System Administration and CDDLM

by Paul Anderson <dcspaul@inf.ed.ac.uk>
Edmund Smith <esmith4@inf.ed.ac.uk>
School of Informatics
University of Edinburgh

1 Overview

System administration takes a collection of resources and uses them to provide services. For as long as there have been computing resources, there has been systems administration, but the methods used and the services provided have evolved as systems have. At present, a management domain may include thousands of machines, but will not typically overlap organisational boundaries. Similarly, a domain may present hundreds of services, but the rate at which services are changed is very low. The emergence of grid computing means this is unlikely to be true in the future, and the development of new techniques and technologies will be necessary.

CDDLM and the solutions it presents have attracted some interest in the systems administration community, since this is the first high profile example of an attempt at solutions to these new problems. In particular, CDDLM is the first attempt that we are aware of to focus on the management of inter-organisational resources (the management of machines owned by other parties and exposed by a grid).

This paper presents our impressions of the solutions developed by CDDLM in the light both of recent advances in systems administration, and of the many decades of experience of the systems administration community in managing resources. We also briefly examine the relationship between CDDLM and our current project investigating techniques for managing grid fabrics (see section 5).

2 Introduction

The administration of current systems involves configuration at every level, from physical wires to user applications. Tools for managing today's resources typically work from the level of bare metal up to user facing services and applications, including many low level services which provide necessary infrastructure. The provision of these low level services is never a goal, only a result of the need to supply services to users.

For an administrator working with inter-organisational resources, many of their infrastructure services must be expected to be adopted from the organisation maintaining them. We suspect it is considerations like these which have lead to CDDLM's focus upon application deployment (the very highest level).

A similar approach has been taken in the management of many current installations, where instead of "adopting" foreign infrastructure, the infrastructure is maintained manually, and a configuration tool used only to manage software packages. Systems like these, however, have proven vulnerable to failure, especially when there are complex interactions between services in each domain.

One of the requirements for a fault tolerant service is the ability to adapt resources to its provision when failures occur. When the service's configuration is co-dependent upon the configuration of the infrastructure (for example services visible through firewalls, services whose host must be registered in DNS) it is necessary for the tool performing the adaptation to be aware of this co-dependent configuration. We argue that this means that even though CDDLM is unlikely to perform full-scale infrastructure management, it will be necessary to represent (at least) partial configurations of low-level services.

3 What is system configuration?

Whether it is providing grid services, or financial workstations, a modern computing installation consists of a highly complex network of hardware and interconnected services. The *fabric configuration* problem[5] involves taking the following raw materials:

- A collection of *bare metal hardware*.
- A *repository* of software.
- A *specification* of the required behaviour of the overall system.

and turning these into a working *fabric* which reliably and correctly implements the given specification. Moreover, it is necessary to maintain this correspondence with all of the above items in a state of continual evolution.

Over the past 15 years, the complexity of real fabric configuration problems has increased considerably, and the process has evolved from a largely manual operation, to one involving a high degree of automation (see [6] for an overview of techniques). Initially,

skilled system administrators would hand-configure individual *nodes* for particular roles, manually planning the relationships between them, without any formal translation between the overall requirements and the individual configurations. Current state-of-the-art systems now deal with higher-level aspects of *services* which span entire fabrics, automatically translating service requirements into configuration parameters for the individual nodes. However, there is still a large gap between the high-level behavioural requirement specification, and the input parameters for such a system; closing this gap is one aspect of current fabric configuration research.

LCFG[2, 7] is a good example of a current configuration tool. It has been in (evolving) production use for over ten years and was the standard configuration tool for development of the EDG[1] (European DataGrid) testbeds[1]. Appendix A provides an overview of the LCFG architecture and notes some of the current problems.

Typical fabric services include Kerberos (authentication), NFS (file service), LDAP/DNS/NIS (name services), NTP (time synchronisation), SMTP/IMAP (mail), printing, web services, authorisation, etc. An LCFG-managed fabric might have in the order of 1000 nodes, with each node having about 5000 configuration parameters, controlling about 50 *components* (software subsystems)[2].

More important than scale however, is (a) the continual rate of change in configurations (see 4.3), and (b) the difficulty of managing and structuring the configuration specifications when many different people need to control different "aspects" of the overall fabric (see 4.2).

We believe that current fabric configuration development is inhibited by the lack of a common framework, or standard language to describe configurations. Hence the interest in CDDLM. However, we also recognise that some important problems are still open research areas, and any such framework would need to be sufficiently flexible to incorporate likely developments.

4 Experiences of practical configuration

There are many configuration tools for managing today's resources, however we believe there exists a general consensus in the system administration community at present that none of these tools is sufficient to meet the demands placed upon it. While we acknowledge that the CDDLM technologies are aimed at a different space (the types of resources that are expected to be mainstream in the future), we believe there is some significant overlap with current systems.

We have therefore taken this section to present what we believe are difficult problems in systems administration at present which will impact upon the effectiveness of CDDLM's approach.

- In section 4.1 we discuss the difficulty of providing autonomic behaviours without allowing a system's configuration to drift.
- In section 4.2 we present some of the problems of federated management
- In section 4.3 we review some experiences on the inevitability of configuration evolution.
- Section 4.4 discusses the difficulty in managing the security of resources in real world situations.
- Finally, section 4.5 examines the lessons that have been learned about the impact of tool complexity both upon its effectiveness and upon its adoption.

4.1 Providing autonomic behaviour

The CDDLM charter indicates that responsibility will be taken for providing autonomic behaviour for managed applications ("liveness and redeployment"). This is a significant challenge, since autonomics on any large scale is an unsolved problem. In this section we hope to outline some of the problems that have been experienced in attempting to provide autonomics in previous system configuration tools, and some of the current thinking in this area.

Previous attempts at autonomics fall into two broad categories:

1. Blind autonomics performs redeployment and migration upon the detection of failure, without involving the fabric management system. This has the advantage of not corrupting the intended state of the system, and the disadvantage that the specification no longer represents anything like the reality.
2. The tested alternative to blind autonomics is what we will call "procedural" autonomics, where the specification itself is changed in the event of failure and the configuration system itself reconfigures and redeploys. This leads to a specification drift: it can be hard to explain a given specification, the administrator's intended specification is

[1] Some sites now use the new tool Quattor[3], which offers a similar level of functionality, and is based on a similar architecture

[2] The type of components available, and the wide range of parameters can be seen from the LCFG Guide[4].

lost, and any return to the desired state after the failure is repaired is generally impossible (without manual intervention and a knowledge of what the previous state was).

The current speculation is that the only way to effectively implement autonomics is with network level specification. That is, the desired service is specified at a network level and the configuration system acts to maintain that service. This means that theres no need to update the specification (it just says the network has the service), which avoids configuration corruption.

4.2 Conflicting configuration requirements

It is a frequent occurrence in the configuration of a system of any complexity that multiple conflicting requirements are placed upon some aspect of the configuration. To some extent, this is inevitable when complex systems are being configured by many different people. However, most current tools handle conflicts very badly:

In some tools (such as cfengine) conflicts lead to oscillations between configuration states. In LCFG, conflicts are deterministically resolved, but only on the order in which they occur (last specified "wins") which makes for a great deal of source file gymnastics as users attempt to assert a particular value.

In the current CDDLM implementation (as we understand it), there is no particular conflict resolution mechanism, which means an individual service might oscillate, might be determined by the most recently requested value, the first specified value, or some other (as yet untried) system. The major arguments against this approach are usability, maintainability.

Usability and maintainability are discussed as separate problems in their own right elsewhere, but in this specific context it is worth noting that none of the above resolution strategies are good; in many cases they yield a configuration which does not satisfy the intent of any of the authors, and may even be indeterminate!

Current thinking suggests that many conflicts can be avoided by providing language constructs which allow the specification of much "looser" constraints, rather than demanding explicit values (this is related to the requirements for autonomics discussed above). In other cases, the outcome should be deterministic, and obvious – for example by the specification of explicit priorities.

4.3 Configuration evolution

CDDLM appears to lean towards the idea of a "perfect deployment", where a system configuration is described perfectly enough to remain unchanged throughout its lifetime. Traditional fabric configuration tools tend to take a *convergent* approach (see [9]) where the lifetime is assumed to be indefinite, and the system is in a state of continuous reconfiguration, attempting to track the evolving specification. In large systems, the phenomenon of "asymptotic configuration" is typical – the configuration requirements actually change before the entire system has reached the previously specified configuration state!

We would argue that CDDLM must consider long-lived services, since many potential CDDLM applications may also run for weeks, months or years. Experiences of general system configuration tell us that systems do not remain static over these time frames; they are often tuned or managed in various ways which do not require them to be shut down or redeployed. While it can be argued that CDDLM does not *prevent* a configuration from changing without redeployment, it does not appear at present to enable this to be done cleanly or in a standard way.

4.4 Managing configuration security

Much of the discussion about security in the context of grid applications and CDDLM is focused around authentication. Most sites have a well-developed user authentication infrastructure internally which a configuration tool may rely upon.

There are two major security problems for existing tools which will also be important for CDDLM:

- Configuration authorisation: managing the aspects of a system's configuration a particular user is authorised to effect.
- Machine authentication: when configuration information is automatically generated, it becomes difficult to authenticate.

In terms of authorisation, we note that some CDDLM-deployed services will need to be (partially) configured by multiple sources (this becomes inevitable as the scale of deployments increases). Furthermore, CDDLM will need consider the sharing of adopted fabric services (see section 2) which will necessitate a robust authorisation approach.

Tools at present have only been able to distinguish between the authority to manage a machine or not. Finer grained options (allowing particular administrators only to manage specific aspects for example) have

(to the authors' knowledge) never been successfully applied in a production system.

Authentication of generated configuration information is an area of active interest at present as current generation tools move towards greater automation. We note that this has been recognised as a difficulty by the CDDLM group, but we are unclear on the current approach. We are not aware of any current tool presenting a viable solution to this problem.

4.5 Complexity and usability

One issue which has become apparent comparatively recently is the problem of usability and complexity of configuration tools. There has been clear feedback from the system administration community that the learning curve is a major barrier to adoption for most of the current tools which attempt to address the overall system configuration problem.

Moreover, a fundamental aim of such tools is to generate a reliable and correct system configuration, and it is essential to include the human element in this process: misunderstandings and unpredictable behaviour of the configuration tool is likely to be a source of many more configuration errors, than technical problems with the tool itself.

We are concerned about the complexity of the CDDLM proposals in the light of these current issues. While we acknowledge much of the perceived difficulty stems from the incompleteness of documentation at this time, we still suggest the overall complexity may be too high.

5 OGSAConfig

OGSAConfig is a JISC funded project investigating dynamic reconfiguration for grid fabrics. As part of our effort to support standardisation in the area, we investigated using the CDDLM proposed interfaces.

Dynamic fabric reconfiguration is a similar problem to that addressed by CDDLM, although it is aimed at grids of a different era. The applications we are most anxious to support are those deployed on current e-science grids. Since many of these applications cannot co-exist, our problem appears very similar to that of CDDLM: describe and deploy an application for consumption by users, then undeploy it.

We have diverged from CDDLM in several ways that are worth discussing. Firstly, the need for centralised configuration management (most major tools take this approach, and it was beyond the scope of our project to develop a new tool) meant that we separated the act of describing a potential configuration to the fabric from the act of deploying it. This allows us to describe it once to a fabric "broker", then deploy it independently at individual nodes.

We found this structure very difficult to reconcile with CDDLM which has both a host oriented bias (it appears a CDDLM service is intended to represent a portal to the configuration of individual hosts) and an emphasis on deployment as description. These considerations and trade-offs will be discussed in some detail in the forthcoming OGSAConfig architecture report [8].

6 Conclusions

CDDLM is one of the first standards efforts in the area of systems administration. It focusses on the management of some particular resource types which are not yet common, and for which standard configuration interfaces will be vital. It seems clear to us that there are many similarities between these resources and the currently prevailing types, and that there is the potential to learn from the problems experienced by tools in present use.

As grids become more common, and services and systems migrate outward to third-party providers, the need for standards like CDDLM will grow. However, for the forseeable future these "on-demand" resources will need to interact with the more traditional kind, which will make bridging between existing tools and CDDLM important. We suggest it is important to consider in advance how such bridging might be achieved, especially given the significant difference in structure between the CDDLM solutions and such tools.

In section 4 we looked at some existing system configuration problems which we believe will impact CDDLM. We believe the danger for CDDLM lies in attempting to establish a standard for problems whose solution is not well understood. This might be alleviated by avoiding mandating unnecessary detail, whose inclusion could prevent future solutions from fitting within the framework.

7 Acknowledgements

OGSAConfig is funded by a grant from the Joint Information Systems Committee (JISC)[3].

Some travel for this research has been funded by the GridNet programme [4].

[3] http://www.jisc.ac.uk
[4] http://www.nesc.ac.uk/esi/gridnet.html

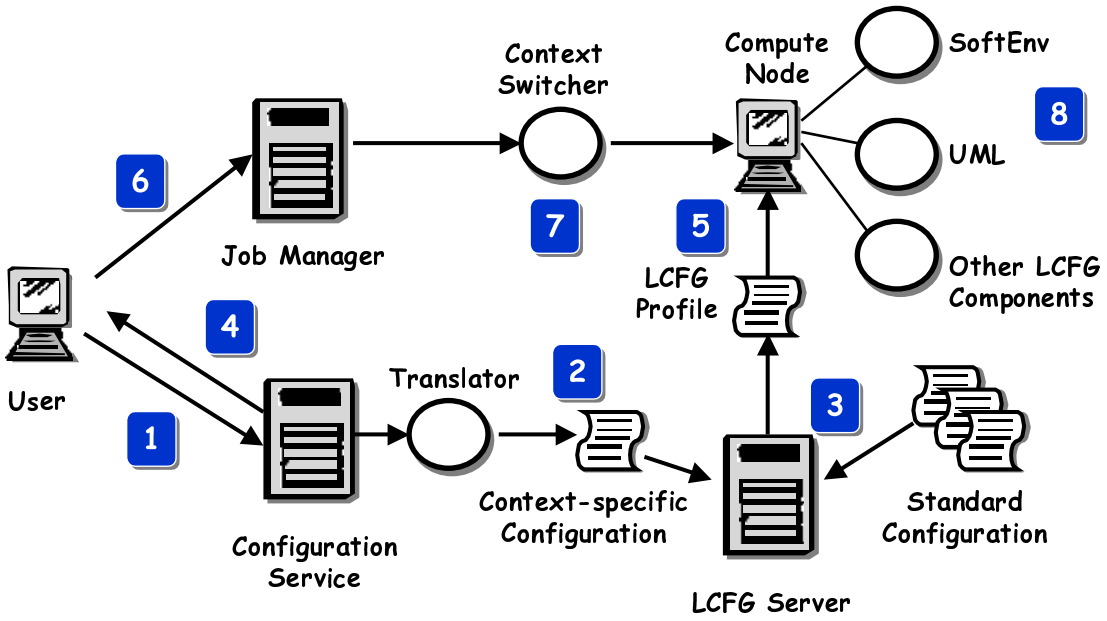

Figure 1: OGSAConfig Architecture

Appendix A LCFG

The LCFG architecture is shown in figure 2:

- The configuration of the entire fabric is described in *source files* held on a master server. These files do *not* correspond to individual nodes, or even individual services; rather, they correspond to different *aspects* of the entire fabric. Frequently, these will correspond to individual node roles, such as a “laptop” or a “student machine”, or an “public web server”.
- A *compiler* on the master server *composes* the source files affecting each node to generate a per-node *profile*. This profile contains all the configuration information necessary to recreate the entire node from the bare metal hardware, and the software repository.
- When a profile changes (due to a source file change), the server notifies the affected client[5], and the client fetches the new profile using HTTP.
- The configuration parameters (*resources*) in the profile are interpreted by a (configurable) set of components on the client machine, each of which translates these into the low-level configuration files and options required for a particular subsystem (for example, mail server, or authentication).

Several points are worth noting:

- LCFG mandates a centralised architecture; all configuration specifications must be channelled via the master server, even if they originate elsewhere. We believe that this is a problem, both for scalability, reliability. More seriously, it requires a single, authoritative source of all configuration information which is not possible in a dynamic and/or federated environment.
- Configurations of individuals nodes are automatically generated by composing various aspects. The aspects may involve inter-node interactions; for example, configuring one node as a firewall, and an another as a public web server, should create a profile for the firewall which includes a “hole” for the web server.
- There is no explicit deployment lifecycle. The LCFG components are responsible for deploying, undeploying and dynamically reconfiguring services at the node level to ensure that they conform to the specification. Exactly the same principle is even used to install new nodes by trans-

[5]The client also polls periodically for changes in case it is unavailable at the time the notification occurs.

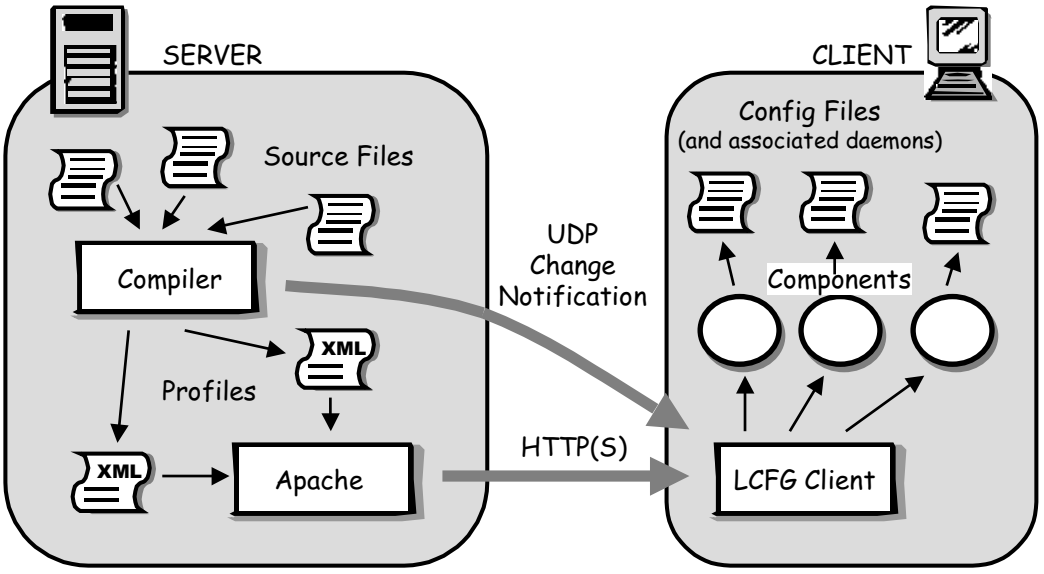

Figure 2: LCFG Architecture

forming the empty disk into one containing the required software. Note that any synchronisation of deployment or reconfiguration of inter-node services requires communication via the master server, or the use of some out-of-band technique. This can be a problem when deploying or reconfiguring complex inter-node services.

References

[1] The European DataGRID. web page. http://web.datagrid.cnr.it/.

[2] LCFG. web page. http://www.lcfg.org/.

[3] Quattor. web page. http://www.quattor.org/.

[4] Paul Anderson. The complete guide to LCFG. Technical report. http://www.lcfg.org/doc/guide.pdf.

[5] Paul Anderson. What is this thing called ”configuration”? LISA Large Scale Configuration Workshop, October 2003. http://homepages.inf.ed.ac.uk/ ... dcspaul/publications/config2003.pdf.

[6] Paul Anderson, George Beckett, Kostas Kavoussanakis, Guillaume Mecheneau, and Peter Toft. Technologies for large-scale configuration management. Technical report, The GridWeaver Project, December 2002. http://www.gridweaver.org/WP1/ ... report1.pdf.

[7] Paul Anderson and Alastair Scobie. LCFG - the Next Generation. In *UKUUG Winter Conference*. UKUUG, 2002. http://www.lcfg.org/doc/ ... ukuug2002.pdf.

[8] Paul Anderson and Edmund Smith. A dynamic configuration architecture. Technical report, The OGSAConfig project, October 2004. http://groups.inf.ed.ac.uk/ ... ogsaconfig/papers/report2.pdf.

[9] Mark Burgess. Automated system administration with feedback regulation. *Software-Practice and Experience*, 28, 1998. http://www.iu.hioslo.no/~mark/ ... research/feedback/feedback.html.