Software provisioning and virtualisation for grid services

Frederik Orellana
Niels Bohr Institute
Copenhagen University
October 2009

Contents

1 Introduction

This note describes an architecture for provisioning software on worker nodes in a batch or grid system without the involvement of a shared file system. The heart of this architecture is a software catalogue in XML format which contains the necessary information for a worker node to download and install a given software package from a server, which may in turn have downloaded the package from another server etc.

The full architecture has been concretely implemented in a hybrid grid/batch/cloud system called GridFactory, which is described in detail in another publication [GRIDFACTORY]. A (simpler) variant of the architecture has previously been implemented by another group, in a system called the Janitor [JANITOR]. This group also developed the original RDF format of the catalogue. In their system, the information from the catalogue is used to install a given software package on a grid front-end machine, which then is assumed to serve the software to worker nodes via a shared file system.

2 The software catalogue

The RDF software catalogue consists of elements of the kind "MetaPackage", "TarPackage", "ImagePackage" and "BaseSystem".

A MetaPackage is an abstract software package that can have multiple concrete realisations or *instances*. These instances are concrete software package that run on a specific platform and can be downloaded and installed. As an example, consider a software package called ATLAS-13.0.40. Its MetaPackage is represented in RDF by:

```
<kb:MetaPackage rdf:about="#1"  kb:name="APPS/HEP/ATLAS/ATLAS-13.0.40">
   <kb:instance rdf:resource="#2"/>
   <kb:provides>APPS/HEP/ROOT</kb:provides>
   <kb:provides>APPS/HEP/GEANT</kb:provides>
</kb:MetaPackage>
```

Larger software packages like ATLAS, providing more than one stand-alone piece of software indicate this via kb:provides, like e.g. the ATLAS-13.0.40 package which also provides ROOT and GEANT.

One way of realising a concrete instance of e.g. ATLAS-13.0.40 is as a TarPackage. A TarPackage is a tarball containing the software necessary to run ATLAS on a given platform, e.g. Ubuntu-7.04 (see chapter 3).

```
<kb:TarPackage rdf:about="#2">
   <kb:url>https://my.server/rtes/atlas-13.0.40.tar.gz</kb:url>
   <kb:basesystem rdf:resource="#3"/>
   <kb:depends rdf:resource="#4"/>
</kb:TarPackage>
```

A platform is described by a BaseSystem catalogue entry, like e.g.

```
<kb:BaseSystem rdf:about="#3" kb:name="Ubuntu-7.0.4">
  <kb:provides>Linux.</kb:provides>
  <kb:provides>Debian.</kb:provides>
  <kb:description>Ubuntu Linux 7.0.4</kb:description>
</kb:BaseSystem>
```

Here the kb:provides element allows the system to match Ubuntu-7.0.4 with jobs requesting just a Debian or Linux operating system[1].

The relation MetaPackage -> TarPackage -> BaseSystem is depicted in illustration 1.

---

[1]The identifiers #1 and #3 are there to be compatible with the Janitor system. With the GridFactory system, instances and BaseSystems can be referred to by their name - APPS/GRAPH/POVRAY-3.6 and Ubuntu-7.0.4 in the case at hand.

Apart from requiring a BaseSystem, an instance package may also depend on other software packages. This is expressed via the kb:depends element. In this case, the "#4" could for instance represent Java-1.5 which would then have a MetaPackage entry like:

```
<kb:MetaPackage rdf:about="#4"  kb:name="ENV/Java/JRE-1.5.0">
   <kb:instance rdf:resource="#5"/>
</kb:MetaPackage>
```

Additional instance formats are: ImagePackage and DebianPackage. The former is supported by GridFactory and not by Janitor and vice versa for the latter[2].

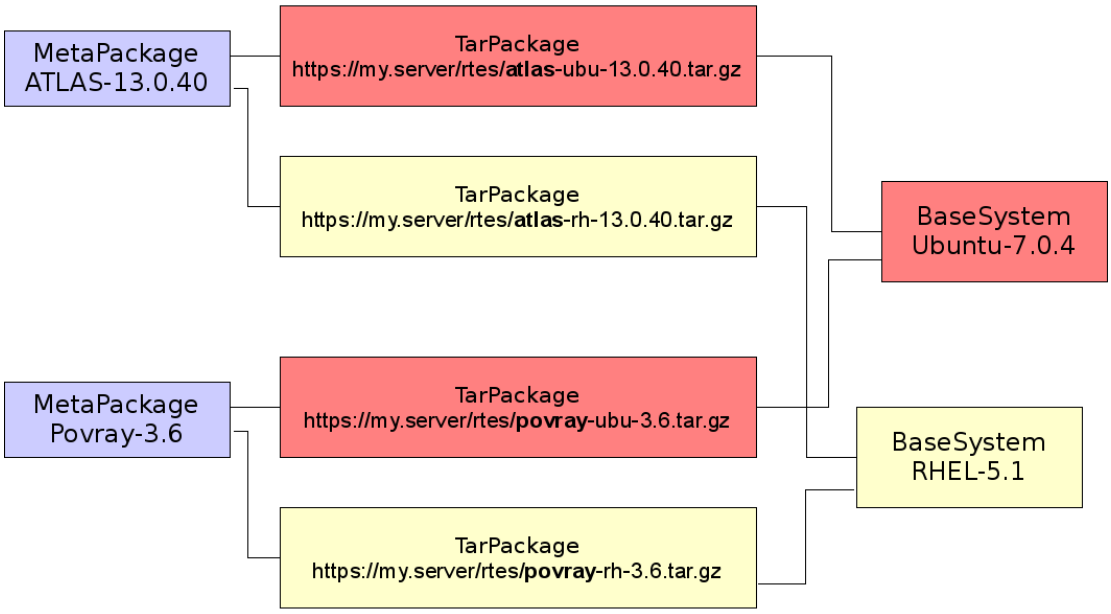

Illustration 1: Structure of the software catalogue.

3 TarPackages

The unpacked tarball of a TarPackage is required to have a certain directory structure and 3 mandatory scripts:

```
control/
        install
        runtime
        remove
data/
```

In case the TarPackage has a Windows platform as BaseSystem, the scripts in the control directory, should be named install.bat, runtime.bat and remove.bat instead. The convention is that once the tarball is unpacked, the install script should be run once and do any other general set-up needed. When a user wishes to use the software, she should then source the runtime script which should cause the commands provided by the software to be made available to the her (put on her "path"). The remove script should do any other clean-up needed apart from deleting the unpacked tarball.

[2]We don't see the reason for introducing special abstractions for installers in the format of Debian, RedHat, Windows MSI, etc. Installing e.g. a Debian package can just as well be done via a TarPackage with a dependency on APT-GET and containing an install script that simply calls apt-get. The APT-GET MetaPackage can simply be realised as an auto-discovered packaged that is only present on Debian systems (see chapter 6).

4 ImagePackages

For an ImagePackage, the URL providing the actual software must point to a disk image. By a disk image is understood a file that can be mounted via the loop-back device. Once mounted, the directory should have the same structure as that of an unpacked tar package.

One motivation for introducing this format is that some software packages are very large and contain many thousand files. Unpacking a tarball providing such a package can take longer than downloading the tarball. In contrast to this, mounting a disk image is almost instantaneous. Another motivation is that some virtual machine engines, allow specifying a disk image at boot time – causing the image to appear as a block device inside the virtual machine and eliminating the need to copy the software into the virtual machine by other means (which can be very expensive).

A representation of the ATLAS-13.0.40 software as an ImagePackage reads:

```
<kb:ImagePackage rdf:about="#2">
   <kb:url>https://my.server/rtes/atlas-13.0.40.img</kb:url>
   <kb:basesystem rdf:resource="#3"/>
   <kb:depends rdf:resource="#4"/>
   <kb:mountpoint>/tmp/atlas.quiqt14116</kb:mountpoint>
</kb:ImagePackage>
```

Notice the optional kb:mountpoint element, which specifies where in the file system to mount the image. If this is not specified, it is assumed that the disk can be mounted anywhere.

5 Virtual machines

A special kind of MetaPackage is a virtual machine MetaPackage. In the catalogue, such a MetaPackage is characterised by having a kb:VirtualMachine element. The corresponding instance packages must provide the means to boot up a virtual machine and manage jobs inside it. This will be discussed in detail in chapter 8. To consider a specific example, the catalogue entries for a virtual machine running Fedora-7 could read:

```
<kb:MetaPackage rdf:about="#5" kb:description="Virtual machine running Fedora 7" kb:name="VM/Linux/Fedora-7">
  <kb:provides>Linux.</kb:provides>
  <kb:provides>Fedora.</kb:provides>
  <kb:instance rdf:resource="#6"/>
  <kb:VirtualMachine>
    <kb:os>Fedora-7</kb:os>
    <kb:out_port>80</kb:out_port>
    <kb:out_port>443</kb:out_port>
    <kb:service>ssh</kb:service>
    <kb:privilege>administrator</kb:privilege>
  </kb:VirtualMachine>
</kb:MetaPackage>

<kb: TarPackage rdf:about="#6"  kb:url="https://my.server/rtes/VM/Fedora-7.img.gz">
  <kb:basesystem rdf:resource="#3"/>
</kb:TarPackage>
```

When a GridFactory worker node encounters a job requiring a MetaPackage, which in turn has only only instance packages on BaseSystems that do not match the system of the worker node, the catalogue is searched for a virtual machine that provides the requested BaseSystem. If a match is found, a relevant virtual machine is booted and the job is run inside the virtual machine.

Notice the element indicating that a given virtual machine automatically starts an SSH daemon:

```
<kb:service>ssh</kb:service>
```

Notice also the element element indicating the minimum memory (in kilobytes) this VM needs:

```
<kb:Requirements>
  <kb:mem_kb>500000</kb:mem_kb>
</kb:Requirements>
```

6 Auto-discovery of installed software

The TarPackage of the previous chapter has no dependencies. This means that. it provides a self contained VM engine + disk image that will run a Fedora-7 virtual machine on Ubuntu-7.0.4. To our knowledge, in user space, this can be realized only with QEMU (and not with very good performance).

Alternatively, the package could depend on e.g. VirtualBox. Software like VirtualBox cannot be installed in user space, since it uses special kernel modules. Thus, it cannot be installed by GridFactory – unless the daemon on the worker node is running as root (not recommended for security reasons).

Describing a software package like VirtualBox in the catalogue is done via a special kind of MetaPackages, namely those that contain a kb:RegistryKey and/or a kb:command element.

The MetaPackage element for VirtualBox reads:

```
<kb:MetaPackage rdf:about="#7" kb:description="VirtualBox virtual machine engine" kb:name="VM/VirtualBox">
  <kb:registryKey>HKLM\SOFTWARE\innotek\VirtualBox:installdir</kb:registrykey>
  <kb:command> VBoxManage </kb:command>
</kb:MetaPackage>
```

The idea is to allow checking directly if the software corresponding to the MetaPackage is installed by checking if a given command is available (and returns 0) or querying the Windows registry. RegistryKey should be of the form path:key. path should be a registry path and key a registry key to query. If the key is present, it is assumed that the given software package is installed.

7 Catalogue format summary

In the previous chapters, various elements of the software catalogue have been described. Here follows a listing of all valid elements:

- **MetaPackage**
  - **rdf:about**
  - **kb:description**
  - **kb:instance**
  - **kb:registrykey**
  - **kb:command**
  - **kb:provides**
  - **kb:depends**
  - **kb:lastupdate**
  - **kb:tag**
  - **rdfs:label**
  - **kb:VirtualMachine**
    - **kb:os**
    - **kb: out_port**
    - **kb: in_port**
    - **kb:service**
    - **kb:privilege**
  - **kb:Requirements**
    - **kb:mem_kb**
    - **kb:out_port**
    - **kb:in_port**
    - **kb:privilege**
- **TarPackage**
  - **rdf:about**

        - **kb:url**
        - **kb:basesystem**
        - **kb:depends**
        - **kb:lastupdate**
        - rdfs:label

- **ImagePackage**

        - **rdf:about**
        - **kb:url**
        - **kb:basesystem**
        - **kb:depends**
        - **kb:lastupdate**
        - **kb:mountpoint**
        - rdfs:label

- **BaseSystem**

        - **rdf:about**
        - **kb:description**
        - **kb:lastupdate**
        - rdfs:label
        - kb:url

The grayed-out elements are not used by GridFactory and their use is discouraged.

8 Virtual machines and job management

A virtual machine software package must provide the means to:

- boot and halt the virtual machine
- set up other software packages inside the virtual machine
- run jobs inside the virtual machine

The last points require transferring files in and out of the virtual machine.

Two different kinds of virtual machines are distinguished, providing different ways of managing jobs:

1. The virtual machine is required to run an SSH daemon and provide information on how to log in. Then all can be done via SSH.
2. A command-line tool provided by the virtual machine software package is simply invoked with the right arguments (executables, input/output files etc.).

Notice that in the last case, it is entirely up to the author of the virtual machine software in question to decide how the command-line tool is implemented. One possibility could certainly be to use SSH. To get files in to and out of the virtual machine, other methods may be more optimal; e.g. for VirtualBox, so-called shared folders can be used.

In both cases, a software package can provide a virtual machine by depending on other MetaPackages or by providing a stand-alone virtualisation system plus disk image. Concretely, the software of such a package must provide either the commands

- vm_boot,
- vm_mboot,
- vm_kill,
- vm_pause,
- vm_resume,

or the command

- vm_job_run.

The two cases will now be described in more detail.

8.1 Job management via SSH

An software package published as a virtual machine running an SSH server must provide a command:

vm_boot [*memory*].

This command should boot a virtual machine, with the specified memory (in megabytes) and return a string of the form

user_name:*password*@*host_name*:*ssh_port*,

allowing an SSH connection to the virtual machine from the host machine. Optionally, one or several of the commands ma be provided additionally:

vm_kill *vm_id*,

vm_pause *vm_id*,

vm_resume *vm_id*,

where vm_id is a string of the form given above. If the vm_kill command is not provided, it is assumed that the the login user is allowed to issue the command 'halt' and that the virtual machine software exits completely when the virtual machine is halted. If the two last commands are not provided, it is simply not possible to pause and resume the virtual machine.

8.2 Job management with a command-line tool

A software package published in a catalogue as a virtual machine, but not providing the command vm_boot, must provide a command:

vm_job_run [-i *input_file_1* -i *input_file_2* ...] [-o *output_file_1* -o *output_file_2* ...] [-m *memory*] *job_executable*.

input_file_1, ... are names of input files already downloaded to the current directory on the host machine and to be used by the job running on the guest machine. output_file_, ... are names of output files that will be produced by the job script in the directory where it is started. vm_job_run should copy these from the virtual machine to the current directory on the host machine. The command should be blocking and have the stdout and stderr and return code of the job script, once the job has finished.

9 Provisioning the software

Given the conventions for content of TarPackages and ImagePackages, the catalogue provides enough information for e.g. a user or a daemon to provision a given software package.

The actual download and installation of the software is the responsibility of the user or daemon that needs it. In the case of GridFactory, on every worker node there is indeed a daemon running that pulls job information from a server and provisions the software required by the job. In the case of the Janitor, this provisioning is done on the grid server by a grid service triggering the Janitor Perl script.

The actual installation consists in:

1. unpacking a tarball or mounting a disk image
2. modifying the scripts in the directory “data”
3. changing directory to the directory “data”
4. executing the script “../control/install”
5. renaming the directory “data” to “pkg”

The modification in step 2, should involve:

- setting the string “%BASEDIR%” to the full path of the directory “pkg”
- setting the string “%CACHEDIR%” to the full path of the directory where downloaded tarballs and image files are kept

10 File transfers

As mentioned, in GridFactory, it is the responsibility of the daemon on the worker node to download the software from a GridFactory server. This server, however, may be subscribing to software catalogues on other servers and may have downloaded the software from one of these - and so on.

In principle, it is the choice of each server (administrator) who is allowed to read its catalogue and who is allowed to download each software package. This is unavoidable, as each server administrator has supreme control of her own machine. Therefore, a typical set-up will consist in creating a virtual organisation and have a number of servers all join this. Each server will then typically allow this read access to the catalogue and to the software packages needed by this particular organisation. If a server downloads a software package from another location and makes it available on a new location, the default behaviour of GridFactory is to assign the same permissions found on the old location to this new location.

Technically, all authorised transfers are done via HTTPS and the authorisation is done via the grid access control language [GACL] (GACL).

Clearly, in the case of large software packages and many worker nodes downloading from the same server, downloading the software may be a serious bottleneck. With GridFactory this can be alleviated, since each worker node (or server) is allowed to subscribe to multiple catalogues. Scaling up the performance can then be achieved by simply adding more servers, serving the same catalogue and software packages. Another way of scaling up is to use a hierarchy of servers and assign groups of worker nodes to different servers in the lowest level of the hierarchy. It it is also under consideration to use Bittorrent [BITTORRENT] for downloading the packages.

11 Bibliography

GRIDFACTORY, Frederik Orellana, “GridFactory – a compute system for distributed clusters”, *in preparation*

JANITOR, Daniel Bayer, Tashfeen Bhimdi, Georg Oechsler, Frederik Orellana, Anders Wäänänen, Balázs Kónya, and Steffen Möller: “Dynamic Runtime Environments for Grid Computing”, Proceedings of the Cracow Grid Workshop '07, https://www.knowarc.eu/wiki/images/3/3a/Dre.pdf

GACL, Andrew McNab, “Grid-based access control for Unix environments, Filesystems and Web Sites”, Talk from the 2003 Computing in High Energy and Nuclear Physics (CHEP03), La Jolla, Ca, USA, March 2003, 3 pages, http://arxiv.org/abs/cs.DC/0306030

BITTORRENT, Bram Cohen, “The BitTorrent Protocol Specification” (January 10, 2008), http://bittorrent.org/beps/bep_0003.html