate authorities corresponding to certificates in the directory cadir

-s dn

consider only jobs belonging to user with certificate subject dn

-f pass

for remote submission: if no x509 credentials are given, use pass to authenticate with the remote server

-b host

query a given host - if not given, the host

name is taken from the identifier

3.2.4 pkill

NAME

pkill – kill a job

SYNOPSIS

pkill **[options] [job_ids]**

DESCRIPTION

pkill is used to kill running jobs.

OPTIONS

-h

print help

-d level

print debug information – levels 0-3, 0 is off

-b host

kill all jobs on a given host belonging to the specified

DN or the DN of the specified certificate

-s dn

consider only jobs belonging to user with certificate

subject dn

-f pass

for remote submission: if no x509 credentials are given,

use pass to authenticate with the remote server

-c cert

for remote jobs: use X.509 certificate cert to authenticate

-k pkey

for remote jobs: use private key pkey to authenticate

-p keypass

for remote jobs: use password keypass to decrypt the private key

-a cadir

for remote jobs: trust certificate authorities corresponding to certificates in the directory cadir

3.3 Pull API

Class DBMgr

Method Summary

JobInfo[] findJobs(String[] identifiers)	Scans the job database for all jobs.
JobInfo[] findEligibleJobs()	Scans the job database for runnable jobs.
JobInfo[] findRunningJobs()	Scans the job database for running or paused jobs.

void requestJob(JobInfo job, String providerInfo)	Sets the status of the job to 'requested' and writes the certificate subject in the 'providerInfo' field of the jobDefinition on the remote database.
void unRequestJob(JobInfo job)	Sets the status of the job to 'ready:n' and clears the certificate subject in the 'providerInfo' field of the jobDefinition on the remote database.
JobInfo getJobInfoFromDB(String id)	Get a job definition record from the job database.
void killJob(String id)	Kill a given job.
void killRunningJobs(String dn)	Kill all running jobs submitted with a certificate of a given distinguished name.
void commit(JobInfo job)	Saves the current state of a job to the job database.
void commit(List jobs)	Runs commit on each of the members of a List of JobInfo objects.
void update(JobInfo job)	Updates the current state of a job from the job database.

3.4 Software manager API

Class RTEMgr

Method Summary

HashMap checkAvailability(JobInfo job, TransferControl transferControl)	Check whether or not some runtime environments are available and downloadable by the credentials of the job.
String findBaseSystem(String os)	Find name of 'VirtualMachine' MetaPackage providing a given operating system.
Vector getRteDepends(jVector rtes, String _os)	Find all runtime environments on which some runtime environments depend.
String getRteURL(String rteName, String os)	Find the URL to download a given runtime environment that runs on a given operating system.
Boolean isOperatingSystemSupported(String os, boolean virtualize)	Check whether or not this operating system can be provided.

3.5 Virtual machine manager API

Class VMMgr

Method Summary

VirtualMachine launchVM(String rteName, int memory)	Boot a virtual machine of a given name.
void pauseInstance(String host)	Pause virtual machine.
void resumeInstance(String host)	Resume paused virtual machine.

void terminateInstance(String host)	Halt virtual machine and terminate the virtual machine engine.
void terminateAllInstances()	Halt all running virtual machines and terminate virtual machine engines.

3.6 Job control API

Class JobControl

Method Summary

void submitJobs(Vector jobs)	Submit a Vector of JobInfo objects.
void killJobs(Vector killJobs)	Kill a set of jobs.
void cleanup(JobInfo job)	Cleanup locally after job.
boolean uploadStdoutErr(JobInfo job)	Copy tmp stdout -> final stdout, tmp stderr -> final stderr.
void postProcess(JobInfo job)	Operations performed when a job has finished: copy output files to final destination and clean up.
String[] getCurrentOutput(JobInfo job)	Get current outputs of this job.
String getFullStatus(JobInfo job)	Get the full status of a job.
String getJobInformation(JobInfo job)	Find information about a job.
String[] getScripts(JobInfo job)	Get the job scripts.
void updateStatus(Vector jobs)	Check the status of the jobs (with the ShellMgr) and updates status for each job accordingly.
void exit()	Cleanup any temporary files etc.

Class TransferControl

boolean copyInputFile(String src, String dest, Shell shellMgr)	Copy input files from the local hard disk (file://..) or a remote location to the run directory of the job, using the ShellMgr of the job.
boolean copyOutputFile(String src, String dest, Shell shellMgr)	Copy output files from the run directory of the job to a local or remote destination, using the ShellMgr of the job.
boolean copyToFinalDest(JobInfo job, Shell shellMgr, String runDir)	Move output files, stdout and stderr of the job to their final destination.

Class RTEInstaller

public RTEInstaller(String _url, String _remoteDir, String _localDir, String _rteName, Shell _shellMgr)	Constructor.
public void install()	Download and install a software package.

3.7 Job description schema

Below (table 4) follows the schema of the job description table in the job database. The field 'status' is allowed only a predefined set of values, given in table 5. Illustration 3 gives a simplified view of the transitions between these states.

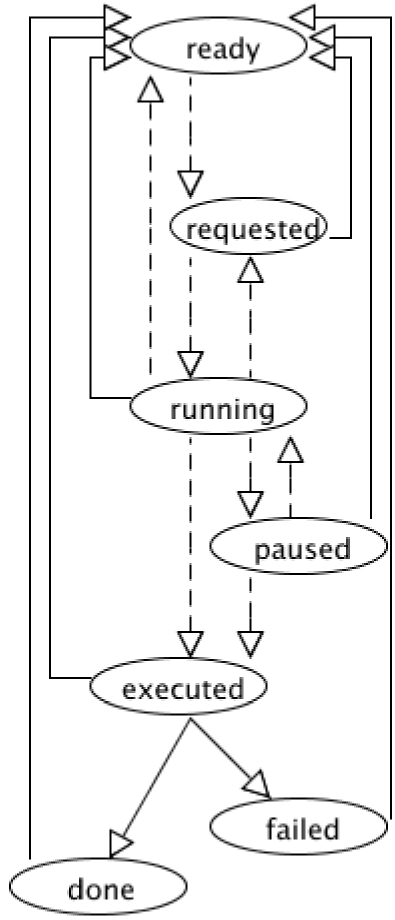

Illustration 3: ***The life of a job – states in the job database.*** *Dashed lines denote a change performed by the WN, solid lines a changed performed by the front-end.*

Field	Description
identifier	unique identifier of the job definition
name	descriptive name of the job
csStatus	the status of the job. For possible values, see below
inputFileURLs	space-separated list of input files that will be downloaded to the WN. The file names are local to the working directory on the WN
executable	the executable to run
arguments	arguments for the executable
executables	space separated list of *file names* from the “inputFileURLs” set, which will be given executable permission
gridTime	maximal CPU time request for the job scaled to the 2.8 GHz Intel Pentium 4 processor
memory	required memory in megabytes
virtualize	whether or not to require this job to run in its own VM
outFileMapping	space-separated list of output file name/destination URL dublets. The file names are local to the working directory on the WN
opSys	required operating system string. Can be prepended with '>=' to indicate that a certain version or higher is required or '<=' to indicate that a certain version or lower is required
rtes	space-separated list of required software packages. Each package can be prepended with '>=' to indicate that a certain version or higher is required or '<=' to indicate that a certain version or lower is required
userInfo	the DN of the grid certificate of the person who submitted the job
providerInfo	the DN of the grid certificate of the WN daemon which picked up the job
allowedVOs	space-separated list of VOs allowed to run this job
stdoutDest	the final destination of the stdout of the job
stderrDest	the final destination of the stderr of the job
jobId	identifier of the job
outTmp	temporary location of the stdout of the job
errTmp	temporary location of the stderr of the job
metaData	optional lines of metadata in the form key: value. The keys could be e.g. “cpuSeconds”, “created”, “lastmodified”, “validationResult”

Table 4: The job description database schema. The gray cells are for internal use and not to be filled out by the submitter.

Value	Description
defined	initial state of a job
ready	set by the server when all input files have been downloaded and are ready for download
requested	set by WN to request a job. The WN must also write its public key / certificate in the field 'public key'
prepared	set by the back-end when the input files are ready to be downloaded by the WN daemon
prepared:downloading	set by the WN when starting to download the input files
submitted	set by WN after picking up all job data and input files and submitting the job to the VMMS
running	set by WN when the job has started running inside the VM
running:requestPause	set by the server to have the WN pause the job
running:requestOutput	set by the back-end to have the WN upload the stdout/stderr of thejob
running:requestKill	set by the back-end to have the WN kill the job
paused	set by WN after suspending a job
aborted	set by server if the submitter killed the job
executed	set by WN after a job has finished
uploaded	set by WN after output files have been uploaded to server
done	set by server after verifying that an executed job has uploaded its output files
failed	set by server if a job is not updating its time stamp, output files have not been uploaded – set by WN if a job fails or exceeds its time limit

Table 5: Allowed values of the field 'status' in the job database.

4 Prototype implementation details

As database, MySQL was chosen because of the straight-forward set-up and the support for two-way X.509 authentication. As file server, Apache+mod_gridsite was chosen. Again because of the straight-forward set-up and because it support GACL.

The 4 daemons, as well as the command line-tools of the prototype are all implemented in Java. There are strong reasons for using Java:

- both MySQL and gsiftp are well supported
- a Java program will run on all desktop workstations
- implementing a GUI in Java is relatively straight forward

4.1 MySQL database

The database server runs on the front-end and contains the job description table with the schema describe in chapter 3.7. Authorization is done via standard MySQL database and table permissions.

In the first implementation all worker nodes with a certificate in an authorized VO are allowed to read and change all records as well as create new ones. In a later implementation, with a web service in front of the database, a registration and approval system for the WN's should be put in place.

4.2 File server

Input files for the jobs as well as the files making up a SW package are served over https (e.g. via Apache+mod_gridsite (http://www.gridsite.org/)). The spool manager downloads requested files or checks that the files have been downloaded by a grid server and makes them available in a directory with GACL access control, where only the WN in question is allowed to read. Output files are received by having the back-end provide a GACL protected directory, where only the WN in question is allowed to upload.

4.3 Java API

The shared library, providing the API's of chapter 3 make use of the library provided by MySQL for communicating with the MySQL server and the COQ-kit library for communicating with the file server.

4.4 Spool daemon

This daemon scans the spool directory regularly and takes the actions described in table 1. This directory is served by the file server - which honours the permissions set via GACL (in .gacl files).

The daemon is implemented in Java, making use of the API's of chapter 3.

4.5 Queue daemon

This daemon scans the job database regularly and takes the actions described in table 2.

The daemon is implemented in Java, making use of the API's of chapter 3.

4.6 Pull daemon

This daemon pulls jobs from a remote GridFactory system.

The daemon is implemented in Java, making use of the API's of chapter 3.

4.7 Job control daemon

This daemon scans the database regularly and take the actions described in table 3.

It keeps a list and a cache of installed SW packages. It will attempt getting any required packages from the SW service running on the GridFactory server. One of these can be a VM (e.g. Fedora-7 on VirtualBox).

The VM image will automatically start an SSH daemon and allow login with a known user name and password. The daemon executes shell commands inside the VM over SSH via the JSCH library. It keeps a list of running and attempted-run jobs. If a job fails in getting any requested SW package, it will be set back to 'ready' in the database, but not attempted run again.

The daemon is implemented in Java, making use of the API's of chapter 3.

4.8 Grid computing integration

As mentioned in chapter 1, GridFactory can be used like any other batch system as a back-end for grid systems. This, typically requires writing some shell or Perl scripts that make use of the command-line tools to run jobs. For performance reasons, it may be preferable to bypass the command-line tools and directly create directories in the spool area. Currently, GridFactory has been integrated with ARC.

4.9 Grid information system integration

If GridFactory is used as a grid back-end there needs to be a way to feed the grid information system with information on total number of CPU's, number of busy CPU's. etc. The queue manager daemon keeps this information in a separate table in the database. The API's provide the means for getting the information. The total number CPUs is estimated by the number of WN's that have recently been looking for jobs to pick up. The number of busy CPU's is found by querying the job database itself.

4.10 Virtualization engine

Currently, a CentOS 5, VirtualBox image and Scientific Linux 4, QEMU (with and without kernel acceleration module) image have been created and tested. Each image is bundled with some command-line tools and provided through a SW catalogue as a SW package. The CentOS 5 package depends on another software package, namely VirtualBox, and on a given OS. VirtualBox cannot, in general, be provisioned by the job control daemon, since the installation requires administrator privileges. The Scientific Linux 4 package with QEMU without kernel acceleration, only has an OS dependency – QEMU is bundled with the package. For QEMI with kernel acceleration, the situation is the same as for VirtualBox. In fact, for both VirtualBox and QEMU, a package has been created for both Windows and Linux.

5 Summary and outlook

The overall vision of GridFactory is to make it completely hassle-free to share computing resources across organisational boundaries, i.e. creating and destroying virtual organisations on the fly.

The prototype described in the present paper represents a big first step in this direction. The outlined strategy followed was to use standard open-source products as far as possible and keep the amount of coding to a minimum.

In order to make this prototype a production-quality system, work still needs to be done in the areas of security, testing, benchmarking, scalability and standards compliance. In particular, a web service layer should be put in place for the database access by the worker nodes.