WHALE, a management tool for Tier-2 LCG sites

L M Barone\textsuperscript{1,2}, G Organtini\textsuperscript{1,2}, I G Talamo\textsuperscript{2}

\textsuperscript{1} Department of Physics, University of Rome “Sapienza”
\textsuperscript{2} INFN - Sezione di Roma

Abstract. The LCG (Worldwide LHC Computing Grid) is a grid-based hierarchical computing distributed facility, composed of more than 140 computing centers, organized in 4 tiers, by size and offer of services. Every site, although independent for many technical choices, has to provide services with a well-defined set of interfaces. For this reason, different LCG sites need frequently to manage very similar situations, like jobs behaviour on the batch system, dataset transfers between sites, operating system and experiment software installation and configuration, monitoring of services.

In this context we created WHALE (WHALE Handles Administration in an LCG Environment), a software actually used at the T2\textsubscript{IT} Rome site, an LCG Tier-2 for the CMS experiment.

WHALE is a generic, site independent tool written in Python: it allows administrator to interact in a uniform and coherent way with several subsystems using a high level syntax which hides specific commands.

The architecture of WHALE is based on the plugin concept and on the possibility of connecting the output of a plugin to the input of the next one, in a pipe-like system, giving the administrator the possibility of making complex functions by combining the simpler ones. The core of WHALE just handles the plugin orchestrations, while even the basic functions (eg. the WHALE activity logging) are performed by plugins, giving the capability to tune and possibly modify every component of the system. WHALE already provides many plugins useful for a LCG site and some more for a Tier-2 of the CMS experiment, especially in the field of job management, dataset transfer and analysis of performance results and availability tests (eg. Nagios tests, SAM tests). Thanks to its architecture and the provided plugins WHALE makes easy to perform tasks that, even if logically simple, are technically complex or tedious, like eg. closing all the worker nodes with a job-failure rate greater than a given threshold. Finally, thanks to the centralization of the activities on a single point and to its logging functionalities, WHALES acts as a knowledge-base of the site and a handful tool to keep track of the activities at a given site. For this reason it also provides a tailored plugin to perform advanced searches in the activity log.

1. Introduction
The LHC experiments are now in a stable phase of data taking, having started to record collisions at the end of 2009. As it is well known, the experiments use a worldwide distributed computing infrastructure, hierarchically organized in so called Tiers. This infrastructure is based on the GRID paradigm in its different flavours, mainly LCG and OSG. The Tier0 is unique and based at CERN: it is mainly devoted to first data reconstruction and archival. There exist about 10 Tier1s worldwide (among which, 7 serve the CMS VO) with data custodial responsibility, performing successive activities: computation of physical quantities from detector signals (reconstruction), selection of reduced data samples (skimming) and sometimes Monte
Carlo simulations. The Tier2s are medium size structures, typically with order of one to several thousands computing cores and of 500 to 1000 TBs for data storage.

The whole machinery is now smoothly operational, being up and running since years, and many related activities can be considered mature enough. However the daily operations in a Tier2 are still a burden. In order to keep a Tier2 in good shape, pass the infinite tests which certify reliability and availability, and respond to all the VO requests, one needs to follow tens of components at several basic levels: operating systems, GRID middleware, storage management, batch system, experiment software.

To be more specific without being comprehensive, the operating systems (OS) level requires continuous update of OS versions on worker nodes (WN), mainly for security reasons, as well as on the hardware hosting the GRID services ie. computing element (CE), storage element (SE), user interface (UI), etc.

The GRID middleware level requires following new releases, often with very bad timing compared to the experiment time schedule, updating certificates, sometimes switching to completely new services (like from the LCG CE to the CREAM CE).

The storage management is a critical component. The systems in use at Tier2s are not many, mainly: dCache[1], DPM[2], StoRM[3], Hadoop[4]. Again following the latest version and ensuring the interoperability among Grids is a painstaking activity, so is the batch system. Usually the batch system is a quite stable component, but it often requires reconfiguration or human intervention to cope with hardware changes and user behaviour and requests.

Finally the experiment software level requires following the development of all the software services needed to make a Tier2 a valid element of a given experiment computing infrastructure; as an example for the CMS experiment, one can consider PhEDEx[5][6], the data transfer service, SQUID/Frontier for database caching, the CMSSW software installation, and more.

On top of all this a Tier2 structure faces the usual hardware problems: failures, upgrades, maintenance and all that.

Each one of the above listed tasks is in itself simple, if not trivial, but the overall combination usually in the hands of a single person may become a maintenance nightmare. For this reason every Tier2 in time has developed a set of scripts to ease the daily activity. The CMS Rome Tier2 (codename T2 IT Rome) made a step forward; having so many and diverse activities to manage and monitor, in an evolving and never stable environment, we decided to create a meta-tool, completely configurable and expandable, by the use of plugins.

This project, called WHALE (recursively WHALE Handles Administration in an LCG Environment), has been designed as the ultimate Tier2 administrator tool, and can be actually employed in other environments as well.

WHALE is both a command-line tool and a Python 2.6 API. The website of the project is http://lcg-whale.googlecode.com where installation instructions and sample code are provided.

2. WHALE Architecture

The architecture of WHALE is composed of two parts, the Plugin System and the Orchestration System, that collaborate to make it highly expandable and customizable (through the Plugin System) and powerful and intuitive (through to the Orchestration System).

The Plug System is a hierarchy of Python classes, organized to reflect the different features of an LCG Tier-2 site. With the current release of WHALE the following categories have been addressed:

- the batch system;
- the storage system;
- the monitoring;
• CMS specific tools.

A tree showing the class hierarchy is shown in figure 1.

The Orchestration System is in charge of connecting the plugins together to provide a simple and coherent interface to the system. Plugins can be added (addPlugin) and removed (removePlugin) dynamically to an instance of the Orchestrator class (whale.plugins.Meta).

It works on the notions of Converters, Generators and Path. A plugin method that accepts as input a value of a certain type \(X\) and returns zero or more occurrences of type \(Y\), is called a \(X\)\(-\)Converter or \(C(X,Y)\) and its name has to be \(X2Y\). An example of a converter is a method that takes as input a host name and returns the list of Nagios services defined on it. Such a method has to be named Host2NagiosService.

A Conversion Path \(P(X,Y)\) is a list of Converters, say \(C_1(X_1,Y_1), \ldots, C_n(X_n,Y_n)\) with \(X_1 = X\) and \(Y_n = Y\), so that \(Y_i = X_{i+1}\) for \(i = 1, n - 1\).

The Orchestration System provides a method (convert) that takes as input two types \((X, Y)\) and a list of values or a single value \((x)\) of type \(X\), finds, if existing, a Conversion Path \(P(X,Y)\), applies all the list of Converters, passing the elements of \(x\) to the first one, and returns the result of the last conversion, hiding all the intermediate steps.

To find a Conversion Path the Orchestration System builds a graph of the plugins, where the nodes are the available types (say Host and NagiosService from the previous examples) and the edges are the available converters. A Conversion Path is a path on the graph, discovered with the Dijkstra algorithm for the shortest path[7], using the python-graph library[8].

The Orchestrator can provide a graphic representation of the Converters graphs in the form of a .dot graph[9] with the method getDotGraph; this representation can be shown with Graphviz or other typical graph drawing software (see figure 2).

We have to point out that in some situations conflicts can arise when trying to find a Conversion Path. This may happen when two plugins provide the same Converters (ie. Converters for the same types) or when we have two instances of the same plugin. This cases
are quite common, for example we have the GridMapDir plugin and the DCache plugin, both providing a User2Dn method, but working in different context, the first in the Computing Element mapping, the latter in the dCache gPlazma authorization mapping[12]. Or we may have two different instances of the Nagios plugin to consult two different Nagios servers. In these cases the Orchestration System interacts with the user, showing the conflict and the available Converters, and asking to choose one.

A plugin method that gives a list of occurrences of type X is called a X-Generator or G(X) and has to be named _2X. An example of a generator is a method that returns the list of all the hosts defined on a batch system. This method will be named _2_host.

The Orchestration System provides a method (generate) that takes as parameter the type X and returns the result of _2X if a plugin exists providing such a method. As with Converters, if more Generators for the same type are found, the user intervention is required.

Furthermore, even if the convert method hides the intermediate steps, it can be useful in some cases to have access to them. Every time a Conversion Path is performed, a variable called lastConversion keeps all the intermediate steps, in the form of a list of Python dictionaries,
with key/value association as results of each conversion.

In figure 3 one can see an example of a Conversion Path, from Dataset to Pool, with the intermediate steps explicitly shown.

3. Applications
In this section we discuss some sample usage of WHALE in the context of typical administration activities of a Tier-2 site. We want to stress that the examples shown may describe specific use cases: however a converter is a general purpose operator, which, like a function in a programming language, can be applied in any appropriate workflow.

The code shown below can be run interactively from a Python 2.6 console, after having installed WHALE and after having imported WHALE with the command:

```python
from tools.lcgwhale import *
```

The commands can also be run with the command-line tool (`lcgwhale.py`) passing as argument to the option `-r`.

### 3.1. Job Management
In the area of Job Management WHALE provides the `BatchSystem` plugin, with empty methods that have to be implemented in the subclasses for different batch system, and the `LSF` subclass, for the Platform LSF batch system[10]. In the Grid context WHALE also provides the `GridMapDir` plugin for getting informations on local and grid users running on the site.

Let us show few examples of real applications to our Tier-2. If we want to know the user’s certificate DN of a specific job (job 965915 in the example):

```python
>>> whale.convert("JobId","DN",965915)
```
[‘/dc=ch/dc=cern/ou=organic units/ou=users/cn=ceballos/cn=488892/cn=gullelmo gomez ceballos retuerto:cms’]

In this case the sequence of Converters used is:

```python
>>> whale.printConverter(whale.findConverter("JobId","DN"))
LSF.JobId2User->GridMapDir.User2DN
```

Making this operation by hand would have required different steps:

- finding the local user associated to the job;
- getting the inode number of the file named like the user in the gridmapdir directory;
- getting the file of the DN with the same inode number.

We can even combine converters with a generator to get the list of all the DN who currently have jobs at our site:

```python
>>> whale.convert("JobId","DN",whale.generate("JobId"))
['/c=it/o=infn/ou=personal certificate/l=roma 1/cn=michael sigamani:cms',
 '/dc=ch/dc=cern/ou=computers/cn=cmspilotjob/vocms157.cern.ch:prdcms',
 '/c=de/o=germangrid/ou=kit/cn=christian boeser:cms',
]
```

We can also perform actions on the jobs, for example we can kill the jobs of a specific DN:

```python
>>> whale.convert("DN","JobKill","/dc=it/ou=user/sn=malicious user")
More than one converter found. Choose one between the following:
1) GridMapDir.DN2User
2) DCache.DN2User
```

In this case, the Orchestrator has to convert a DN to a User but there are the GridMapDir plugin and the DCache plugin providing that converter, so WHALE asks the user to choose one of them. After selecting the first one the system continues autonomously:

```
Job <911269> is being terminated
Job <911275> is being terminated
```

Furthermore, we may want to close in LSF the worker nodes that have the disk Nagios service in critical status. In our Tier-2, when this service is in critical status it means that the main disk on the node is almost full, so it is better to close it, to avoid execution of further jobs before appropriate checks.

```python
>>> whale.convert("Host","NodeClose",
whale.generate("Host",
{"host":"cmsrm-wn",
 "service":"disk",
 "inStatus":"CRITICAL"}))
Close <cmsrm-wn097> ...... done
Close <cmsrm-wn073> ...... done
```

3.2. Storage System

In the area of Storage System WHALE provides support for dCache, a distributed storage system broadly used in the LCG community. The plugin package for storage systems is `whale.plugins.storagesystem` and inside it there is the `dcache` module with the `DCache` plugin class.

By combining a Generator provided by the plugin, together with its converters we can get a list of all the pools in our dCache installation, with the ratio of their free space:
A common operation on a dCache installation is checking if a file is an orphan. An orphan file in the dCache context means a file that appears in the file namespace but is not present on any storage pool. This can happen especially in case of hardware failures on storage pools. The dCache plugin provides a method to check if a file is an orphan by providing its PnfsId, an alpha-numeric string unique in all the dCache filesystem. To make the check (the filename has been shortened for sake of clarity):

```python
dcache.isOrphan(whale.convert("Pfn","Pnfsid","/pnfs/roma1.infn.it/da..."))
False
```

By combining this method with a converter and a generator we can get the list of all the orphan files in our current dCache installation:

```python
>>> orphans=filter(dcache.isOrphan,whale.generate("Pnfsid"))
>>> len(orphans)
16
```

And then get their filenames:

```python
>>> whale.convert("Pnfsid","Pfn",orphans)
[/pnfs/roma1.infn.it/data/cms/store/user/spadhi/CCRC08/cmsrm-se01.roma1.i...
```

### 3.3. PhEDEx

The CMS experiment uses PhEDEx for high-performance dataset transfers between sites, verification of links quality and more. It is a complex system and provides a very complete HTTP/JSon API, that we integrated into WHALE.

Continuing from the dCache example, we may want to know the list of orphan Dataset, that is a dataset containing orphan files, to eventually retransfer them to our site:

```python
>>> orphansDatasets=whale.convert(Pnfsid,Dataset,orphans)
```

WHALE can also be used to cross-check informations, for example we may check that the files belonging to a certain dataset (the real dataset name has been shortened for sake of clarity) have all the correct checksum in our dCache instance.

```python
dataset="/GluGluToHToGG_M-120_7TeV-powheg-pythia6/Winter10-7TeV_ProbDist_2011"
retransfer=[]
for l in whale.convert("Dataset","Lfn",dataset):
    phedexchecksum=whale.convert("Lfn","Adler",l)[0].zfill(8)
dcachechecksum=whale.convert("Pnfsid", "Adler",
    whale.convert("Lfn","Pnfsid",l,"protocol":"direct"))[0][0]
if phedexchecksum!=dcachechecksum:
    retransfer.append(l)
```

Note that we had to specify `zfill(8)` on `phedexchecksum` to fill with leading zeros the resulting string. This is necessary for comparing with the dCache ones, since this is what dCache returns.

Now we look for a site (different from T2_IT_Rome) that has that dataset.
>>> sites=whale.convert("Dataset","PhedexNode",dataset)
>>> sites.remove("T2_IT_Rome")
>>> site=sites[0]
>>> site
u'T2_FR_CCIN2P3'

And we can retransfer the files to our site, by using the SRM plugin:

```python
for lfn in retransfer:
    source=whale.convert("Lfn","Pfn",lfn,
            {"protocol":"srmv2","PhedexNode":site})[0]
    destination=whale.convert("Lfn",
            "Pfn",
            lfn,{"PhedexNode":"T2_IT_Rome",
            "protocol":"srmv2"})[0]
    print source
    print destination
    srm.srmCp(source,destination)
```

4. Conclusions

WHALE is a tool developed and in use at the CMS T2_IT_Rome site, as can be seen by the example plugins provided. Anyway most of the plugins are usable in LCG sites with the same technology even for different experiments and many are usable by other CMS sites even with different technologies.

A great improvement of the tool would come if other sites would integrate it in their working environment, expanding WHALE with new plugins to support their technologies, their experiments or any feature that could help administrators in their work. This would also help the community of LCG site administrators to share their experience, problems and solution, helping all the infrastructure to achieve better site performance and resource usage.

5. References

[1] Fuhrmann P for the dCache team http://www.dcache.org/manuals/dcache-whitepaper-light.pdf
[2] https://svnweb.cern.ch/trac/lcgdm/wiki/Dpm
[3] Donno F, Ghiselli A, Magnoni L, Zappi R 2004 StoRM: disk management middleware, CHEP ’04
[4] http://http://hadoop.apache.org/
[5] Tuura L et al. 2006 PhEDEx high-throughput data transfer management system CHEP ’06
[6] Wildish T 2012 From toolkit to framework - the past and future evolution of PhEDEx CHEP ’12
[7] Dijkstra E W 1959 A note on two problems in connexion with graphs Numerische Mathematik 1
[8] http://code.google.com/p/python-graph/
[9] http://www.graphviz.org
[10] http://www.platform.com/workload-management/high-performance-computing
[11] http://www.nagios.org
[12] http://www.dcache.org/manuals/Book-1.9.12/config/cf-gplazma.shtml