Devolved Management of Distributed Infrastructures With Quattor

Stephen Childs – Trinity College Dublin, Ireland
Marco Emilio Poleggi – INFN-CNAF, Bologna, Italy
Charles Loomis – Laboratoire de l'Accélérateur Linéaire (LAL), Orsay, France
Luis Fernando Muñoz Mejías – Universidad Autónoma de Madrid (UAM), Spain
Michel Jouvin – Laboratoire de l'Accélérateur Linéaire (LAL), Orsay, France
Ronald Starink – Nikhef, Amsterdam, The Netherlands
Stijn De Weirdt – Universiteit Gent, Ghent, Belgium
Germán Cancio Meliá – European Organization for Nuclear Research (CERN), Geneva, Switzerland

ABSTRACT

In recent times a new kind of computing system has emerged: a distributed infrastructure composed of multiple physical sites in different administrative domains. This model introduces significant new challenges: common configuration parameters must be shared, local customization must be supported, and policy domains must be respected. We believe that such features can best be implemented by a system that provides a high-level configuration language (allowing structuring and validation of configuration information) and that is modular (allowing for flexible structuring of the overall infrastructure).

The Quattor configuration management toolkit has been designed to meet these requirements. Quattor uses a declarative model where high-level descriptions are translated into configurations and enacted by autonomous components running on the configured machines. Quattor's Pan language provides features for composing complex configuration schemes in a hierarchical manner, for structuring configuration information along node or service lines, and for validating parameters. Finally, the modular architecture of Quattor allows great flexibility in the placement of configuration servers within a distributed infrastructure.

Quattor is being successfully used to manage distributed grid infrastructures in three countries. The suitability of the Pan language is demonstrated by the fact that a comprehensive distribution of configuration templates has been developed as a community effort.

Introduction

Distributed computing paradigms such as grid and cloud computing are changing the face of system configuration management. The traditional computing center made up of hundreds or thousands of computers located at a single site is being reconstituted as a federated system where resources are spread across multiple sites. Each of these collaborating sites, whether independent institutions or departments within a larger organization, requires enough autonomy to implement local policies, and hence the management of the overall infrastructure must be devolved. Nevertheless, a consistent view of the overall system must be provided to resource users. Management strategies for distributed infrastructures have an inherent tension: effective mechanisms for sharing common configuration must be provided without restricting the independence of individual sites.

The high-energy physics community has extremely demanding data processing requirements and has been involved in large-scale computing for decades. It was one of the first communities to see the need to rethink the traditional computing center model, and now concentrates on the creation and maintenance of a global grid infrastructure. An important consequence has been a focus on tools for managing the "fabric" of the grid, namely, the hardware and software installed at sites around the world. Such tools must be scalable to deal with the large installations found at grid sites, flexible to deal with complex services on heterogeneous resources, and modular to deal with the wide range of site structures (including distributed sites). Quattor was designed and implemented to deliver an infrastructure management system that meets these needs.

Grid infrastructures have now moved beyond the prototype stage and one of the biggest challenges currently being faced is the provision of sustainable, managed, and monitored production system. In order to achieve this, a number of grid initiatives have put in place structures for aggregating multiple locations as logical sites. In some cases, this arrangement mainly affects administrative and support structures. In other cases, an integrated configuration management system is put in place to reduce the management load on any one site by allowing individual institutions within a

logical site to share common configuration parameters and tools. We believe this is the better approach and that Quattor is an excellent tool for implementing such a system.

Principles

The challenge of structuring and sharing components in a collaborative system is not new; over the years programming language designers have attacked this problem from many angles. While trends change, the basic principles are well understood. Features such as encapsulation, abstraction, modularity, and typing produce clear benefits. We believe that similar principles apply when sharing configuration information across administrative domains.

The Quattor configuration toolkit derives its architecture from LCFG [1], improving it under several aspects. At the core of Quattor is Pan, a high-level, typed language with flexible include mechanisms, a range of data structures, and validation features familiar to modern programmers. Pan allows collaborative administrators to build up a complex set of configuration templates describing service types, hardware components, configuration parameters, users, etc. The use of a high-level language facilitates code reuse in a way that goes beyond cut-and-paste of configuration snippets. (See the ''Pan Language'' section.)

The principles embodied in Quattor are in line with those established within the system administration community [2, 3]. In particular, all managed nodes retrieve their configurations from a configuration server backed by a source-control system (or systems in the case of devolved management). This allows individual nodes to be recreated in the case of hardware failure. Quattor handles both distributed and traditional (single-site) infrastructures (see Table 1).

We consider devolved management to include the following features: consistency over a multi-site infrastructure, multiple management points, and the ability to accommodate the specific needs of constituent sites. There is no single ''correct'' model for a devolved infrastructure, thus great flexibility is needed in the architecture of the configuration system itself. Sometimes a set of highly-autonomous sites wish to collaborate loosely. In this case each site will host a fairly comprehensive set of configuration servers, with common configuration information being retrieved from a shared database and integrated with the local configuration. This is the model used in the Belgian grid infrastructure BEGrid [4]. In a closer collaboration, it may be desirable to centralize the majority of services (configuration database and software package repository), with a bare minimum of services (maybe just node installation) hosted at the individual sites. Grid-Ireland, the Irish grid initiative, is organized along these lines [5].

Distributing the management task can potentially introduce new costs. For example, transmitting configuration information over the WAN introduces latency and security concerns. Quattor allows servers to be placed at appropriate locations in the infrastructure to reduce latency, and the use of standard tools and protocols means that existing security systems (such as a public key infrastructure) can be harnessed to encrypt and authenticate communications.

Applicability

Theory is one thing, but practical implementation brings its own significant challenges. In this paper we use the GRIF [6] deployment, the research grid infrastructure in the Paris region, to illustrate the strengths of Quattor for distributed management. GRIF was formed in 2005 as a collaboration between five sites wishing to present a unified view of their disparate resources. The goal was to amortize the management load of complex grid software by collaborating on the development of shared configuration templates, and to improve service quality by sharing automatically best practices via these templates. The requirements of GRIF, along with those from other infrastructures, have driven the development of a comprehensive set of Quattor templates, known as the Quattor Working Group (QWG) templates [7]. This can be thought of as a ''configuration distribution'' (after the model of a Linux distribution); it gathers together all the potential settings needed to set up basic operating system (OS) services and middleware.

GRIF was built on the foundation of the pre-existing LAL [8] Quattor configuration, which at that time was managing 20 machines. Since the inception of GRIF in 2005, all five sites have added machines to reach the current (2008) number of more than 600 machines spread over six locations, with 100-200 more expected next year. LAL also uses Quattor to manage non-grid servers (now roughly 50 machines) and Linux desktops (25); other GRIF sites are also starting to manage non-grid systems with Quattor.

One of the main challenges that the development of the QWG templates had to address was the shortage of manpower within GRIF. There is little dedicated manpower, and many of the technical team only work part-time on administration. In 2005, only three people had significant Quattor and/or grid experience; Quattor has enabled the incremental growth of a team which today consists of 20 people with good skills. However, due to the support for sharing configuration, only two to three people need to be involved in the development of core templates. Quattor and QWG, despite the somewhat steep learning curve, allow this ''sustainable'' model wherein the operational overhead is minimized allowing people to focus on the services for which they are responsible.

The rest of the paper is organized as follows. The next section describes a distributed management workflow and shows how it can be implemented in Quattor. We flesh out the theory by exploring real deployments of Quattor-managed distributed infrastructure in the ''Real-world distributed management'' section.

Distributed sites often face several management issues, so we present our deployment experiences and then discuss related work, and finally, we outline conclusions and future work.

Devolved Workflow for Distributed Management

We now present a typical workflow for devolved management of a distributed infrastructure and show how it can be implemented using Quattor. For a more in-depth treatment of individual Quattor components, please see a previous work [9].

Figure 1 illustrates the entire ''ecosystem'' of a devolved management infrastructure, showing the typical workflow involved in creating and deploying configurations. To start, administrators create or edit configuration source files, called *templates*, describing the services and nodes in the system. These templates are written in Pan [10, 11], a high-level, declarative configuration language. Administrators then store these templates in a *configuration database*, which may aggregate configuration templates from various sources. The configuration templates are then processed by the Pan compiler which validates their content and compiles them into an XML representation. The resulting XML *profiles* are stored in a *machine profile repository*; there is exactly one profile per managed node. Each managed node retrieves and caches its own profile and autonomously aligns its state with that described in the profile. Corrective actions are performed by specialized agents, or components, which are triggered upon changes in the part of the configuration with which they are registered.

Administrators may or may not be located at the same site as their managed nodes. In the figure, the organization bar.org manages its own machines; in contrast, the management of machines in foo.org is devolved to administrators in baz.org. In fact, Quattor's architecture means that the communication between any stages in the workflow may be carried out across site boundaries.

Configuration Management System

Quattor's configuration management system is composed of a configuration database that stores high-level configuration templates, the Pan compiler that validates templates and translates them to XML profiles, and a machine profile repository that serves the profiles to client nodes. Only the Pan compiler (see the ''Pan Language'' Section) is strictly necessary in a Quattor system; the other two subsystems can be replaced by any service providing similar functionality.

Devolved management in a cross-domain environment requires users to be authenticated and their operations to be authorized. For the configuration database, we chose to adopt X.509 certificates[1] because of the support offered by many standard tools,

[1]Kerberos 5 tickets and encrypted passwords are supported as well.

	Distributed			Single-site			
Metric	**BEGrid**	**Grid-Ireland**	**GRIF**	**CERN**	**CNAF**	**Nikhef**	**UAM**
Managed machines	260	417	619	8000	800	301	553
Administrators	8	11	25	100	10	4	3
Physical sites	6	18	6	1	1	1	1

Table 1: Quattor deployments.

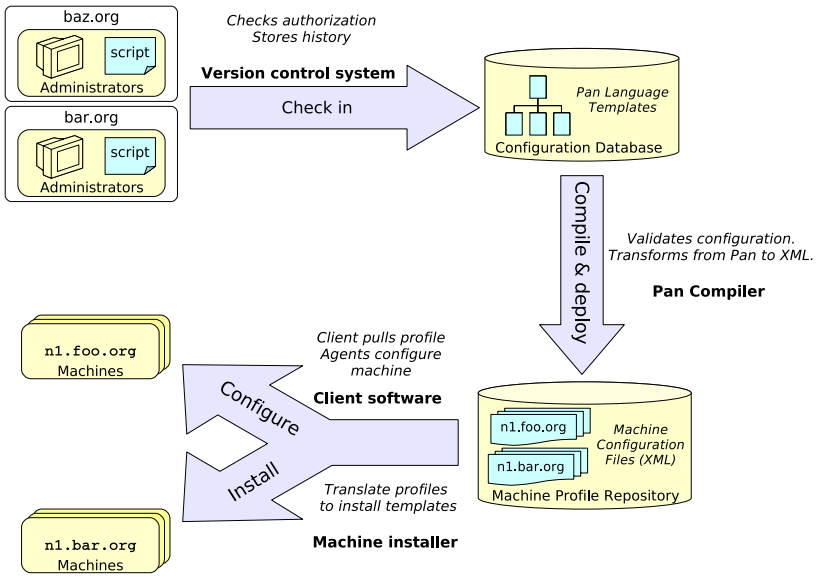

Figure 1: Configuration data workflow in Quattor.

and access control lists (ACLs) because they allow a fine-grained control (an ACL can be attached to each template). When many users interact with the system, conflicts and misconfigurations may arise which require a roll back mechanism; to this purpose, a simple concurrent transaction mechanism, based on standard version control systems, was implemented.

Quattor's modular architecture allows the three configuration management subsystems to be deployed in either a distributed or centralized fashion. In the *distributed* approach, profile compilation (at development stage) is carried out on client systems, templates are then checked in to a suitable database, and finally the deployment is initiated by invoking a separate operation on the server. The *centralized* approach provides strict control of configuration data. The compilation burden is placed onto the central server, and users can only access and modify templates via a dedicated interface.

Since the two paradigms provide essentially the same functionality, the choice between them depends on which fits the management model of an organization better. For instance, the centralized approach fits large computer centers well because of its strictly controlled workflow, whereas multi-site organizations such as GRIF prefer the distributed approach because it allows different parts of the whole configuration set to be handled autonomously. In this paper, we focus on the distributed approach (see the ''Real-world Distributed Management'' section) as it fits best with the devolved management model we are presenting.

Pan Language

The Pan language compiler sits at the core of the Quattor toolkit. It compiles machine configurations written in the Pan configuration language by system administrators and produces XML files (profiles) that are easily consumed by Quattor clients. The Pan language itself has a simple, declarative syntax that allows simultaneous definition of configuration information and an associated schema. In this section, we focus only on the Pan features that are relevant to devolved management of distributed sites: validation, configuration reuse, and modularization. The original specification of Pan can be found in [10]; a better description of the current features of Pan is available in [11].

Validation. The extensive validation features in the Pan language maximize the probability of finding configuration problems at compile time, minimizing costly cleanups of deployed misconfigurations. Pan enables system administrators to define atomic or compound types with associated validation functions; when a part of the configuration schema is bound to a type, the declared constraints are automatically enforced.

Configuration reuse. Pan allows identification and reuse of configuration information through ''structure templates.'' These identify small, reusable chunks of Pan-level configuration information which can be used whenever an administrator identifies an invariant (or nearly invariant) configuration subtree.

Modularization. With respect to the original design, two new features have been developed to promote modularization and large-scale reuse of configurations: the *namespacing* and *loadpath* mechanisms.

A full site configuration typically consists of a large number of templates organized into directories and subdirectories. The Pan template namespacing mimics (and enforces) this organization much as is done in the Java language. The namespace hierarchy is independent of the configuration schema. The configuration schema is often organized by low-level services such as firewall settings for ports, account generation, log rotation entries, cron entries, and the like. In contrast, the Pan templates are usually organized based on other criteria like high-level services (web server, mail server, etc.) or by responsible person/group.

The namespacing allows various parts of the configuration to be separated and identified. To effectively modularize part of the configuration for reuse, administrators must be able to import the modules easily into a site's configuration and to customize them. Users of the Pan compiler combine a loadpath with the namespacing to achieve this. The compiler uses the loadpath to search multiple root directories for particular, named templates; the first version found on the loadpath is the one that is used by the compiler. This allows modules to be kept in a pristine state while allowing sites to override any particular template.

Further, module developers can also expose global variables to parameterize the module, permitting a system administrator to use a module without having to understand the inner workings of the module's templates.

The ''Real-world Distributed Management'' section explains the use of the Quattor Working Group (QWG) templates used to configure grid middleware services. The QWG templates use all of the features of Pan to allow distributed sites to share grid middleware expertise.

Automated Installation Management

A key feature for administering large distributed infrastructures is the ability to automatically install machines, possibly from a remote location. To this purpose, Quattor provides a modular framework called the Automated Installation Infrastructure (AII). This framework is responsible for translating the configuration parameters embodied in node profiles into installation instructions suitable for use by standard installation tools. Current AII modules use node profiles to configure DHCP servers, PXE boot and Kickstart-guided installations.

Normally AII is set up with an install server at each site. However, the above mentioned technologies allow the transparent implementation of multi-site

installations, by setting up a central server and appropriate relays using standard protocols [12, 13].

Node Configuration Management

In Quattor, managed nodes handle their configuration process autonomously; all actions are initiated locally, once the configuration profile has been retrieved from the repository. Each node has a set of configuration agents (components) that are each registered with a particular part of the configuration schema. For example, the component that manages user accounts is registered with the path /software/components/accounts. A dispatcher program running on the node analyzes the freshly retrieved configuration for changes in the relevant sections, and triggers the appropriate components. Run-time dependencies may be expressed in the node's profile, so that a partial order can be enforced on components' execution. For example, it is important that the user accounts component runs before the file creation component, to ensure that file ownership can be correctly specified.

By design, no control loop is provided for ensuring the correct execution of configuration components. Site administrators typically use standard monitoring systems to detect and respond to configuration failures. Nagios [14] and Lemon [15] are both being used at Quattor sites for this purpose. In fact, Lemon has been developed in tandem with Quattor, and provides sensors to detect failures in Quattor component execution. We discuss integration of external tools further in the ''Integration with External Tools'' section.

While nodes normally update themselves automatically, administrators can configure the system to disable automatic change deployment. This is crucial in a devolved system where the responsibilities for, respectively, modifying and deploying the configuration may be separated. A typical scenario is that top-level administrators manage the shared configuration of multiple remote sites and local managers apply it according to their policies. For instance, software updates might be scheduled at different times.

Software Management

Quattor offers a software package management framework based on the separation of package *repository* and *configuration*. One or more physical repositories can be placed anywhere provided they are accessible via HTTP; the repositories' contents are made known to the configuration system via special Pan templates. Each time the content of a repository changes, the corresponding template must be regenerated. A node's software composition is specified by package lists placed in the configuration templates: package names are checked against the repository templates, and the resulting list is compiled to a machine profile. In a distributed multi-site scenario, the benefit of this separation is clear: package repositories can be placed in strategic locations, typically close to the user sites, even though package lists are stored in a remote location.

The package manager can be configured to act either in a secure mode, where locally-installed packages are automatically removed, or in a flexible mode, where these packages are ignored. The secure mode allows tight control over packages installed on a node, indeed, the package manager will never try to install anything which is not explicitly listed in the node's profile. Rollbacks can be easily performed as the package manager executes operations transactionally, checking that no requested change will result in a dependency conflict. In flexible mode, several versions of the same package can be installed and the package manager can be configured to respect manual installation of packages not listed in the machine profile. This is ideal for devolved management: even when some central policy strictly dictates the basic software composition, local administrators may experiment with customized setups while having the guarantee that their systems can be cleaned up at any time.

Real-World Distributed Management

The Quattor approach to managing distributed infrastructure proved to be effective in the deployed use case detailed in this section. We will focus on the example introduced earlier: the QWG templates for managing a collection of resources spread across multiple institutions as a single grid infrastructure. We show how Quattor provides sufficient support for sharing configuration between sites and for structuring each site according to its own requirements. We illustrate this using examples from the GRIF deployment, which now (2008) comprises more than 600 machines spread over six locations.

Structuring Shared Configuration

A configuration framework which is based on a corpus of shared templates needs to address three main ''structuring'' issues: template distribution, configuration deployment and organization of the infrastructure's configuration. The following sections illustrate how the QWG framework tackles them.

Structured Configuration Distribution

The configuration of grid software (for example, the Globus [16] and gLite [17] middleware) is in itself a challenge for configuration management systems. Grid software includes a wide variety of logical services (worker node, compute server, storage server, etc.), and it is possible for these to be combined on a single physical node. In a distributed infrastructure, certain parameters are common across entire infrastructures (such as the addresses of central grid servers), there are also many configuration parameters specific to each site (such as network settings, local user details, etc.), and local variations in how the services are configured or combined. The challenge in a distributed infrastructure is how to share common configuration while retaining maximum flexibility for local site customizations.

The solution to this problem in our context has been to develop a core distribution of Quattor templates that can be used with minimal customizations by sites. This distribution, known as the *QWG templates*, is the result of a collaborative effort among a set of European grid sites. This distribution can be used "as is" by collaborating sites; all they need to do is configure site-specific parameters (e.g., network addresses). New services or OS distributions need only to be integrated once into the core distribution, and can then be used by all collaborating sites. Additionally, there is substantial scope for customization. The standard templates contain variables that conditionally include local templates with custom settings, providing great flexibility without requiring modification of the core templates. Together, these features reduce the potential for creating incompatible forks and minimize the number of templates maintained at each site.

The QWG distribution has grown dramatically as new services and OS distributions have been added. At the time of writing, a checkout of the QWG templates includes 2805 Pan templates: 1167 for 10 OS distributions, 964 for grid services, 550 for standard components, 115 example templates, and 9 legacy templates. A large proportion of these templates is automatically generated by processing package lists distributed by OS or grid vendors. Even so, a distribution of this size needs to be carefully structured to be manageable; indeed, Pan namespaces are used extensively to tie templates to locations in the directory structure.

Structured Deployment

In order to use the QWG templates, a site must implement a suitable configuration database as described in the "Devolved workflow for distributed management" section. We have developed the Subversion Configuration Database (SCDB) [18] to provide a configuration database suitable for cross-domain use. Authorization is enforced by the version control system (Subversion by default, although CVS has also been successfully used). In this way, standard credentials can be used and a rich set of source control options are available.

In order to share administration load, several projects employ a single Quattor instance to manage a grid infrastructure distributed over multiple sites. Examples are GRIF [6] in Paris, BEGrid [4] in Belgium, and Grid-Ireland (see Table 1 for characteristics). A feature of such management schemes is that access to the configuration must be controlled based on the identity of the user, so that administrators from different sites can safely implement their own modifications without disrupting other sites. This is easy to implement when configuration information is stored in a version control system.

For example, within GRIF, administrators at each site have full control of their own clusters and site templates, but the right to modify standard and GRIF-wide templates is restricted to a set of experts. This ensures that the core contains only well-tested templates. However, this does not prevent local administrators from deploying a modified version of a core template at their own site. To ensure autonomy for local administrators, somebody who does not have the right to modify a "global" template can make a copy in his own cluster or site area and then modify it. When he has tested and validated his changes, he can submit the new template to one of the experts for integration in the standard template set. This approach encourages contributions from all users, yet enforces strict control over the core distribution.

A variety of techniques is used for integrating templates from the core repository with a particular site's (or infrastructure's) own configuration database. Within GRIF and BEGrid, QWG templates are merged into the local repository. Grid-Ireland uses Subversion's "externals" [19] mechanism which allows direct inclusion from an external repository, effectively inserting a pointer to a particular revision of the core templates, causing them to be automatically updated as required. Local templates are then certified against a particular revision of the core templates, which is then used until new features are

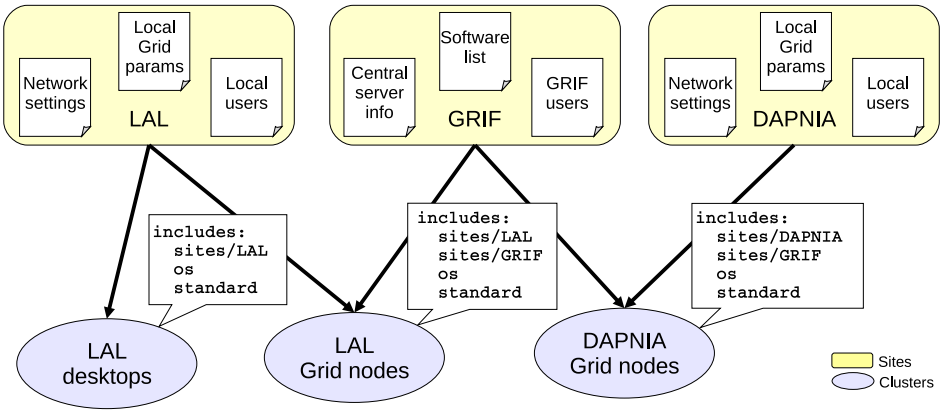

Figure 2: Sites and clusters in QWG.

needed from the core – at this point a new revision of the core templates is certified.

Structured Infrastructure Configuration

The configuration is usually organized using the concepts of *site* and *cluster*. A *cluster* is an arbitrary grouping of machines that share configuration information (for example, ''compute nodes'' or ''grid servers''). A *site* is a logical group that defines a set of configuration elements to be shared by different clusters. For example, in Figure 2, there are three sites. The ''LAL'' and ''DAPNIA'' sites contain local configuration data such as network settings and user accounts; the ''GRIF'' virtual site contains configuration parameters common to all GRIF machines (location of central grid services, common software lists, and user accounts). The LAL and DAPNIA grid clusters include templates from their local site and GRIF, while the LAL desktops cluster only includes the local LAL templates as it is not part of the GRIF infrastructure.

The sites associated with a cluster are specified as an ordered list that defines their precedence in the template search path. For example, for the ''LAL Grid nodes'' cluster, if the template users.tpl exists in both the LAL and GRIF sites, the version from LAL will be selected as LAL appears first in the list. This technique is used within QWG to create a hierarchical structure; a template in the core distribution can be easily overridden by creating a copy of it in a higher-priority site (e.g., the local site).

Local Customization

Quattor and Pan as used in the QWG templates provide two main methods for customization: loadpaths, which are used for high-level switches between versions and hook variables, which allow the inclusion of custom templates to tweak a specific area of the configuration space.

Loadpaths. Each site is likely to deploy a different mix of base OS types, each of which needs its own package lists and configuration. From time to time, a node (say a web server) will need to be upgraded to a new OS release, while preserving any custom configuration. For example, consider a Xen host machine which needs to install OS-specific RPM packages for the core Xen software; the machine's ''object'' template includes the template rpms/xen/host:

```
object template xenhost01;
variable LOADPATH = list('os/sl450-x86_64');
include {'rpms/xen/host'};
```

and the template referenced is available for two platforms in two different directories:

```
os/sl450-x86_64/rpms/xen/host.tpl
os/sl510-x86_64/rpms/xen/host.tpl
```

The LOADPATH definition instructs the compiler to prefix its search path with the string os/sl450-x86_64, so that the template found is os/sl450-x86_64/rpms/xen/host. By simply changing the loadpath, the machine profile can be upgraded to use a different version. A similar technique can be applied for version-specific configuration, for example variables that take different values depending on the OS version. In QWG the loadpath for OS templates is set using a simple hash structure that maps machine names to OS version. A configurable default is also provided removing the need to set up a mapping for each machine.

Hook variables. Another approach commonly used in QWG is the use of hook variables that allow sites to integrate their own custom templates. Most of the core templates that configure services or node types contain a conditional include block. The block checks the value of a variable and includes the referenced template if the variable is set.

For example, the template that sets up the basic services for a cluster head node contains the following line:

```
variable CE_CONFIG_SITE ?= null;
# Add local customization to standard
# configuration, if any
include { CE_CONFIG_SITE };
```

In order to customize a local head node, an administrator creates a new template containing the customization (say tcd_torque_config.tpl) and then simply sets the variable CE_CONFIG_SITE to the name of this template. The basic machine type can thus be customized without modifying the core template. Note also the use of ?= in the variable assignment: this is a conditional assignment meaning ''assign if a value is not already set.'' This allows default values to be set that will be used if no local configuration is defined.

A Concrete Example

Figure 3 shows a partial inclusion graph of a worker node foo in the cluster bar at LAL.

The QWG framework defines a set of ''machine types'' corresponding to typical grid service elements and represented by dedicated templates in the namespace machine-types. The part enclosed in the dashed-line box represents the customization space: these templates are looked up in a cluster-dedicated namespace site/bar/ set externally as an inclusion path to the compiler. A first level of local LAL's site-wide customization is done in machine-types/wn, where a hook variable WN_CONFIG_SITE defines a possible extra template subtree as discussed above: for instance, some of the ''local grid parameters,'' as shown in Figure 2, are set here. A second level of customization for the OS is done in site/cluster_info, where the variable NODE_OS_VERSION_DB defines the template in which the mapping between machine names and platforms is done (site/os/version_db): here foo is mapped to sl450-i386. At this point, all the information for selecting OS-dependent templates is almost completely defined: the last bit is the loadpath which is set in os/version; then config/glite/3.1/base and its cascading templates are

searched in the namespace os/sl450-i386. With reference to Figure 2, all the information in this subtree comes from the GRIF site's template set.

To give an idea of the configuration space size of a worker node, there are 32 platform-dependent templates and 195 platform-independent templates with local site-wide customizations. In addition, there are 25 upstream platform-dependent templates (out of 1208 OS templates) and 138 upstream platform-independent templates (out of 1109 grid and standard templates) without local site-wide customization.

Flexible Deployment Architecture

The use of a high-level language and a source control system facilitate maintainable sharing and customization of configuration information between sites. However, flexibility of deployment architecture is also important for integrating disparate sites.

It is important not to mandate a deployment architecture for sites wishing to collaborate. Local preferences or policies may lead to different implementations at partner sites. Because Quattor is highly modular, each site can choose which elements of their infrastructure to share and which to keep private. Often a similar configuration will be used for all sites within a collaboration, but this is not required. Figure 4 shows a variety of sites participating in a single distributed infrastructure. There is a common configuration database holding core templates and parameters common to all sites, and a shared package repository holding common software. Each site has a set of local machines and an installer that performs their initial installation.

- **Site 1** is an advanced site with a high degree of customization. It hosts its own local configuration database with site-specific templates. These are combined with the core templates from the common database to generate a complete configuration for the site, with machine profiles served locally. This is essentially the model used within BEGrid [4].
- **Site 2** still requires local configuration but to a lesser extent. It does not install custom software locally and so uses the shared package repository exclusively. While the local configuration database is logically distinct, it may actually be hosted on the same server as the shared repository. This is the case within GRIF [6].
- **Site 3** is completely dependent on the shared configuration and is thus centrally managed.

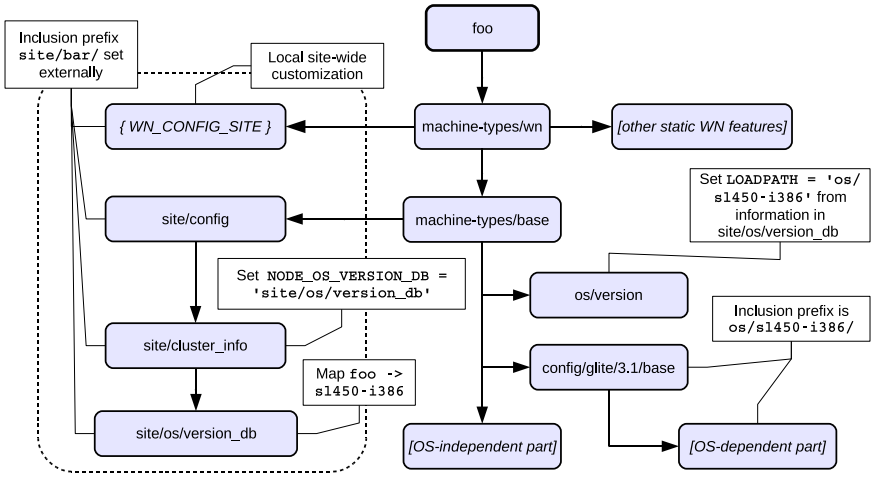

Figure 3: A partial inclusion graph of a worker node in QWG.

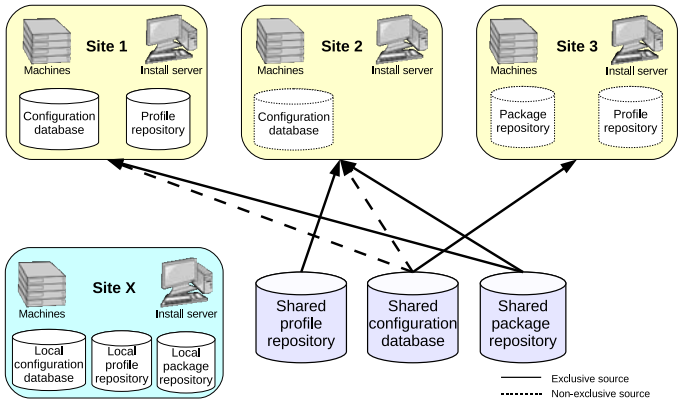

Figure 4: Site configurations in a distributed infrastructure.

All configuration and software are sourced from the shared database. High-level templates are compiled to low-level profiles at the central site and pushed out to a machine profile repository at the site. A similar process is used for software; packages are mirrored locally, but are treated as part of the same logical repository. This is the model used within Grid-Ireland [5].

For comparison, Site X is a traditional computing center where all configuration and software packages are retrieved from local servers, as happens at CERN's computer center [20].

Experience with Distributed Deployment

The QWG templates have been used to configure production sites since 2005. As such, we have gathered some experience on using Quattor to manage sites, and we present some reflections here.

What Worked Well

Distributed configuration database. The original installation of Quattor at CERN uses a single configuration database which is accessed via a proprietary interface that enables users to check in and compile templates. This approach does not allow offline working as it requires a connection to the central database at all times. For this reason, a Subversion-based configuration database was developed which allows administrators to treat configuration like source code, using all the features of the version-control system directly. Thus configurations can even be developed and tested (at least to confirm that they compile correctly) on an offline laptop, and then deployed when network connectivity is available again.

Complete representation of system state. One of the key benefits of Quattor profiles is that they hold the complete configuration state of a particular system in a structured form. Within a particular machine's configuration, this means that multiple components can access the system configuration (for example, the list of users defined could be accessed both by the ssh component and the accounts component). Machines can also access the configurations of other machines, which is useful for configuring monitoring servers that need to aggregate descriptions of the whole site, and for virtual machines, where host VMs need access to the configuration of their guests. This is done in a controlled way, via Pan constructs that allow to access machine profiles' contents.

Namespaces and loadpaths. These two features (new since the earlier Pan paper) have proved to be extremely useful in managing the large set of templates now in use. Namespaces allow hierarchical naming schemes to be enforced, resulting in a clear structure for the overall distribution (for example, a component schema definition template (schema.tpl) with the name components/accounts/schema **must** be located in a directory whose path ends in components/accounts.

Loadpaths have proved useful in keeping machine template code OS-agnostic as detailed earlier in the paper. They allow large portions of a machine or service configuration to be easily switched to a different OS or middleware version (the template does not usually have to be changed at all, as the change is made in an external file). They have also been used to implement "stage-based deployment." In this approach, templates are classified as "development" or "production," and stored under different paths accordingly. Again, the switch from development to production can be quickly affected by changing the value of the loadpath variable. Furthermore, loadpaths are used for installing and configuring different versions of the same component.

What Does Not Yet Work Quite So Well

Here we list some ongoing limitations that impact on the day-to-day task of managing sites, together with our current thoughts on resolving them.

Software dependency management. Quattor's software package manager is declarative: administrators define the exact list of packages to be installed on a node, and the package manager is normally run in an enforcing mode so that no other packages are permitted to be installed locally. This behavior is different from other commonly-used package managers such as YUM and APT which automatically pull in packages to satisfy the dependencies of a newly-installed or updated package.

The advantage of this approach is that administrators have strict control over software on managed nodes. The disadvantage is that package dependencies must be comprehensively defined at configuration time rather than allowing the package manager to resolve them automatically. If a package list is deployed which contains unsatisfied dependencies, then the whole package manager transaction will fail. As many other components depend on the package manager having run successfully, the result is often that the new configuration does not take effect. As this failure mode was only detected after deployment, it proved extremely frustrating (becoming known in the community as "RPM dependency hell").

This is a difficult problem to solve at the time of profile creation because package dependency information is contained in the packages themselves which are typically stored in a central repository. Rather than implementing our own dependency checker, we decided to leverage the YUM package manager, whose functions are easily accessible from Python code. We have written a Python program that processes a given Quattor profile, extracts details of software repositories used, generates YUM repository descriptors to represent them, and then invokes YUM code to indicate whether there are missing dependencies in the RPM set. This approach relies on YUM metadata being kept up to date on the package repository server, but this can easily be automated.

Dependencies on external tools. One of the goals of creating a shared configuration distribution is to reduce the learning curve needed to establish a new Quattor installation. Quattor has been designed to make as much use of standard services as it can. For example, network installation in Quattor uses DHCP, PXE, and Kickstart, as well as relying on correct DNS configuration. In principle this is a good thing as it makes Quattor easier to integrate with existing infrastructures. However, when new, inexperienced people want to join a collaboration with a new site, they must master all of these services (at least to the point where they can debug problems) before they can install a single box!

Fine-grained authorization. As configuration management becomes more widely distributed through a larger number of people, authorization and control become extremely important in maintaining the quality of the deployed machine configurations. A critical issue in distributed management is that of fine-grained access control to restrict users from interfering with configuration that lies outside their sphere of control. To date, this has been handled by enforcing authorization for templates in the configuration database. For example, administrators of a particular cluster might be restricted to editing the templates for their own cluster machines in an effort to prevent them from modifying the configuration of core services. However, this is not a sufficient solution, as any template can modify any part of the configuration namespace. A solution is needed that ties authorization at the template level (enforced by the configuration database) to authorization at the level of portions of the profile namespace (this must be enforced by the compiler).

Structure templates (see the ''Pan Language'' section) provide a partial solution for this. An administrator allowed to edit such a template can modify only a small and well defined sub-tree of the configuration hierarchy. However, a structure template can read the entire hierarchy, and this still has to be restricted in some way.

Debugging a complex configuration. The tradeoff for having so much flexibility is an increased difficulty in debugging a complex configuration instance. Indeed, the high-level Pan representation is based on dynamic includes which are resolved at compilation time, so that, ''navigating'' the templates becomes almost impossible. This has increasingly generated frustration among QWG users, limiting the possibility of a closer than just operational approach for people who wish to better understand and contribute to Quattor developments. As a response, we are developing some facilities for visualizing and browsing a site's configuration via different graphic formats.

Lessons Learned

Here we describe the features which have proved to be crucial in sharing configuration between sites.

Stability of the core configuration. One of the key problems for early adopters of the QWG template distribution was the instability of configuration mechanisms. The developers were still working out the best way to represent the complex configuration, and so the underlying data structures frequently changed. This caused great frustration for those using the configuration as local templates had to be continually adapted. As a result of this experience, backward-compatibility is now a key goal for the QWG templates. In many cases, this is achieved by hiding the internal details of the data structures behind simpler functions. For example, the configuration of Xen guests is performed by setting various parameters (e.g., the list of guests on a particular machine) and then including the `configure_xen_guests` template which fills in the relevant part of the profile. The internal representation may change, but the interface for users remains the same.

Low-effort mechanisms for applying updates. Quattor is principally used to configure grid systems that rely heavily on external distributions both for basic OS functionality and grid middleware. Updates to both are regularly released and must be deployed to ensure that systems remain secure. The general approach taken in QWG is to automatically include a template that ''upgrades'' any packages selected to the latest versions from the OS or middleware's updates. The update list is automatically generated from the OS or middleware updates directory.

Integration With External Tools

Implementing a distributed infrastructure requires peaceful coexistence with systems already in place at various sites. Collaborating sites often have existing commitments to deployed tools for monitoring, management, or reporting. A key requirement for a distributed configuration management system is the ability to integrate smoothly with such local systems. Because configuration information is stored in a structured fashion, it can easily be reused to generate configuration files for new external tools.

This topic is a major focus for the expanding Quattor community as new tools are encountered. In this section we describe some existing integration work that links systems for monitoring, virtualization, and user desktop management into Quattor infrastructures.

Monitoring. Configuration of monitoring tools such as Nagios [14] and Lemon [15] often requires the administrator to define lists of machines with their associated services. This information is needed to determine how to group machines together for reporting and which sensors to read on a given machine.

With Quattor, this information is already available, as configurations are structured in terms of service and node types. Hence the configuration lists needed for monitoring tools can be automatically derived from the configuration database. It has proved straightforward to implement Pan functions that

extract information about all existing nodes and provide it to the Nagios templates. These templates can then generate complex, fine-grained policies, for instance *raise an alarm if the load on a single-CPU node goes above three, but hold that same alarm on four-CPU nodes until the load reaches 8*, without explicitly instructing Nagios about how many processors each node has. Also, node-specific checks can be specified, and hosts not listed in the configuration database can be added to the Nagios configuration. This is useful when monitoring external services, such as routers.

Virtualization. A major strength of a Quattor configuration database is the ability to reference configuration parameters from other services or even machines. This has proved particularly useful in the configuration of virtualization, where a number of guest nodes are hosted as virtual machines (VMs) on a host. The configuration of a VM has two dimensions: the configuration of the node itself (its network settings, users, services, etc.) and the configuration needed on the host to run the VM (location of file system storage, virtual MAC address, boot mechanism, etc.). The node configuration resides in the profile as normal, and much of the "external" configuration resides in the machine's hardware description template which lists disk sizes, RAM size, and network cards. The host can thus reference this information and use it to generate the VM configuration files.

In broader terms, Quattor supports virtual machines using the concept of an *enclosure*. An enclosure is an entity composed of a parent and one or many children. The parent is always a Quattor representation of a physical machine whereas a child can represent either a physical or virtual machine. This allows modeling of real hardware enclosures as well as virtual machines.

Quattor-based virtualization management using Xen has already been deployed in production environments [21] and OpenVZ virtualization has also been implemented. A variety of methods for instantiating VM file systems has been used, ranging from full PXE-based automatic installation to bootstrapping from pre-built images. In all cases, AII is used to guide the installation and bootstrap procedure for virtual machines.

User desktops. Once administrators see the benefit of having grid clusters and services under Quattor management, they often look around to see whether other parts of their infrastructure can also benefit. At UAM, Quattor is used to manage Linux installations on about forty user desktops that also have a Windows partition. Typically, desktop users have little knowledge of system administration, and often tend to deviate from security policies, for instance, by installing unauthorized software.

Using Quattor allows administrators to keep all machines in compliance with site's policies. Quattor's management tools can be configured to automatically remove any unauthorized software and user accounts. Other security measures such as firewall rules or controlled privilege escalation (i.e., sudo) can also be configured by the administrators without imposing additional burdens on users.

AII can detect and preserve existing file systems on the disks, and so Windows installations can live side by side with Quattor-managed Linux installations. This is a specific example of a general principle in Quattor, where parts of the machine's configuration can effectively be ignored in order to delegate their management to other tools. This flexibility enables system administrators to "start small," initially just managing a small set of critical services. By gradually extending Quattor management, they can consolidate configuration information, making the most of their investment in Quattor.

Related Work

Fabric management systems abound in both the open source and the commercial sector. Some systems like Oscar [22] and Rocks Clusters [23] use pre-built system images, creating new machines by installing and customizing a pre-built file system. Active Directory [24] provides a hierarchical schema for managing network resources, but it does not easily allow handling arbitrary (i.e., non-Microsoft) services and SW packages. Other systems define declarative languages [3] to facilitate the configurations of machines within the fabric. For a complete survey, see the literature [25, 26]. We believe that systems using high-level declarative languages can provide the flexibility needed to implement distributed management infrastructures. Here, we compare the language features in four systems – LCFG [1], Cfengine [27], Puppet [28], and PoDIM [29] – to those in Pan.

LCFG relies on configuration source files expressed in a high-level declarative language which are transformed into low-level, machine-readable XML profiles, similar to Quattor. However, LCFG's configuration language, based on the C pre-processor, is limited when compared to Pan. It lacks user-defined types, user-defined functions, and validation constructs for expressing constraints (although see PoDIM discussion below). It doesn't provide any equivalent to Pan's namespaces and loadpath, whose importance we have demonstrated for devolved management of remotely distributed clusters based on different architectures. A number of the authors have experience with using LCFG to manage grid sites. The main limitations we found in practice were the lack of an overall configuration schema, which made it difficult to share information between components, and the lack of easy-to-use programming-language features such as hash data structures and iterators.

Cfengine is a policy-based configuration management system in which each managed host belongs to one or more *classes* for which some *policies* apply.

Since a policy specifies the actions requested to align a part of the system with the desired state, the configuration description is partly *procedural* [3]. This makes it difficult to reason about the relations of different configuration parts; in fact, there is no validation support. Although Cfengine's language provides functions for manipulating parts of the managed system, such as symbolic links, it lacks more refined constructs like type definitions and inclusion statements. This paradigm does not easily allow hierarchical schemes, limiting the possibilities for devolved management over distributed sites.

Puppet uses a high-level declarative language with a feature set similar to that of Pan's. In the Puppet language one can define *resources* with associated attributes. The language permits the administrator to define values as well as providing defaults. In addition, *classes* can be defined to group resources into high-level units. Further, *modules* can be defined and shared with others. The real difference with Pan is the validation. Puppet validates the schema on the server, but only validates the attribute values once the configuration is instantiated on the client. This provides late notification of problems and limits the possibilities of cross-machine validation.

PoDIM is a recently proposed language aiming, like Pan, to provide high-level configuration management [29]. Compilers for both languages operate in the same manner, converting source configuration files into a series of XML configuration files, one for each managed node. Pan uses a declarative syntax and PoDIM uses a rule-based one. One distinctive feature of PoDIM is its ability to define cross-machine constraints via the rules declared by the system administrator. Pan provides similar features via validation functions that can verify consistent configurations between machines. The primary difference is how inconsistent configurations are resolved. PoDIM solves for alternate configurations consistent with the declared rules; Pan requires that a system administrator change the configuration to resolve the identified problem.

Other important features of high-level configuration languages are the ability to define a configuration schema, mechanisms for propagating common configuration parameters between machines, and multiple inheritance of attributes and constraints. Both languages have facilities for all of them. PoDIM defines a schema through class attributes, rules, and invariants; Pan, as shown above, uses an extended typing system. In Pan, a managed object may only read information from another object's configuration. This is accomplished through use of the `value()` function and is critical for cross-object validation. PoDIM uses commands to allow managed objects to set attributes within other objects. PoDIM naturally allows multiple inheritance through the underlying Eiffel implementation. With Pan, any path can have multiple types bound to it, thereby allowing multiple inheritance of validation constraints.

PoDIM has an internal mechanism to authorize (or deny) certain changes to a configuration based on identity, an important feature in environments where many people maintain a fabric's configuration. Pan currently has no equivalent functionality. The Quattor toolkit instead controls access to particular source configuration files; this avoids accidental modifications but cannot prevent malicious changes.

Recent collaborative work between the LCFG and PoDIM teams [30], has shown that integration of the two is possible. It also identified a couple issues (slow performance and configuration oscillation) that will need to be eliminated before PoDIM can be used in production. The paper also indicates that a unified language combining the features of LCFG and PoDIM is desired; Pan to a large extent does combine them. Moreover, production use of Pan has been shown to be scalable and manageable, handling up to 8K machines efficiently in the deployment at CERN's computing center.

Conclusions and Future Work

With the adoption of grid computing, it has become increasingly common for different institutions to group their resources and present them as a single logical site. The consequence is a new type of fabric – a co-managed, distributed infrastructure composed of multiple physical sites in different administrative domains. Quattor meets the main requirements for the management of such a system: sharing of common configuration data, possibility of local site customizations, use of standard secure communication protocols, and flexibility.

Quattor's effectiveness mostly comes from the use of Pan, a declarative high-level configuration language which allows the definition of hierarchical configuration schemes using a simple syntax. The namespacing and loadpath features permit large-scale reuse of a configuration's parts and foster the collaborative deployment and management practices which enable the use case described in this paper. The most interesting result of this collaboration is a complete configuration framework, the QWG templates, intended for the configuration of grid services.

Having more actors on the scene requires a tighter control. Pan supports validation, that is, definition and checking of constraints before deployment. Also, appropriate means for authenticating and authorizing users and machine profile transfers are provided by Quattor. The result is an effective devolved management mechanism, with a reduced probability of service disruption due to misconfigurations or mischievous interventions.

Quattor's modularity and use of standard protocols make it attractive in a wide spectrum of different

site configurations. The elements of the Quattor toolkit – the configuration databases, tools for software installation management, tools for package repository management, and automated installation facilities – can be used, ignored, or replaced to meet the specific needs of a site, while peacefully coexisting with other management tools. The proof of this flexibility is the variety of use cases described in this paper: BEGrid relying mainly on local customizations for the configuration, GRIF which uses a local configuration repository and a shared package repository, and Grid-Ireland which depends completely on shared, centrally managed, repositories.

The Quattor toolkit is a mature solution used by a large number of diverse sites. Mature, however, does not mean static: the Quattor toolkit continues to evolve incrementally in response to the needs of the Quattor community, which now includes industrial as well as academic users. While Quattor usage is still concentrated within the academic grid community, the model we propose is more widely applicable. For example, large multi-national corporations already run their own grid-like infrastructures spread across geographical locations. Policies at individual sites may differ due to local historical or legal constraints, resulting in the same requirements for sharing and specialization we have already described. Mergers and acquisitions may result in an ever-changing set of policy domains and local tools that must all be integrated into a coherent whole. These are similar problems to those faced by distributed grid sites; similar solutions could be applied.

Quattor also functions well as a general fabric management system and is being used successfully in other, more traditional scenarios, such as that of a large centralized computer center like at Nikhef, CNAF, and CERN. In fact, many of those who started using Quattor to manage their grid infrastructures are now putting other aspects of their operations (mail servers, desktops, etc.) under Quattor control.

Quattor is a complex tool with a steep learning curve: although the community provides an active support, we need to improve our knowledge base, which entails providing clear use cases and possibly out-of-the-box solutions. We are also actively working on mitigating the difficulties encountered by new users via tools for visualizing a site's configuration. Another area for further development is the integration with tools for handling migration of virtual machines: this is somewhat conflicting with the base line dictating that a node should look as it is declared to be, since dynamically changing the location of a virtual machine necessarily modifies the node's configuration. Moreover, in response to increasingly rising security concerns, we are discussing how to extend Pan to support an authorization mechanism that directly protects parts of the configuration schema (not just templates). On the same line, we are working on integrating Quattor with SELinux: we need to define minimum contexts for the core services and confine them accordingly, as well as to manage target nodes' configuration via an NCM component.

Acknowledgements

The first version of Quattor was a product of the European DataGrid project and subsequent versions have been used to manage some grid resources of the EGEE series of projects. The authors gratefully acknowledge the European Commission's support of all of those projects. We would like to thank Kent Skaar and Narayan Desai for shepherding and Paul Anderson for encouraging the writing of this paper.

Author Biographies

Stephen Childs is an Associate Research Lecturer and Research Fellow at Trinity College Dublin. He is the Deputy Grid Manager for Grid-Ireland, and is responsible for fabric management and virtualisation. He is an active in grid operations and parallel jobs support within the EGEE project and a contributor to Quattor. He received a Ph.D. in Computer Science from the Cambridge University Computer Laboratory in 2002, and a B.Eng. in Computer Engineering from the University of Limerick in 1997.

Marco Emilio Poleggi received his M.Sc. in Electronic Engineering from University of Rome ''La Sapienza'' and his Ph.D. in Computer Engineering from the University of Rome ''Tor Vergata.'' During his studies, he focused on cache cooperation mechanisms for cluster-based web servers. He worked at CERN as a research fellow from 2005 to 2007, developing Quattor, for which he was also appointed release manager. He currently works at INFN-CNAF – the Italian National Center for Telematics and Informatics – where he manages the service configuration for the storage group. Marco Emilio is a member of the IEEE Computer Society.

Charles Loomis is currently a research engineer with CNRS in France and a founding partner of Six^2 Sàrl in Geneva, Switzerland. In 1992, he received his doctorate from Duke University in high-energy physics and subsequently worked as a researcher in major experiments on both sides of the Atlantic. His participation in grid technology projects started in 2001 with the European DataGrid project and continues to this day, within the EGEE series of projects. He initially played a major role in integrating, testing, and deploying the grid middleware. Now, he is responsible for the ''application'' activity within EGEE, working with end-users to ensure that the EGEE middleware and infrastructure meets their needs. Over this period, he has been a major contributor to the Quattor toolkit and is responsible for the Pan language compiler.

Luis Fernando Muñoz Mejías obtained his degree on Computer Science and Engineering at UAM (Autonomous University of Madrid, Spain) in 2006. He currently works as system administrator at UAM's

site of the Spanish LCG cloud. As a member of the Quattor community, Luis Fernando is responsible of the installation framework.

Michel Jouvin has been a system administrator for 25 years. He joined CNRS/LAL in 1992 and had responsibilities in many different areas; he joined the grid effort in the early 2000s. He became technical manager of the GRIF grid site when it was created in 2005. Michel has been involved with Quattor since 2004. He has been a contributor to various Quattor components and had a major role in re-engineering the first generation of QWG templates into a generic framework. He initiated the collaborative effort for documenting the Quattor tools, in particular the QWG wiki. He also participates in several EGEE/LCG bodies involved in grid operations.

Ronald Starink works as Research Associate at the National Institute for Subatomic Physics (Nikhef) in Amsterdam. Since 2005 he is responsible for grid system administration at Nikhef, which includes operations, storage systems and fabric management. He participates in the operations activity of the EGEE project and contributes to the Quattor community as maintainer of various components. Ronald received his Ph.D. in nuclear physics from the Free University Amsterdam in 1999. Subsequently he worked as a software engineer in the telecommunications industry and in business automation. In 2004 he joined Nikhef as software engineer, working on the grid project Virtual Laboratory for e-Science.

Stijn De Weirdt obtained his M.Sc. in physics at University of Ghent, Belgium in 2000. He started to work in November 2004 as system administrator at Free University of Brussels to setup a grid site for LCG (LHC Computing Grid) and BEGrid (Belgian research grid). For BEGrid, he integrated Quattor in the management of the distributed infrastructure. Since July 2008, he started to work at the University of Ghent to coordinate several scientific computing projects.

German Cancio received his M.Sc. after studying Computer Science at the UPM (Polytechnic University of Madrid, Spain) and the University of Savoie, France. After being a Research Assistant at the Artificial Intelligence Department of the UPM, German moved on to CERN in 1998 working on various activities related to large systems administration. He was the Fabric Management Architect of the European DataGrid project (2001-2004) and responsible for the development of Quattor until 2007. Since 2008, German is leading a team charged with the development of grid and mass storage management tools for CERN's Computing Grid. German is a member of USENIX, ACM and IEEE Computer Society.

Bibliography

[1] Anderson, Paul and Alastair Scobie, "LCFG – The Next Generation," *UK UNIX User Group Winter Conference*. UKUUG, 2002.

[2] Traugott, Steve and Joel Huddleston, "Bootstrapping an Infrastructure," *Proceedings of the 12th Large Installations Systems Administration (LISA '98) Conference*, 1998.

[3] Anderson, Paul, "A Declarative Approach to the Specification of Large-Scale System Configurations," 2001, http://www.dcs.ed.ac.uk/home/paul/publications/conflang.pdf .

[4] *The Belgian Grid for Research*, http://www.begrid.be/ .

[5] Coghlan, B. A., J. Walsh, and D. O'Callaghan, "Grid-Ireland Deployment Architecture," in Peter M. A. Sloot, et al., editors, *Advances in Grid Computing – EGC 2005*, LNCS3470, Springer, Amsterdam, The Netherlands, February, 2005.

[6] *GRIF – Grille de Recherche d'Ile de France*, http://www.grif.fr/ .

[7] *LCG Quattor Working Group (QWG templates)*, https://trac.lal.in2p3.fr/LCGQWG .

[8] *LAL – Laboratoire de l'Accélérateur Linéaire*, http://www.lal.in2p3.fr/ .

[9] García Leiva, R., et al., "Quattor: Tools and Techniques for the Configuration, Installation and Management of Large-Scale Grid Computing Fabrics," *Journal of Grid Computing*, Vol. 2, Num. 4, 2004.

[10] Cons, L. and P. Poznański, "Pan: A High-Level Configuration Language," *Proceedings of the 16th Large Installations Systems Administration (LISA '02) Conference*, 2002.

[11] *Pan Configuration Language Reference*, https://trac.lal.in2p3.fr/LCGQWG/wiki/Doc/panc .

[12] Droms, R., *RFC 2131 – Dynamic Host Configuration Protocol*, Technical report, Network Working Group, 1997.

[13] Patel, B., B. Aboba, S. Kelly, and V. Gupta, *RFC 3456 – Dynamic Host Configuration Protocol (DHCPv4) Configuration of IPsec Tunnel Mode*, Technical report, Network Working Group, 2003.

[14] *Nagios, An Open Source Host, Service, and Network Monitoring Program*, http://www.nagios.org/ .

[15] *LEMON – LHC Era Monitoring*, http://lemon.web.cern.ch/lemon/index.shtml .

[16] *Globus Toolkit, An Open Source Software Toolkit Used For Building Grids*, http://www.globus.org/toolkit/ .

[17] *gLite, An Open Source Grid Middleware Developed by the Egee Project*, http://glite.org/ .

[18] Jouvin, Michel, *Subversion Based CDB*, https://trac.lal.in2p3.fr/LCGQWG/wiki/Doc/SCDB .

[19] Subversion, *Externals Definitions*, 2008, http://svnbook.red-bean.com/en/1.4/svn.advanced.externals.html .

[20] Meliá, G. Cancio, et al., "Current Status of Fabric Management at CERN," *Computing in High Energy and Nuclear Physics (CHEP) Conference*, Interlaken, Switzerland, September 2004.

[21] Childs, Stephen and B. A. Coghlan, "Integrating Xen with the Quattor Fabric Management System," Michael Alexander and Stephen Childs, editors, *Euro-Par 2007 Workshops (VHPC '07)*, Num. 4854 in LNCS, pp. 214-223, 2007.

[22] *OSCAR – Open Source Cluster Application Resources*, http://oscar.openclustergroup.org .

[23] *Rocks Clusters*, 2008, http://www.rocksclusters.org/ .

[24] *Windows Server 2003 Active Directory*, 2008, http://www.microsoft.com/windowsserver2003/technologies/directory/activedirectory/default.mspx .

[25] Poznański, Piotr, *Framework for Managing Grid-enabled Large Scale Computing Fabrics*, Ph.D thesis, AGH University of Science and Technology Krakow, Poland – Faculty of Electrical Engineering, Automatics, Computer Science and Electronics, 2005.

[26] Delaet, Thomas and Wouter Joosen, *Survey of Configuration Management Tools* Technical Report, Katholieke Universiteit Leuven, Department of Computer Science, April, 2007.

[27] Burgess, M., "Cfengine: A Site Configuration Engine," *USENIX Computing systems*, Vol. 8, Num. 3, 1995.

[28] *Puppet*, http://reductivelabs.com/trac/puppet .

[29] Delaet, Thomas and Wouter Joosen, "PoDIM: A Language for High-level Configuration Management," *Proceedings of the 21st Large Installation System Administration Conference (LISA '07)*, USENIX Association, 2007.

[30] Delaet, Thomas, Paul Anderson, and Wouter Joosen, "Managing Real-World System Configurations with Constraints," Technical Report, Katholieke Universiteit Leuven, Department of Computer Science, 2007.