

Auto-administration of gLite-based grid sites

Alexandru Stanciu¹, Bogdan Enciu¹, Gabriel Neagu¹,

¹ National Institute for Research and Development in Informatics – ICI Bucharest,
Romania
{alex, enciu, gneagu}@ici.ro

Abstract. Glite middleware is intensively used as a grid middleware deployed in production infrastructures such as the one operated by the EGI-inSPIRE project. In addition of the complexity of the grid infrastructure, gLite has its own particularities which can increase the effort required to properly administrate a grid site. It is very desirable to automate the configuration and administration of gLite-based grid sites, and one solution could be the usage of the Puppet application. By creating a configuration policy for a gLite-based grid site it is also possible to have self-recovery from errors.

Keywords: automatic system administration, self-repairing, gLite grid site

1 Introduction

A general problem of large distributed ICT infrastructures, like the Grid infrastructure, is their extremely high complexity which makes their management especially difficult and error prone [1]. Automated configuration of the systems is designed to solve the problems which occurs with systems that are administrated manually [2]. It is desirable to use a configuration management system such as Puppet, if there are defined exhaustive rules to achieve a specified system state which can be defined as a configuration "policy" always to be enforced whenever there is a deviation from it. Following we present a solution for the gLite grid middleware configuration and Grid site administration using Puppet for automation and error recovery. The rest of the paper is organized as follows: we present Puppet as a solution for automatic systems configuration, and the next chapter propose a solution for automatic configuration of gLite grid middleware using this application.

2 Using Puppet for automatic configuration of systems

Puppet is an open-source configuration management system designed for declarative management of the configuration of "Unix-like" systems [3]. Puppet has a declarative language for describing systems configuration using the paradigm of "client-server".

The level of resource abstraction configuration allows administrators to describe high-level terms, such as “users”, services” and “packages”. At the base level, Puppet uses "resources" to manage systems. These resources can be packages, files, “templates”, directories, services, and one can define her/his own resources. "Modules" combine collections of classes, definitions and resources. These are collections of portable configurations.

Conceptually, the system configuration is modelled as a collection of resources and relations between them. When shaping a configuration problem, one should carefully plan how to “split” the problem in small pieces containing resources and relationships. The decomposition of the configuration of resources and relationships is essential for effective use of the configuration management system. When the modelling of the problem in these terms is successful, Puppet proves to be very useful, otherwise it can become a "burden".

The Puppet language describes resources, collections and relationships that shape a system configuration. The fact that Puppet is a declarative language means that one should not indicate in a Puppet's manifest file what and how to do, but what is desired to be achieved, the final result, and it is the configuration agent's task to carry out the actions needed to achieve the final result.

Puppet accomplishes the configuration changes in a convergent way, i.e. each change operation has the character of a “fixed point”. Instead of describing the necessary steps to make a change, it is described the desired final state of the system. After that, the Puppet agent assures the execution of the necessary configuration steps. The agent can run many times, regardless the initial state of the system, and the result could be predictable.

In addition to Puppet there are other solutions for the the automatic systems management and there is an ongoing debate regarded configuration management tools. There are applications like Cfengine, Chef, Quattor which can provide the same functionality as Puppet, however we found some advantages that Puppet has over others like Chef and have determined us to choose Puppet: it has a larger installed base and a larger developer base and it also has longer commercial track record. Puppet has explicit dependency management and it is convergent, where Chef is imperative.

3 Automatic configuration of Grid middleware according to a policy

There are three methods for a system installation and configuration: imperative, generative and convergent[4]. These methods form a hierarchy in which imperative methods are closest to the machine and convergent methods closest to becoming self-healing. During initial machine bootstrapping, imperative methods are inescapable (the cost of replacing them with generative or convergent methods is too high).

After initial configuration, convergent methods (used by Puppet tool) have the potential to inadequately cover all required options, a behavior not exhibited by

typical generative systems. The initial run of a convergent system must “configure all variables” and thus looks like a generative one. An ideal configuration management tool actually has attributes of each of the strategies:

- i. an imperative bootstrap;
- ii. a generative initial configuration;
- iii. a convergent ongoing management process;

For the latter, convergent should be considered in the strong sense rather than the weak sense exhibited in most convergent tools, including Cfengine.

We want to further develop the specification of the "policy" that must be maintained by each system which is part of a Grid site belonging to the overall Grid infrastructure. In this context, the "policy" means the system state, namely how it is configured and how it ensures operating systems parameters such as service start, maintaining consistent access rights, etc. Therefore when an error occurs, it can be detected and resolved automatically by the configuration system because the system state becomes inconsistent with the "policy" defined for it, and so must be corrected.

We argue that a system which is capable to automatically administrate itself and to perform self-repair actions in the case of error occurring can be implemented if the system's configuration is specified according to the rules defined as a “configuration policy” which must exhaustively describe the system correct state.

The wrong behavior of the system, which causes applications errors, can be viewed as a deviation from the correct state which is enforced by the configuration management system – Puppet. The issue is fixed by system reconfiguration to the desired state.

The algorithm used to employ the auto-administration and self-recovery of the systems in case of an error contains the following steps:

1. perform system configuration according to a policy;
2. monitor the system state in order to detect any change of the desired system state which could represent the occurrence of an error;
3. automatic re-configuration which brings the system back to its default state specified in the configuration policy.

Steps 2 and 3 are repeated successively in order to detect/revert any error state that can occur during the system normal operation.

By specifying all the parameters that define the configuration of a system, which we call the configuration "policy" for that system it can be assured that if an error occurs in the operation of the system this will be notified to the configuration management system that constantly checks the system state. When it notices that the current state is different than the specified state, it will automatically re-configure the system to the desired state, which means that the system should get back to normal operation.

Puppet thus must act to restore the system to the properly configured state, and must meet the "policy" that has been defined for the node. This action implicitly corrected the error detected.

For example, in the case of gLite middleware, a very common problem is when the Workload Management Service (WMS) fails to receive the final status of the job execution when two different methods have been tried:

1. The final state of the application run-time was recorded in a file called "Maradona" and it was tried to copy the respective file to WMS node using globus-url-copy command;
2. The job's wrapper has redirected its final status to stdout, and then it should have been delivered to WMS using specific Globus' commands.

When both methods have failed usually means that the application did not run until the end. This can happen either when the application is not started at all, or it was stopped before the finish. Among the many causes that may lead to application non-execution there could be a problem with the Local Resource Management System (LRMS). But another reason for the failure to receive the final status of application could be due to the fact that it had stopped or it was completed before reaching the normal end.

Listing all the possible causes of this error, we can say that the solutions proposed to resolve the error can be implemented automatically if this requires that a declared system state should be monitored and constantly maintained - this is Puppet's responsibility - and if the solutions proposed to resolve the causes of the error can be defined as some system states that could create a "correct policy" to administer the system.

For example, in order to ensure proper configuration for the "/home" directory we may include this snippet in a Puppet manifest file:

```
mount { [ "/home":
          atboot => true,
          ensure => mounted,
          device => "se.rncgrid.ici.ro",
          fstype => glusterfs,
          name => "/home",
          options => "defaults,_netdev,volume-
name=home",
          require => [ Package["glusterfs-client"] ],
        ] }
```

4 Conclusions

Glite grid middleware is a very complex stack of software, and the administration of a grid site requires a proper solution for the configuration management. Various applications have been used for this task, and this paper presented Puppet as a novel approach for a complete site configuration, including the gLite grid middleware.

As a future work we plan next to replace the YAIM tool which was used for the configuration of the gLite middleware, and define the "policy" for gLite Grid services

as Puppet specific manifests. This will be done in accordance with the configuration scripts for gLite Grid middleware.

References

1. Stanciu, A., Neagu G.: Help desk structure for the support service of a Virtual Organization supported by multiple grid infrastructures. In: Proc. of the 17th Int. Conference on Control Systems and Computer Science (CSCS17) , Volume 1, ISSN 2066-4451, pp: 429-432 (2009).
2. Traugott, S., Brown L.: Why Order Matters: Turing Equivalence in Automated Systems Administration. In: Proceedings of the 16th LISA Conf. , p. 99 (2002).
3. Turnbull, J.: Pulling Strings with Puppet: Configuration Management Made Easy . Springer, ISBN 978-1590599785 (2008).
4. Couch A., Sun Y.. On the Algebraic Structure of Convergence. Lecture Notes in Computer Science, Volume 2867. (2003)