

# Experiences and Challenges of Large-Scale System Configuration

*Paul Anderson <paul@dcs.ed.ac.uk>  
George Beckett <g.beckett@epcc.ed.ac.uk>  
Kostas Kavoussanakis <k.kavoussanakis@epcc.ed.ac.uk>  
Guillaume Mecheneau <guillaume.mecheneau@hp.com>  
Jim Paterson <jpaters2@inf.ed.ac.uk>  
Peter Toft <peter.toft@hp.com>*

## Abstract

This report identifies the large-scale fabric configuration requirements of the emerging Grid world. We present a number of real-world case studies to illustrate how several different organizations currently manage large-scale system fabrics. We then examine some projected use cases that aim to predict the way in which large-scale system configuration will need to evolve in order to meet the challenges of tomorrow's fabrics. We conclude the report with a summary of key research challenges.

The logo for GridWeaver features the text "GridWeaver" in a blue, sans-serif font. A light blue, wavy line arches over the text, starting from the left, passing over the "i" and "d", and ending under the "r".

GridWeaver

**Revision 1.0 – March 26, 2003**



# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
<b>2</b>	<b>Case Studies</b>	<b>9</b>
2.1	Study 1: CERN: A Large Grid Fabric . . . . .	10
2.1.1	Background . . . . .	10
2.1.2	Configuration . . . . .	10
2.1.3	Problems . . . . .	10
2.2	Study 2: Argonne National Labs: A Large Research Cluster . . . . .	12
2.2.1	Background . . . . .	12
2.2.2	Configuration . . . . .	12
2.2.3	Problems . . . . .	13
2.3	Study 3: A Government Research Lab . . . . .	14
2.3.1	Background . . . . .	14
2.3.2	Configuration . . . . .	14
2.3.3	Problems . . . . .	14
2.4	Study 4: A Peripheral Manufacturer . . . . .	16
2.4.1	Background . . . . .	16
2.4.2	Configuration . . . . .	16
2.4.3	Problems . . . . .	16
2.5	Study 5: Astronomy at the University of Chicago: A Small Academic Department . . . . .	18
2.5.1	Background . . . . .	18
2.5.2	Configuration . . . . .	18
2.5.3	Problems . . . . .	18

2.6	Study 6: Informatics at the University of Edinburgh: A Large Academic Department . . . . .	20
2.6.1	Background . . . . .	20
2.6.2	Configuration . . . . .	20
2.6.3	Problems . . . . .	21
2.6.4	Summary . . . . .	22
2.7	Study 7: The European Bioinformatics Institute: A Research Computing Facility . . . . .	23
2.7.1	Background . . . . .	23
2.7.2	Configuration . . . . .	23
2.7.3	Problems . . . . .	23
2.8	Summary . . . . .	25
2.8.1	Installation versus configuration of nodes . . . . .	26
2.8.2	Separation of server and client node configuration . . . . .	27
2.8.3	Manual sequencing of configuration change . . . . .	27
2.8.4	Dependencies on administrative staff . . . . .	27
2.8.5	Monitoring and simulation of configuration . . . . .	28
<b>3</b>	<b>Projected Use Cases</b>	<b>29</b>
3.1	Projected Use Case 1: A Large Academic Department . . . . .	30
3.1.1	Background . . . . .	30
3.1.2	Configuration . . . . .	30
3.1.3	Challenges . . . . .	30
3.2	Projected Use Case 2: A Remote Rendering Utility . . . . .	32
3.2.1	Background . . . . .	32
3.2.2	Configuration . . . . .	32
3.2.3	Problems . . . . .	32
3.3	Projected Use Case 3: A Computing Utility Providing Infrastructure On Demand . . . . .	34
3.3.1	Background . . . . .	34
3.3.2	Configuration . . . . .	34
3.3.3	Problems . . . . .	34
3.4	Projected Use Case 4: Interacting Grid Services . . . . .	36

---

3.4.1	Background . . . . .	36
3.4.2	Configuration . . . . .	36
3.4.3	Problems . . . . .	36
3.5	Summary . . . . .	38
3.5.1	Multi-level abstraction of configuration description . . . . .	38
3.5.2	Devolution of management and composition of configuration . . . . .	39
3.5.3	Security . . . . .	40
3.5.4	Highly dynamic fabric configuration . . . . .	40
3.5.5	Monitoring and failure recovery . . . . .	41
3.5.6	Massive scale configuration . . . . .	41
<b>4</b>	<b>Conclusions</b>	<b>43</b>
4.1	Preamble: A Paradigm Shift for Computing . . . . .	43
4.2	Research Challenges . . . . .	45
4.2.1	Representing system configurations . . . . .	45
4.2.2	Using and manipulating system configurations . . . . .	46
4.2.3	Deploying system configurations . . . . .	47
4.3	GridWeaver Research Focus . . . . .	48
<b>5</b>	<b>Appendix: Case Study Questions</b>	<b>51</b>
5.1	Physical composition and scale of the fabric . . . . .	51
5.1.1	Background . . . . .	51
5.1.2	Scale . . . . .	51
5.1.3	Diversity . . . . .	51
5.1.4	Dynamism . . . . .	52
5.2	Current configuration management tools used . . . . .	52
5.3	Current configuration management strategy . . . . .	52
5.3.1	Scope of configuration . . . . .	52
5.3.2	Delegation of tasks . . . . .	53
5.3.3	Tasks which require manual intervention . . . . .	53
5.3.4	Monitoring and problem detection . . . . .	53
5.4	Future plans and problems envisaged for the future . . . . .	53
5.5	Wishes for future configuration management tools . . . . .	54



# Chapter 1

## Introduction

This report is the second in a series of deliverables from the *GridWeaver* project [Edi], a collaboration between Edinburgh University School of Informatics, HP Labs in Bristol, and EPCC, as part of the UK e-Science Core Programme. GridWeaver is focused on the challenges of managing the configuration of very large-scale computational infrastructures, especially those that form parts of computing Grids. Our goal is to develop approaches and technologies that are capable of representing and realising complex system configurations of hardware, software and services, across very large-scale infrastructures incorporating high degrees of heterogeneity, dynamism and devolved control.

In our first report, *Technologies for Large Scale System Configuration Management* [ABK<sup>+</sup>02], we discussed the basic approaches in use for configuring and managing large fabrics, and examined a number of specific system configuration technologies. We concluded that there is no complete tool to tackle all aspects of fabric management. The technologies available from the consortium were known to address parts of the problem, but they were found to be complementary and thus have a lot of potential.

In this report, we aim to identify the fabric requirements of the emerging Grid world. We first present a number of real world case studies to illustrate how several different organizations manage large-scale system fabrics. We discuss the nature and purpose of each fabric, how system configuration tasks are staffed, and what combination of technologies are in use. We examine how well these technologies are currently meeting the requirements placed upon them, and look for any limits being encountered now or in the future. Our effort extends to an investigation of the fabric managers' thinking, in addition to the shortcomings of the tools and approaches currently available to them.

Secondly, we suggest some *Projected Use Cases* that aim to predict the way in which large-scale system configuration will need to evolve in order to meet the challenges of tomorrow's fabrics. This has a dual intent: to be thought-provoking and to test whether our predictions are shared by others; and to illuminate what we think are the key research and engineering problems that need to be solved to meet the needs of tomorrow's fabrics.

Finally, we summarise our views on the key research challenges facing large-scale system configuration, set these in the context of a paradigm shift that is taking place in the way we think about computing infrastructure, and indicate the research areas that the GridWeaver project will focus on.

## Acknowledgements

Thanks to the following people for providing case studies: Narayan Desai of Argonne National Laboratories, Tim Smith of CERN, John Valdes of the University of Chicago, Alastair Scobie and George Ross of the University of Edinburgh, and those others who donated their time anonymously.

The report is a deliverable of the *GridWeaver* project, a collaboration between the University of Edinburgh School of Informatics, HP Labs in Bristol, and EPCC.

GridWeaver is part of the UK e-Science Core Programme, and we gratefully acknowledge the programme's assistance with this activity. We also wish to thank the other members of the GridWeaver team – Carwyn Edwards, Patrick Goldsack, John Hawkins – for their contributions to the project as well as Alexander Holt for his valuable input.

For administrative purposes, GridWeaver is also known under the project name *HPFab-Man*.

## Chapter 2

# Case Studies

In the first GridWeaver report [ABK<sup>+</sup>02], we examined a representative sample of the current state-of-the-art in automated configuration management technologies. We concluded that the field is populated by a considerable number of packages which either specialise in resolving specific challenges facing fabric configuration (such as installation of a new node), or provide a suite of inter-related utilities to implement a more wide-reaching solution. However, in spite of the array of tools available, it is our opinion that the key problems of configuration experienced in today's fabrics have yet to be satisfactorily resolved.

To justify this opinion, in this chapter we present six case studies from a carefully selected set of real institutions, chosen in light of their fundamental reliance on effective configuration management. The examples cover computational clusters, research and development centres, and academic departments. They provide a representative sample of the experiences from the more general community. Note that some interviewees requested that we anonymise the studies, a request we could only honour.

In each case study we establish the background to the current installation, assess the size and extent of the fabric in place, and review the functionality and flexibility required by its users. We then reprise the technologies that are utilised to implement configuration management and ascertain the advantages which these technologies have furnished. We also ask our interviewees to summarise the problems that currently face their particular environment, and to speculate on how these difficulties would be affected by future expansion and development of the institution.

Based on these studies, we aim to confirm the opinion stated above, identify common threads of concern raised by the different partners, and set challenges and aspirations for future configuration technology innovations.

## 2.1 Study 1: CERN: A Large Grid Fabric

### 2.1.1 Background

The computing facilities at CERN [cera] include a large cluster which is used for experimental physics applications. This is highly typical of the resources which are expected to form part of scientific computing GRIDs. The cluster consists of 750, dual-processor compute nodes; about 200 disk servers; 70 interactive machines; and a smaller number of assorted servers. 20-30 different base-hardware types are represented.

The cluster runs a uniform environment based on the Linux operating system. The attraction of the very large compute power has been sufficient to persuade users to relinquish the individually customized environments that they had previously considered necessary. Applications supply their own software which is mounted via AFS, or staged onto local disks.

The cluster runs as a 24x7 operation with 4-5 FTEs of support. However, initial installation of new machines and standard maintenance operations are all outsourced.

### 2.1.2 Configuration

Initial installs are performed using a network boot and a Kickstart [Ham] script which is automatically generated by local software to account for the hardware differences. Despite bulk purchases, the hardware variety is too great for image based installs to be practical.

Configurations are maintained by SUE [CERb] features which synchronize the nodes against a master AFS server each night.

Only the compute nodes are managed automatically. Servers (for example, AFS servers, tape servers, DNS servers, etc.) currently require manual configuration by one of a small number of experts. Significant changes in the overall fabric configuration require carefully scheduled downtime.

The future intention is to replace the current configuration procedures with the EDG fabric management software [WP4] as this becomes available. It is hoped that this will alleviate most of the current problems.

### 2.1.3 Problems

The following current problems were identified:

- There are multiple sources of configuration information, in different formats, with the associated potential for errors and inconsistency.
- There is a lack of connection between the configuration of related clients and servers. In some cases, this requires careful manual co-ordination of data managed by different administrative domains.

- The SUE features do not provide a complete synchronization of the entire system. This leads to differences between nodes installed at different times.
- Uncoordinated changes to the master server by different people can result in non-functional configurations being shipped to all nodes.
- The fabric frequently has machines in different configuration states since migration from one state to another must be synchronized with the requirements of the running jobs. Scheduling this manually is difficult.
- There is a lack of a good GUI interface for performing high-level configuration operations on the fabric.
- There is a lack of fine-grained access control to permit devolved management of certain aspects of the fabric.
- There is a lack of adequate monitoring to confirm the actual configuration state of nodes before allowing them to be used for applications.

The following additional problems were also envisaged should the fabric size grow significantly:

- Manual management of servers would become untenable and a more flexible configuration system would be necessary to support the diversity required for automatic server configuration.
- The manual coordination of related changes to servers and their associated clients would become even more difficult. Some automatic way of sequencing these changes would be "hugely beneficial".

## 2.2 Study 2: Argonne National Labs: A Large Research Cluster

### 2.2.1 Background

The Argonne National Laboratory [arg] at the University of Chicago runs a development facility for researchers working on cluster computing. This comprises of about 1100 nodes, 300 of which form the experimental cluster known as “Chiba City” [Eva01]. The remaining machines are desktop workstations, which are mostly running Linux, although Solaris, AIX, SGI and MS Windows are also represented.

14 system staff are involved in managing this facility, although these staff provide a wide range of services from low-level network support, to diverse application support (not including MS Windows).

### 2.2.2 Configuration

Identical tools are now used to manage both the cluster, and the desktops:

Vendor-based tools, such as Jumpstart [Mic97] and Kickstart [Ham] are normally used for initial installation. However, since this is an experimental cluster, users sometimes have a need to install highly specific specialized images on the cluster, and this is performed by an imaging process.

A local tool called “Sanity” customizes and maintains the installed configurations. Sanity consists of a number of Perl modules which read configuration files from a central repository and modify the node configuration appropriately (one Sanity module is responsible for managing the packages installed on a node). A central database defines a set of tags for each node which specify the base configuration file, and a set of files containing deltas which modify lines in the base configuration. Sanity is run regularly from cron, or on demand.

Gradually, those configuration operations which used to be performed manually have been incorporated into the Sanity modules, and it is now possible to rebuild most nodes completely automatically, although some servers still require a small amount of manual work, and laptops are not currently supported. In general, the system staff are happy that the site-wide configuration is now largely represented in a single central repository.

A basic monitoring framework runs on the cluster and records the system health in a database. The tests for this have been developed as problems have been identified. There is no automatic feedback of the monitoring for error correction, and no monitoring on the workstations.

The system staff believe that the basic concepts of the configuration system are sound and would scale to a ten-fold increase in cluster size. However, there would be some concern about a ten-fold increase in the number of servers or workstations; this concern is related to problems of structuring configuration information to support the necessary diversity, as discussed below.

### 2.2.3 Problems

The following current concerns were identified:

- “Reading the repository” to determine the current configuration of a node is not always easy. This involves a knowledge of the database tags, and the contents of the configuration deltas. This is especially difficult where more than one person is involved in the configuration specification.

It is not clear at this stage whether the problem can be largely solved by restructuring of the configuration files and a better user-interface, or whether there are deeper representation issues.

- There is a need to abstract the base operating system and support independent “aspects” so that, for example, an operating system can be upgraded while maintaining the configurations of aspects such as “the web server” or “a visualisation node”.
- System administrators are often uncomfortable using a powerful and complex tool to make potentially far-reaching changes to the entire site configuration. They would rather be able to “reverse-engineer” a hand-crafted configuration to generate a high-level specification which could be incorporated into the tool.
- Similarly, it would be useful to have simulation tools to allow “what-if” experiments on the entire site configuration.

## 2.3 Study 3: A Government Research Lab

### 2.3.1 Background

The research and development facility of this Government organization includes around 250 Unix machines used by scientists, developers and students. The population is roughly 85% Linux and 15% Solaris. Four Linux servers provide the usual services, including email, web and a small Beowulf cluster.

The Unix installation is managed by a team of three system administrators, although networking and management of the MS Windows machines are provided by the central IT department.

### 2.3.2 Configuration

Vendor tools such as Kickstart[Ham] and Jumpstart[Mic97] are used for initial installation. Cfengine[Bur95] is used post-install to configure all machines for the local environment. Within one operating system release, workstation configurations tend to be quite uniform, and the number of simultaneous operating system releases is deliberately kept small.

Ongoing configurations are maintained using cfengine and this handles the diversity of servers by using cfengine classes for the various different services. Third party application software is NFS-mounted from central file-servers. Cfengine handles local software updates using the native package format for each Unix flavour. The master node for the Beowulf cluster is managed in the same way as any other server, and the cluster nodes are created using an imaging tool.

A simple monitoring system attempts to connect to services at regular intervals and raises an alarm if there is no response. There is no feedback of the monitoring and this is not used for error correction.

Most desktop and server configuration is under reasonably automatic control and this makes it easier for the small number of system staff to run an effective fabric.

### 2.3.3 Problems

- It is difficult to contain the need for diversity, and there is a need to support more diverse configurations such as isolated remote machines, and laptops, as well as more diverse software such as operating system releases;
- Operating system upgrades are a problem because of the need to review/modify the cfengine scripts;
- The complexity of the cfengine scripts would be likely to become unmanageable if there was a significant increase in diversity of the site. The configuration informa-

tion is currently structured in terms of aspects such as "services" (rather than structured by platform/release), but a richer method of abstraction/classing is needed;

- It is difficult for people outside (or new members of staff) to understand how the entire system works. This includes both the operation of the tools, and the policy/reasons behind their use. Current tools don't provide a way to easily identify how customized configurations differ from the default.

## 2.4 Study 4: A Peripheral Manufacturer

### 2.4.1 Background

This study describes the computing infrastructure in the development laboratory of an international peripheral manufacturer. This computing facility supports about 150 engineers working on embedded software development, hardware design, and device-driver construction.

Currently, the Unix hardware base of about 200 nodes is mostly Solaris, although this is rapidly evolving towards Redhat Linux. A further 250 PCs run MS Windows for administrative use, as well as platform-specific driver development. (Note that most users have multiple machines). A central cluster of 50 machines is used for dedicated compilations (a nightly rebuild of the codebase), but this does not run any formal clustering software.

Current staff include only three people dedicated to system support, and this is considered seriously inadequate (and poorly-understood by management). These staff manage all the infrastructure, including networking and user applications, except that the Windows PCs are co-managed with the corporate IT organisation.

### 2.4.2 Configuration

Initial installs are performed using vendor tools such as Jumpstart [Mic97] and Kickstart [Ham]. These load a package which overlays the base configuration files with local configuration information. A wrapper script supports some degree of diversity by loading different package sets for each of a small number of different node classes. Otherwise, an attempt is made to keep configurations as uniform as possible. Servers require additional manual configuration which is logged and documented, but not automated.

There is no coordinated attempt to apply ongoing configuration changes; nodes are completely rebuilt (and upgraded) at convenient times for individual developers (for example, between projects). This provides a stable environment for the duration of a project. Essential updates, such as security patches, are applied manually, but these are not frequent. Future plans include the ability to apply critical patches under Linux by updating RPMs automatically from cron.

A monitoring script runs nightly to harvest log files and attempts some simple aggregation to detect common errors. There is no feedback of monitoring for automatic fault recovery.

The migration from Solaris to Linux is currently consuming most of the available effort, and there are no immediate plans for major changes to the configuration technology.

### 2.4.3 Problems

- Installation of workstations is considered adequate, but reconstructing servers is difficult and there is a reliance on hardware techniques such as RAID to minimise the need for this.

- Ongoing configuration maintenance is expensive. It is difficult to decide for each new patch, whether it is worth the effort of manually applying it to all the nodes.
- Operating system upgrades are a major problem.
- The small number of staff and the lack of automation makes the site very vulnerable to the loss of critical staff.
- Improved monitoring is necessary to detect failed or misconfigured systems.
- The small number of staff does not permit the development work necessary to learn and adopt more sophisticated tools which may improve the overall efficiency.

## 2.5 Study 5: Astronomy at the University of Chicago: A Small Academic Department

### 2.5.1 Background

The Department of Astronomy and Astrophysics at the University of Chicago supports about 120 machines, mostly running Linux, although about 25% are Solaris and there are a few MS Windows systems for administrative use. The user community is the usual mix of students and staff engaged in teaching and research.

One system administrator supports the entire installation, including email and web services, although core networking is provided by the central IT services. The user community is generally well-trusted and users often have root access which is rarely abused.

### 2.5.2 Configuration

Kickstart [Ham] and Jumpstart [Mic97] are used for initial installation with cfengine [Bur95] being called from the post-install phase to customise the configuration. Interestingly, RPM is used to install local software on both Linux and Solaris<sup>1</sup> (apart from the core operating system). All additional application packages are created locally from the sources.

The ongoing configurations are maintained by cfengine. Generally the workstation configurations are fairly uniform, although there is a wide range of hardware, and several different operating system versions are in use at any one time, due to the effort required to upgrade the machines. The RPM lists for the machines are stored in a central (mySQL) database, and a local cfengine module installs the RPMs specified in the database onto each machine. A similar process is used for Sun patches.

cfengine is used to perform some basic monitoring checks on each run, and this sends email to the administrator if problems are found. An extra “watchdog” process on a separate machine detects systems which are not responding (i.e. systems which have not run cfengine).

In general, most of the workstations are supported by the automatic configuration system. The servers require some manual configuration effort, but this could be incorporated into the automatic system given time. Even laptops are supported to some degree using the same process.

### 2.5.3 Problems

- The local tool for installing RPMs based on the contents of the database does not currently handle dependency ordering which means that RPM dependency-checking is disabled and there are often problems with dependencies. Extracting dependency information into the database would allow this to be taken into account;

---

<sup>1</sup>RPM is a Redhat Linux tool and is not widely used on Solaris, but it is being used (successfully) in this case to provide a cross-platform package management solution.

## *2.5. Study 5: Astronomy at the University of Chicago: A Small Academic Department*

---

- As would be expected with such a small staff, the time consumed by user-support and one-off problems makes infrastructure development difficult;
- Currently, separate password files are used and there is no directory service, which makes user management very time-consuming. An LDAP service, integrated with the central IT organisation is being considered;
- If there were to be a significant increase in scale, then the complexity of the configuration information would cause problems. This is currently distributed between the cfengine configuration files, the tool modules, and the database;
- The complexity of the system is also a significant barrier to understanding, either by new staff, or by other departments who would welcome the benefits of the technology.

## 2.6 Study 6: Informatics at the University of Edinburgh: A Large Academic Department

### 2.6.1 Background

The School of Informatics (SOI) [inf] at the University of Edinburgh manages a medium-scale fabric containing approximately 800 nodes. It is a multi-site enterprise, distributed between four academic departments that are located at various sites around the city.

The fabric consists mainly of desktop personal computers running Red Hat Linux, though there are a small number of Solaris-based Sun workstations, several Apple Macintosh computers, and a recently installed Beowulf cluster. The fabric is employed both for teaching and research-based activities, and is subject to high frequency (daily) and diverse configuration changes.

Maintenance of the configuration of the fabric utilises the combined resources of approximately 25 technical staff. These people are called upon as issues arise, sharing their time between configuration management tasks and other responsibilities. The configuration environment within the School is called DICE [DIC]. This integrates a collection of technologies, such as LCFG [AS02] (for management of nodes, including server and client workstations and printing facilities), along with other home-grown utilities for configuring and monitoring the integrity and performance of the network infrastructure, and creating/updating user accounts.

### 2.6.2 Configuration

The key configuration tasks undertaken on a regular basis within the school include:

- hardware and software failure recovery;
- installation of new nodes onto the fabric;
- software updates (representing patches or new versions);
- configuration changes and amendments in response to requests from individuals or project groups within the school.

The configuration environment of DICE has been specifically designed to reduce the level of manual involvement required from technical staff to complete such tasks. While it has not been possible to achieve complete automation and, typically, staff are required to schedule and initiate the deployment of configuration tasks, much of the mundane and intricate mechanics of configuration management has been effectively hidden within the DICE environment.

In the school, tasks are delegated to staff based on the evaluation of a number of criteria. A task specific to a locally maintained LCFG component, or other DICE utility, is generally assigned to the particular author. In addition, certain staff have responsibility for specific aspects of configuration or designated localities of the fabric. In fact, the school would

like to take the devolution of management further, allowing individual users to control prescribed aspects of the behaviour of their particular, local machine. This would require a finer grain level of access control than is currently implemented.

Different aspects of configuration are not usually independent and, potentially, have the ability to interact with each other. An administrator must employ caution when scheduling and implementing changes. Typically, before applying a configuration change, the individual must evaluate the scope of effect which it will have, contact and negotiate with affected staff, and agree on a schedule and process for successfully implementing the change. This form of process is most effective and reliable amongst small groups of administrative staff operating from a single site.

Careful consideration and negotiations of this type are very important since, currently, DICE has little provision for the automated detection and resolution of conflicts and inconsistencies which may be present in the description of a node—either at the point of compilation or at run-time. This places responsibility for maintaining the integrity of the system firmly on the shoulders of technical staff.

### 2.6.3 Problems

- The procedural execution of configuration change is not coordinated by LCFG on a pan-nodal scale. This makes enforcing dependencies between individual changes problematic, and prevents any transactional characteristics being inferred from the process.
- With the expansion of the school, it is expected that a greater diversity of hardware components and configuration requirements will be realised in the future<sup>2</sup>. Currently, new types of hardware are accommodated within the fabric by manually editing and extending the existing LCFG header files. However, as hardware diversity grows, it is desirable that this manual task be replaced by some form of systematic automation.
- The expansion within the school is also likely to generate a need for support of a greater range of operating systems. Currently, only Linux-based nodes receive comprehensive coverage within the DICE environment—while a small number of Solaris nodes fall within its jurisdiction, these are managed by an older version of LCFG which is not maintained. At the time of writing, work is in progress to port the technology to Macintosh Computers running Mac OS X.

While adding support for Unix-like environments (such as Mac OS X) to DICE is relatively straightforward, it is also anticipated that some users will desire to have a version of Microsoft Windows on their desktop machine (either as the primary or an auxiliary operating environment). No provision for such systems is currently available and the significant difference in underlying architecture between Microsoft Windows and Unix may make any future provision problematic.

---

<sup>2</sup>This increase in diversity will in part be due to a relaxation of hardware purchasing control, allowing a greater diversity of manufacturers to contribute devices to the fabric.

- At present, a significant problem is attributed to the volume of nodal source file re-compilations which are required. Whenever an LCFG header file is modified, all configuration descriptions which utilise this header file must be re-compiled in order that the changes take effect. At the time of writing, to rebuild all such descriptions takes approximately 5 minutes. Clearly, as the size and dynamism of the fabric grows, this problem will be accentuated.
- The configuration environment within the school is perpetually evolving, and both user and reference documentation are currently inadequate. This, coupled with the unusual and complex nature of the configuration modelling environment, makes it difficult for new members of technical staff to become familiar with the technology, and inhibits the imposition of a controlled operating practice.
- In the future, the current administrator-oriented negotiation and conflict resolution procedure will be strained by an increase in the number of operational staff (distributed between the various sites). In light of this, it is desirable to have a more formal process for scheduling and deploying changes to the fabric and a validation process for highlighting potential conflicts prior to deployment.

#### **2.6.4 Summary**

Within the SOI, the current provision for automating the configuration of the fabric has evolved in response to practical needs. The school has chosen to adopt a home-grown technology, that is constantly under review and development. Key to the success of the current configuration process is simplicity and the ability to delegate responsibility between multiple administrators. The school is also attempting to anticipate future demands and increase the functionality, whilst reducing the volume of manual handling required within the DICE system.

## 2.7 Study 7: The European Bioinformatics Institute: A Research Computing Facility

### 2.7.1 Background

The European Bioinformatics Institute (EBI) supports around 150 Linux nodes used both by internal researchers and to provide web services to the external scientific community. The set of nodes is comprised of around a hundred desktops machines used by researchers, and a cluster of approximately forty servers used for scientific computation. The remaining machines provide the information and analysis services over the web: web serving, running the SQL databases, etc. The Linux installation is managed by a team of two administrators, who developed most of the software used for the configuration.

### 2.7.2 Configuration

The philosophy behind the EBI configuration tool is to minimize the need to be concerned with the differences between nodes. Nodes tend to be based on similar hardware, for similar roles.

Each node configuration (both hardware and software) is derived from a common standard. The software configuration is kept as inclusive as possible: the goal is for every researcher to be able to find most of the software tools they will need in the standard configuration. This policy minimizes the need for specialized configuration. In total around fifty different configurations are managed at the moment, with differences arising mainly from heterogeneous hardware. The farm machines are all identical, but the desktops and the web servers do not all have the same directory structure, network configuration, or display drivers. Desktop machines cannot be configured by the users, but, it is possible for specific software to be installed through scripts, however this is avoided as much as possible.

All machines are booted over the network, the local hard disk is not bootable. A minimal client is uploaded to the machine at boot time to update the local disk according to its intended configuration.

LSF is used to monitor the state of the running machines. If anything unexpected happens, the default action is to reboot the failing node and let it recover its configuration from the network.

### 2.7.3 Problems

As the current tool used for configuration was developed in-house to meet local requirements, it currently mostly fits the needs of this computing facility.

The main problem identified was one of dealing with the hardware diversity. It remains critical for the administrators to:

- thoroughly test the hardware and software configuration they use to define their standard configurations;

- restrict hardware diversity by buying similar hardware as far as possible.

In addition, supporting a significantly larger number (e.g. 10 or 100 times larger) of machines would be very hard without changes to the actual configuration technology. Replacing a failed node is a particular example of an activity not supported very well currently: given the higher likelihood of such an incident in a larger fabric, the process would need to be more streamlined in order to be efficient enough.

Also, at present, the deployment of node configurations is serialized, and trying to reboot or reconfigure a very large number of nodes would take too much time. So, if a script needs to be run in every machine, changes would be required to allow it to run in parallel.

There are also a number of instances in which human intervention is required, where there are dependencies between nodes. A specific example is the set-up of the naming system: information about DNS is not explicitly stored, it is implicit in the configuration of DHCP. So, when the naming system needs to be reconfigured, information is propagated through the DHCP daemon, which triggers a renewal of the DHCP lease on each node, or a reboot. Finally, a set of scripts has been written to build the DNS and DHCP tables, and this information is propagated back with rdist calls. In this case the complexity of the distribution of information is handled by an ad-hoc system, which is distinct from normal node configuration.

## 2.8 Summary

The case studies considered in this chapter represent medium-size installations with between several hundred and several thousand nodes. Such fabrics are sufficiently large to indicate current and near-future problems to be anticipated.

Reviewing the case studies, it is immediately apparent that while all of these institutions need reliable, efficient, and automated configuration management strategies, each delivers a different set of requirements and challenges. It is also evident that, in each institution, the underlying configuration strategy has evolved and adapted in order to tackle problems as they arise. In all cases, the configuration management process utilises a combination of third-party software and home-grown solutions, augmented by the technical expertise of the local administrative staff.

Within the organisations considered, the number of dedicated operational personnel varies between one and fourteen. In addition to maintaining the fabric, these staff have a number of other key tasks to perform. Such tasks may include:

- development and enhancement of the fabric resources;
- user support;
- one-off solution to obscure technical problems;
- other non-related, research activities.

Any substantial time allocated to maintain the availability and integrity of the fabric configuration will result in one or more of the above tasks being neglected. It is therefore imperative for a successful configuration technology that no significant manual intervention be required to maintain the day-to-day operation of the fabric. Sadly, having reviewed the roles described for technical staff in each of the studies, we conclude that this criterion is not satisfied in any of our case studies. Stated briefly, the evidence presented in the various case studies demonstrates that the current deficiencies in configuration technologies are:

- a lack of an abstract representation of pan-nodal relationships and dependencies within the fabric;
- an absence of an automatic sequencing mechanism, needed to resolve complex, inter-nodal dependencies both during the validation of a description and during its actual deployment;
- an absence of a sufficiently mature, fine-grain representation environment allowing devolution of authority and responsibility between multiple operational staff;
- a lack of support for mobile users—this includes users who regularly work at different terminals, and users who connect to the fabric from laptop computers;

- a lack of a mature and simple user-interface to the technology, coupled with an absence of formal and comprehensive documentation, resulting in a critical dependence on the availability of existing (knowledgeable) administrative staff.

We infer that these deficiencies lead to undesirable and debilitating restrictions on a user's freedom, such as: (i) inability to guarantee the correctness of a configuration, leading to an increased rate of configuration rot; (ii) limits on the availability/support of hardware, operating environment, and software products; (iii) constraints on the diversity and dynamism of the configuration of localities within the fabric; and (iv) deficiencies in the support of mobile or transient connections to the fabric.

It is evident from the studies presented in this chapter that the available configuration technologies are insufficient to meet the expectations of either the administrators or users: decisions regarding hardware and software acquisition are being adversely influenced by limiting factors in the technology (Studies 2.6 and 2.7). Also, flexibility of use for the compute resource is being tapered to constrain the volume of manual, administrative intervention required (c.f. Study 2.1).

We consider below some more specific observations relating to the case studies considered.

### 2.8.1 Installation versus configuration of nodes

It is interesting to note that only Study 2.6 fully integrates into the configuration technology the task of installing a new node onto the fabric. In all other studies, the initial task of igniting a new node is separated from the ongoing task of maintaining its configuration.

Separating these tasks aids simplicity and means that, for example, a node running a different flavour or version of a Unix operating system can quickly be incorporated into the fabric, without the need to address specific proprietary caveats and subtle nuances. However, the approach also has the following disadvantages:

- Once installed, the exact configuration of a node is difficult to determine within the context of the configuration environment—it may indeed be impossible to make guarantees regarding the details of the configuration.
- A proprietary and independently developed installation utility (such as Solaris Jumpstart) will not, in general, be capable of fully satisfying the target description provided within the configuration environment. For example, inter-nodal dependencies and aspects of configuration that require information from the wider fabric cannot cleanly be addressed by such a proprietary tool.

In Study 2.6, the administrative staff have chosen not to utilise existing proprietary operating system installation tools, but instead to regard installation as a “context” (i.e. special case) of configuration. Such an approach necessitates that the configuration technology be able to interact with the node at a rudimentary level, simulating the procedure normally executed by an installation tool. Within the School of Informatics, the additional effort required to achieve this has been found to constrain the diversity of hardware and operating system platforms which are supported.

### 2.8.2 Separation of server and client node configuration

It is interesting to note that in the majority of case studies conducted, the configuration of client and server nodes is addressed separately. Furthermore, the support of automated configuration for client nodes and services is generally more mature than the equivalent for servers.

Historically, it has been acceptable to assume that the number of servers within a fabric will be relatively low, and that their configuration will be much less dynamic, with only infrequent configuration changes being applied (for example, security patches). However, with the advent of Grid computing, the size and diversity of a fabric grows, the number and distribution of network services is likely to expand, as is the complexity of the relationships that exist between these services. Furthermore, modern fabrics frequently exist across multiple and disparate sites. In such an environment, manual configuration of servers is difficult, inefficient, and potentially unreliable.

### 2.8.3 Manual sequencing of configuration change

It is evident that the descriptive model employed in the studies under consideration fails to effectively represent pan-nodal relationships between nodes and services in a dynamic and diverse facility. Because of this, appropriate sequencing of the dependent tasks which exist across nodes—or which affect multiple resources—cannot be automatically resolved by the configuration technology and thus is referred to technical staff for consideration. This is both undesirable and prone to errors. Additional problems arise in fabrics which span multiple sites and for which communication between relevant staff is non-trivial; this will be typical of Grid fabrics.

As highlighted in Study 2.3 and Study 2.4, this may result in a reluctance to implement complex changes and thus mean that essential modifications relating to services, security and reliability are delayed or cancelled.

As the size and complexity of a typical fabric grows, obtaining a snapshot of the configuration state as a whole becomes meaningless. Then, scheduling the transition between one configuration and another becomes impossible to sequence by hand, necessitating some form of automated resolution.

### 2.8.4 Dependencies on administrative staff

All of the case studies considered in this chapter utilise significant amounts of home-grown software and experience. Due to practical time constraints, such solutions tend to be poorly documented with little formal process in place. Such a scenario places considerable reliance on the expertise of individuals and implies that new staff will need additional time and support in order to acquire a detailed knowledge of the system (see Study 2.5).

Furthermore, in Study 2.6, it has been observed that newly appointed technical staff require greater time to gain experience with the environment and appreciate the subtleties and nuances which are inevitably introduced.

### 2.8.5 Monitoring and simulation of configuration

A lack of effective monitoring of the state of the fabric is cited as a problem by all organisations interviewed. In a large and dynamic fabric, failures and errors will occur constantly: it is not sufficient for a monitoring tool to simply highlight such failures to staff, it must also instigate some form of corrective action or contingency in order to maintain an agreed level of service. An effective monitoring tool must not only address the effects of one-off failures, but also be robust to the less evident but more dangerous problem associated with configuration rot—this is particularly true of large-scale facilities which are intended to operate for long periods of time, virtually unattended.

In Study 2.7, we see an example of a typical response to configuration failure—to reboot the affected node (or nodes). This behaviour is, however, likely to be unsuitable for fabrics which support multiple, large tasks and persistent services. For such installations, it is more desirable that failed configuration changes be reversed, with minimal disruption to running services—such flexibility requires that the environment attain certain transactional characteristics with regard to configuration change.

Also, as alluded to in Study 2.2, in none of the facilities considered is there a formal system in place for simulating and validating the effects of configuration changes prior to implementation. This introduces an element of unreliability into the fabric management procedure and places additional strain on the configuration technology and the expertise of the administrative personnel.

In the future, as the typical fabric grows in size, dynamism, and diversities, these pressures will be magnified. The need for an environment in which experiments can be performed on prospective configurations and candidates can be tested for reliability and performance will become evident.

## Chapter 3

# Projected Use Cases

In this chapter we attempt to predict how the current culture of configuration management will evolve in the near-future. These predictions are based on: (i) an extension of the information which we have gleaned in Chapter 2; (ii) the unmet requirements of current-generation configuration technologies as identified in [ABK<sup>+</sup>02]; and (iii) the more general experience and knowledge of the authors. We hypothesise about how these changes in culture will affect the usage and expectations of users of configuration management tools, highlighting issues which are likely to present the most significant problems.

We give context to our thoughts by composing a number of realistic use cases, taking the real examples of the previous chapter as our guide. We present four use cases:

- a large academic department (network of workstations);
- a web service provider;
- a large-scale rendering facility;
- an OGSA-compliant Grid computing application.

For each use case, we set the scene by describing the purpose and extent of the example and define the infrastructure which is in place. We then give an explanation of the resulting configuration requirements, and illuminate the key problems which we envisage these requirements will spawn. We summarise our conclusions from these use cases in the last section of the chapter.

## **3.1 Projected Use Case 1: A Large Academic Department**

### **3.1.1 Background**

A large academic department consists of about 1500 desktop machines, running different operating systems and application sets. This is supported by a wide range of servers such as DNS, DHCP, print servers, web servers, file servers and databases. There are also a large number of assorted laptops, and one compute cluster with 64 nodes. The user community ranges from experienced and trusted computer scientists to naive administrative staff and first-year undergraduates.

A large team of system administrators manages these facilities, although often only as part of their job function. Some aspects of the system management such as the networking and mail infrastructures are managed by a central computing organisation, and some aspects are delegated to small research groups, or even to individual users.

### **3.1.2 Configuration**

All nodes, from servers to laptops, are managed by a single configuration tool. Access control on the configuration system allows management of various aspects to be delegated to appropriate users, or other organisations. A range of different interfaces provide different views of the configuration, suitable both for managing different aspects, and for use by different users with different levels of expertise. This allows individual users to have control over the configuration and software available on their machine, whilst ensuring a correct, secure, and recoverable configuration. The configuration tool includes a model of the entire site which is sufficient to permit validation of configuration correctness and integrity with a high degree of confidence.

Configuration changes are reflected immediately in changes to the actual fabric. If a new configuration is specified, the configuration system plans and deploys the sequence of changes necessary to move smoothly to the new configuration with minimum disruption. Where manual intervention is necessary, the system guides and advises technicians on the necessary operations.

Many aspects of the configuration adjust autonomously and dynamically, according to policies set with the configuration tool. This provides automatic load balancing and fault recovery for many services. For example, a failed print server will automatically be detected and some other node will reconfigure to adopt the role of the failed machine. This occurs without the intervention of a central configuration change, but the candidate set of replacement machines is specified by a central policy.

### **3.1.3 Challenges**

The above scenario presents many challenges which cannot be met by existing technologies. For example:

- Current configuration systems do not provide models and languages with adequate support for aspect delegation, either in terms of the basic compositional features, or in terms of security.
- Current configurations tend to be either very limited, or very complex. The more complex systems have steep learning curves and are not suitable for inexperienced users.
- Current configuration systems cannot plan non-disruptive sequencing of changes between declarative configuration states. This must be performed manually.
- Support for autonomous reconfiguration using peer-to-peer protocols is currently limited to a few specific applications, rather than being a basic feature of a configuration framework.

## 3.2 Projected Use Case 2: A Remote Rendering Utility

### 3.2.1 Background

A specialized computing facility is offering a remote rendering service. Customers can submit a rendering job (a 3D model along with textures, etc.) and are sent the final rendered images or video back. The hardware base in the facility itself is made of hundreds of thousands of highly standardized, low-end nodes. The facility can provide a set of rendering applications for customers. The management staff are responsible for provisioning the hardware and taking care of operating systems and rendering applications maintenance.

### 3.2.2 Configuration

Individual nodes in this type of facility are considered disposable. The administrator deals with a sea of mostly identically configured nodes. In case of failure, a single attempt is made to reinstall the node from a standard image, and if this fails again the node is ignored. Eventually whole blades of servers are replaced when most of their nodes have failed.

Specialized customer-owned clusters are deployed on demand. The customer describes the number of nodes and the rendering application he needs. A set of nodes is isolated that satisfies the customer's demand. Appropriate standardized images are installed. This set is partitioned between a master and its slaves, and a handle to the master is finally returned to the customer to get back the result of his computations on the cluster.

### 3.2.3 Problems

This scenario illustrates two different issues. The first is the management of sheer scale. It will be difficult to extract, represent or reason about information coming from the fabric. There are two main aspects of this:

- Even after reducing the complexity, the facility manager might not be guaranteed to be able to describe the state of its entire computing facility. Nodes keep coming and going, being reallocated, failing, etc.: the information will be inaccurate.
- Distributing and gathering information on the facility may be a very complex task. On the one hand if the information is distributed across nodes and shared in a complex way, it might take too long to get the answer to a query on the system if nodes have to collaborate to generate it; on the other hand if the information is centralised it can then easily be out-of date, because the speed at which the information is collected may be slower than the speed at which the system changes. In any case, computations on cross-nodes information may increase the response time of the fabric.

A second problem is linked to the different descriptions of the system, held at all levels: low-level descriptions of the nodes and their hardware, (preferably) high-level descrip-

tions of the customer's intended job, descriptions of the network's configurations, of the applications licenses pools, of clusters deployed in the fabric.

It is very likely that the fabric administrator would want to be able to get information on, or describe the facility in terms of its capacity, for example the average computation time they may provide to a customer and the price to pay for it. The translation from number of nodes, their memory and their CPUs to an intelligible figure for the customer might be impaired by poor language richness at every level:

- It is impossible to use the same language to describe nodes, clusters, applications requiring translations at every stage;
- It is difficult to manipulate the descriptions to get the relevant information;
- There is no versatility in the language to allow for new constructs to be introduced and understood by the fabric, the management, and the customers.

In any case the administrator would like to be able to give high level commands to the fabric, such as “update security patches of all the rendering nodes of the fabric with Maya and Linux”. The tools (language, visual interface etc.) provided to the humans administering the fabric need at the very least to provide abstractions and the possibility to aggregate vast entities into logical ones that can then be coherently manipulated at the administrator's level.

In addition, the whole facility may want to be able to manipulate and work with some precise type of values: for example time, or percentages. Customers, for examples, might not want to think about the number of nodes they'll have to use. They want to submit their jobs in more “implicit” terms: the quality of the rendering and the time constraints they have. It is then up to the facility to give them options on the price they'll have to pay for this service. This type of dialogue is only possible if both parties can:

- Extract and provide the relevant information in sufficient detail;
- Translate the relevant information correctly, in which case the language used by both the facility and the customers needs to be flexible enough to allow such derivations.

### 3.3 Projected Use Case 3: A Computing Utility Providing Infrastructure On Demand

#### 3.3.1 Background

A computing facility provides “infrastructure on demand” to customers: a computing farm of large size (thousands of nodes) is used to provide hardware and networking infrastructure to external users on a “utility” basis. Customers can then deploy their applications and services on the allocated infrastructure, leaving the responsibility of provisioning and maintaining the infrastructure to the computing facility. The actual administration of the customer’s applications remains the responsibility of the customer. As an added service, the computing facility offers application deployment facilities, providing the customer with pre-packaged software solutions (three-tier web service, for example).

#### 3.3.2 Configuration

The deployment of a customer infrastructure is dealt with in two phases: the customer describes the hardware capacity they need, as well as some of the networking constraints on this network. If it can provision the demand, the facility allocates the necessary number of nodes, internally configures the network and deploys the requested servers (firewall, dns, etc.). The customer is then given a handle on this deployed infrastructure. They can either:

- Deploy the applications they need through their own chosen configuration technology;
- Or use the deployment facilities provided by the utility to get its services deployed.

In the second case the customer is given a choice of pre-packaged solutions where they can customize some pre-defined attributes. Once the solution has been chosen and configured, it is handed over to the facility, which ignites it on the set of nodes owned by the customer.

#### 3.3.3 Problems

Two main concerns arise from this hypothetical scenario. First, the customer should be able to assume the security threats on their infrastructure are not increased by deploying it on the computing facility. More specifically:

- The flow of information should be clearly defined and auditable: the customer should be able to communicate with the utility (ask for more resources, change existing ones, etc) but the actual content of the node, for example files created by and belonging to the user, should remain impermeable to the computing facility itself.

### *3.3. Projected Use Case 3: A Computing Utility Providing Infrastructure On Demand*

---

- Such a facility will typically have several customers, hence sharing the nodes between them. It is therefore crucial to ensure the computing facility's own management system cannot be used as a bridge between customers, and thus customers remain impermeable to each other.

The second problem illustrated by this scenario is the lack of versatility of the proposed application system. It is unrealistic to assume customers will find the variety they need merely in the parameterization of pre-packaged solutions. Ideally they'd like to be able to:

- Describe and replicate their own solutions or existing infrastructures. They might then reuse these templates on other fabrics, or even trade them with other customers;
- Compose their own templates with others, including the facility's solutions, to build more complex solutions.

Parameterized packages do not have the necessary expressive power to allow either of the above.

## 3.4 Projected Use Case 4: Interacting Grid Services

### 3.4.1 Background

Both Web Services and Grid services (or OGSA services) are used for setting up a distributed service for help to decision during a crisis, for example response to a chemical threat. We'll consider the four following actors:

- A long-running, well established service such as weather prediction;
- A database access service for maps and critical buildings' blueprints (the chemical plant, villages around etc.);
- A computation and visualisation service allowing deciders (MOD, emergency services, etc.) to plan future actions;
- An OGSA-compliant Grid fabric.

All these services are managed by different fabrics, some of them Grid-enabled.

### 3.4.2 Configuration

Out of these only the last one is, strictly speaking, a Grid service: we'll assume the first two are standard web services, i.e. the resources they use to provide the service are not on the Grid. The third is deployed as a service on the Grid, at the time of the crisis. In this particular case the use of OGSA is two-fold:

- Request is made for a particular set of resources through OGSA-compliant Grid farms;
- Once these resources are obtained, the visualisation service is deployed and is itself accessible by other Grid services through an OGSA-compliant interface.

### 3.4.3 Problems

This projected case study highlights two problems linked with interoperable services on the Grid. The first one relates to the information a Grid-enabled fabric has to communicate. The Grid fabric registers resources with directory services which are then read by resources brokers. This causes several problems:

- It may not be possible for the fabric to precisely describe the resources it owns; some may be virtualised, some of the information may be out of date. Generally, it may not be possible for the fabric to know its own state at a very low-level;
- A fabric may wish to hide the physical nature of the resources it provides, for security or strategic reasons.

But at the same time, it is also essential to be sure to provide sufficient information, or access to it, for the resource consumers to correctly choose the resource provider which they think is the best match for the task at hand. It comes down to being able to decide what is the minimal amount of exposed information necessary for a Grid to be usable.

The second problem is the generic problem of composition. Grid Services are generally described in terms of their interdependencies. In our example, the rendering service needs the weather service and the blueprints. However it is not easy to specify further relations between these services, nor to compose new services out of these building blocks. Extra languages may be used, but as such the descriptions (WSDL) of these services can not be easily composed. Once a service has been formed it is not an easy task to decide to group or rearrange in some other the parts it is made of. For example one might want to:

- Group all the services of our example in a single description, to be able to deploy it as a single instance in a single fabric;
- Build a slightly different service out of the example, such as replacing the real weather service by a fake one to conduct simulations, or the visualisation part by a controllable view point to train emergency services employees;
- Reuse or inherit from a standard service description and change only a few parameters, such as running several simulations with different weather conditions.

These type of changes will require most of the time to rewrite the whole service anew. It is hard to reuse existing templates: they do not form well contained, self-specified units. For example some may not specify the values their parameters use (range, conversion), or not give sufficient information on the services they rely on themselves. Reusability is hindered by the lack of composability between services.

## 3.5 Summary

The use cases considered in this chapter are tailored to demonstrate and explain the fundamental problems of configuration management which the authors perceive. In preparing these use cases, we have not only considered the effects of increasing scale, but also the significant influences of the following factors:

- Increased expectations of users and administrators with regard to dynamism and responsiveness of configuration;
- An expansion of hardware/software diversity;
- The emergence of complex, multi-site installations with overlapping trust domains, and consequent issues of security and reliability;
- The physical (and conceptual) gap introduced between the user/administrator and the configured fabric in the case of “utility computing”;
- Changes to the scope of user control over scheduling and availability as introduced by “utility computing”.

In the remaining sections of this chapter, we have highlighted each problem in turn, demonstrating the conditions under which it will develop, and beginning to consider how to resolve such issues.

### 3.5.1 Multi-level abstraction of configuration description

The anticipated expansion in configuration diversity and dynamism makes it impractical for administrative staff to explicitly configure and manage all of the details of a fabric. Furthermore, in devolved, multi-site installations individual administrators may have incomplete knowledge regarding the facility and its configuration (see Use Case 3.1) or, for security reasons, may have privileges restricted to specific aspects of configuration, (see Use Case 3.2).

In this situation, what is required is an ability to specify the broad structure of the fabric using high-level template components. From this, low-level configuration information (needed at the deployment level) can be deduced automatically and appropriately, based on predicates and default attribute values. Such a mechanism allows one to specify configuration in terms such as “a cluster of ten nodes with a naming service, web server, and a database”. Such a high-level descriptive model also furnishes the possibility of expressing more abstract concepts such as contractual requirements—service level agreements (for example, specifying the agreed level of down-time which is acceptable in a web service, see Use Case 3.3).

However, in practice this high-level representation is not—on its own—sufficiently versatile, since individual fabric installations are subject to site-specific subtleties and idiosyncrasies which cannot be expressed or deduced without manual intervention and preparation at a lower level. Thus, the model must also provide the opportunity to an adminis-

trator to delve down into these lower levels of the configuration, possibly affecting the description at the point of individual attribute assignments.

At present, a typical configuration language and model provides only a shallow range of depth in which to express a configuration description. If this is set at a high level (of abstraction), then configurations are simple to create but the environment is too restrictive and bland for truly meaningful and practical expression. Conversely, if the level of abstraction is low and staff create descriptions in a language which is comparable to the canonical form interpreted by the deployment engine, then the description quickly becomes over-verbose and unwieldy. At a low level of abstraction, the technology is difficult to learn and creating configuration becomes error prone. Such an environment is not conducive to the implementation of consistent authoring practice and standards. Furthermore, in diverse fabrics it becomes impossible for staff to appraise the configuration of the fabric as a whole.

The ideal situation is to have a language which allows users to interact with and create configuration descriptions with a range of depth of detail. This would allow staff to sketch out the broad structure of the fabric (or a part thereof) using simple template constructions, then concentrate on resolving any subtle site-specific issues at a lower level. Such a multi-level technology does, however, realise other problems relating to devolution of management and security which are considered in the following sections.

### **3.5.2 Devolution of management and composition of configuration**

The need for devolved management is something which we have already identified when considering the case studies in Chapter 2. In current examples, the way one partitions the configuration of the fabric into domains of responsibilities is seriously constrained by the capabilities of language and modelling technology—these environments are not mature enough to allow convolution of multiple, partial configuration descriptions with significant overlapping effects. As the size and complexity of a typical fabric grows, it will quickly become impossible to suitably partition configuration responsibility between multiple administrative staff, avoiding intrusive overlap and the subsequent scope for conflict (see Use Case 3.1).

When overlapping descriptions are augmented together, repeat attribute assignments have to be sequenced correctly, and predicates have to be combined so as to create valid and meaningful rules in the resulting description. Such complexity demands careful consideration during the creation of the model and language.

In the future, problems related to conflict resolution are likely to be exacerbated. Within the multi-level model, as described in Section 3.5.1, conflicts are difficult to identify and an invalid configuration may not be apparent until late on in the compilation process, at the point of deployment.

It quickly becomes apparent that some conflicts (or failed constraints) are inevitable. It is not sufficient that the deployment engine simply highlights such issues, but must also establish some automatic resolution, based on the relative merits and priorities of the conflicting contributions.

In addition, to facilitate problem rectification and debugging, it is important that individual contributions to the united description can be authenticated to their respective author(s), and the origin of attribute values can be traced back through the augmentation process to the source.

### 3.5.3 Security

In many situations, it is of great importance that the security of individual aspects of the fabric not be compromised when the associated configuration descriptions are combined. Within a devolved environment, such as that described in Use Case 3.1, it is a significant challenge to ensure that the integrity of the fabric is maintained in a situation for which different overlapping aspects of configuration may be submitted by both users and administrative staff.

In Use Case 3.3, we see an increasingly common situation, in which jurisdiction over security and configuration are dis-associated: the service provider has responsibility for security and integrity, while individual clients submit autonomous requests for particular configurations. This situation is best visualised as a multiple tier configuration, in which information in each tier must be isolated from all other tiers, but implicitly has an influence on the dynamic behaviour of the overall installation. In the Use Case presented, the customer implicitly trusts the service provider to secure their particular allocation within a shared fabric resource, and anticipates that the service provider will not be able to access any of their data.

Such an example is further complicated if we add the assumption that resource provision may be delegated by the appointed service provider to one or more sub-contractors during a period of high demand: then data and customer integrity has to be secured in an even more complicated arrangement of inter-linked tiers.

### 3.5.4 Highly dynamic fabric configuration

Pressure for a greater dynamism in configuration is already a reality in large fabric installations. Sadly, the typical response, currently, is to suppress such dynamism and restrict the rate at which the configuration of the fabric can be changed, therefore postponing any invasive modifications to the configuration until a time at which the affected portion of the fabric can be reset. In the future, such a practice will not be viable and it will become essential to tackle the problems associated with highly dynamic environments (see, for example Use Case 3.1).

At the heart of dynamic configuration is the ability to be able to seamlessly change the configuration of one aspect of a neighbourhood of the fabric in a non-disruptive manner. To do this, one needs a configuration technology that can support versioning (that is, can generate appropriate transformations to migrate a subset of the configuration from one state to another) without violating the predicates which are built into the governing model or creating an invalid configuration (for example, through failed dependencies) at any point during the migration.

For web services, the ability to version configuration changes is crucial when maintaining

the availability and reliability of the installation (see Use Case 3.3). In this situation, one needs to be able to patch and upgrade complex applications in order to preserve security without affecting the running systems. Either the fabric must remain in a consistent and secure state at all times, or potentially affected services must be suspended.

### 3.5.5 Monitoring and failure recovery

For large installations, such as a rendering farm (Use Case 3.2), it is unrealistic to expect that the fabric will ever function at one hundred percent, and failures due to faulty hardware, user applications, or configuration changes become an everyday occurrence.

As noted in Chapter 2, in such a scenario it is not practical to expect administrative staff to be able to intervene at the source of failure and manually correct problems: in fact, appropriate staff may not be on site at the point of failure. What is needed is a monitoring environment which can both detect and resolve problems as they arise. The monitor should be sufficiently capable that only a small proportion of failures need ever be referred to a technical staff member.

Failure should also be considered during the modelling process. This is particularly the case for high-level, abstract model components which express contractual requirements such as level of service or time to completion. At this level, it is important that the user can express concepts such as probability of failure and mean response times, and can reason within the description based on these concepts.

### 3.5.6 Massive scale configuration

When predicting potential failings of configuration technologies in the future, it is useful to consider how the current deployment mechanisms and services would cope with the effects of a significant increase in scale. We have already seen in Chapter 2 that many existing deployment mechanisms, such as those employed in Study 2.6 and Study 2.7, would become overwhelmed should the size of the installation grow by as little as five times. This is a symptom of the centralised server-side control of configuration which is implemented by the majority of configuration technologies. As highlighted in Use Case 3.2, in a situation where tens of thousands of nodes need to be re-configured quickly and concurrently, the impact on network load becomes an important consideration and such bottlenecks need to be eliminated. Furthermore, one must evaluate whether the services which are deployed on the fabric (such as DHCP and DNS) can be configured in a manner that allows them to be responsive and reliable in such a large installation.



## Chapter 4

# Conclusions

### 4.1 Preamble: A Paradigm Shift for Computing

This report explores a range of real-world and hypothetical case studies, to illustrate how fabrics are being managed today, and to anticipate the difficulties that will be experienced in the future. The intent is to guide our thinking about the key research challenges in large-scale system configuration that must be solved to enable the configuration and management of Grid resources.

This deserves a higher-level description of the context in which our research is taking place.

The drive towards Grid-based computing is one leading indicator that computing infrastructure is in the initial phases of its next paradigm shift – where a new, over-arching computing model replaces the previous one. As computer hardware has become more powerful, cheaper and smaller, it is proliferating to an extraordinary degree. And there is no sign that this trend will decelerate, especially as networked computing capabilities become embedded in ever-increasing numbers of devices which are not traditional computing nodes.

This is evidenced, for example, by the emergence of very large commodity clusters and high-density, blade computing arrays. The number of individual nodes that comprise such infrastructures is increasing rapidly. Further evidence is offered by the emergence of connected digital appliances, such as smart-phones and home entertainment equipment, which incorporate powerful, networked computing capabilities.

The proliferation of computing infrastructure has generated challenging system configuration problems that must be addressed, and which are helping to drive the paradigm shift. The first challenge, clearly, is that of scale. We must find ways of configuring and managing fabrics which are orders of magnitude larger than before.

The second challenge is that of complexity: our computing fabrics now have an effectively endless set of configuration possibilities, often with numerous inter-dependencies. This complexity is approaching the level where it is simply intractable using current approaches. A number of the case studies in this report illustrate dimensions of the complexity problem: the interactions between different *elements* of the fabric (e.g., relationships

between clients and servers); the interactions between different *aspects* of the configuration (e.g., networking, security); the requirement to support high levels of *diversity* in the fabric.

The third challenge is a side effect of the first two: that of the sheer labour required to keep large, diverse and complex computing infrastructures in correct, working order. The number of staff required is still scaling approximately linearly with the size of the infrastructure, and this is simply not sustainable. This is also reflected in the case studies, such as in the typical case where servers are managed manually, even if clients are supported automatically.

The paradigm shift, therefore, is one of raising the abstraction level at which we manage our computing resources. Under the new paradigm, we will no longer think of individual nodes within computing fabrics, but we will think at least at the level of collections of nodes (and other infrastructure components) which are configured and managed as single entities. For example, we need to be able to deal with a complete computing cluster as single entity, rather than the individual compute nodes, servers, storage and networking components that comprise the cluster.

There are useful analogies in the way that a computer operating system works. A computer is made up of many discrete components, including CPUs, memory, secondary storage, I/O interfaces, and directly connected devices. The role of the operating system is to orchestrate the activities of these separate components, make them function as a whole, and offer – through virtualisation – an abstraction to applications which makes it appear as if each application ‘owns’ the computer. The operating system hides the complexity of configuring each element, making the complete assembly of elements work together.

In the new computing paradigm, individual nodes are just like components of a larger, abstract, distributed computer which is managed as one entity.

Realising the new paradigm raises many interesting research problems. “Mainstream” Grid computing research is attacking many of the the upper layers of the problem space, providing ways to discover and utilise appropriate resources in a global scale computing infrastructure.

The focus of the GridWeaver Project is on the lower levels of the problem space, dealing with complex system configuration. The ultimate goal of our research is to provide fabric configuration and management solutions that are capable of handling the full complexity and diversity found in emerging computing fabrics, and which are able to automate many node-centric activities that would today need to be performed by people. Such solutions will be capable of (for example) automatically configuring a complex Grid fabric from the ground up, to permit its participation as an element in a global Grid.

The case studies in this document, as well as supporting our intuition that new approaches and technologies are needed to configure and manage future computing fabrics, have provided guidance as to what key research problems need to be solved to realise our vision.

## 4.2 Research Challenges

In this section we discuss some of the key research challenges that emerge from these studies. We have split these into three categories, covering different dimensions of the problem:

1. Representing system configurations;
2. Using and manipulating system configurations;
3. Deploying system configurations.

### 4.2.1 Representing system configurations

We believe that in order to manage complex system configurations, it is first desirable to have a language-based system that allows us to express, manipulate, and reason about, all relevant aspects of the the configuration. We further believe that this goal is best served by adopting a declarative language for expressing configuration. Given this, research challenges include:

- The basic design of the language itself: we require a language that is able to express all the necessary relationships between the configuration parameters that form part of our system configurations. The language does not itself embody the models we are representing, but it must be capable of representing all of the required attributes of the models.
- Representing a wide spectrum of abstraction levels and levels of detail: we must be able to express levels of detail and abstraction varying from very low-level device configuration parameters up to large-grained entities such as clusters. This requires that we be able to compose system configurations by building up high-abstraction components from lower level ones.
- Representing a rich range of dependencies between configuration parameters: this is vital in order, for example, to support relationships between clients and associated servers and to perform “what if” analyses on planned configuration changes. We also want to be able to support resolution of dependencies at different times: some will be resolved statically, some at configuration deployment time, while others will change during the lifecycle of a deployed configuration.
- Support for automated checking of configuration correctness: as configurations become larger and more complex, the scope for errors increases. We need to support mechanisms in our representation system that allow automated checking of configuration parameters, even as configurations are composed from arbitrarily many levels of sub-configurations.
- Support for temporal relationships: we would like to support configurations that change over time. This requires that the language be able to express the way in which configuration parameters should change over time.

- Implicit specification of configuration parameters: this is the ability to automatically generate appropriate sets of configuration parameters based on some specified set of configuration requirements – e.g., “I need a cluster that supports 200 Gflops” might automatically populate cluster configuration parameters specifying node performance, node numbers, and inter-node networking configuration to deliver the required cluster performance. This desired property can be extended to more abstract requirements, such as those related to reliability or security.

#### 4.2.2 Using and manipulating system configurations

Given a language system that is capable of correctly representing complex configurations, we have to consider many practicalities related to how the system is used in the real world. Research challenges include:

- Ease of use: representing complex system configurations using the language directly poses ease of use challenges. Writing complex configurations – including the rules used for correctness checking – is a highly skilled task, and we would not expect every system administrator to have to deal with configurations at this level. We need a system that exposes configuration information at an appropriate level of detail by, for example, providing graphical environments for configuration manipulation.
- Devolved control over configurations: any large scale fabric will have several different aspects to its configuration. Examples of aspects include the configuration parameters related to security, networking, shared storage, mail-servers, and so on. Any practical system will allow different individuals or organisations to control different aspects of the configuration, and to do so securely. That is, they get to control only the aspects they are permitted to control. Also, different aspects inevitably interact, and managing this interaction in a way which preserves correctness and aspect security is a difficult problem.
- Pre-deployment configuration inspection and simulation: when making configuration changes, it is useful, if not essential, to be able to determine *a priori* the effect that the change will have. When a set of configuration parameters are changed, we need to be able to detect the resultant impact this will have on the overall, fully-resolved system configuration. This includes being able to trace how each parameter in the resultant system configuration was set. Furthermore, it is desirable to be able to test the impact of a change by seeing its application to a test fabric – either by simulation or by supporting a test environment isolated from the full production systems.
- Automatic sequencing of configuration changes: when a set of configuration changes needs to be made to a fabric, it is very useful to have automated support for determining the order in which changes should be carried out.

### 4.2.3 Deploying system configurations

The third set of research challenges relates to the process of deploying system configurations onto fabric resources to create correctly configured, running systems, and then to maintain them over time. This requires the creation of runtime deployment components that can gather and interpret relevant configuration information, and correctly execute the resulting configuration actions. Research problems include:

- **Scaling:** the deployment mechanism must scale to tens or hundreds of thousands of nodes. Among other things, this implies that configurations cannot necessarily be deployed “instantaneously”, and may need to converge over some time with the desired configuration.
- **Supporting volatile connections:** fabrics may include resources that are intermittently connected to the network, and indeed attached to different networks over time. This also implies that nodes may adopt different “personalities” depending upon some set of conditions, in this case the type of network connection.
- **Monitoring configuration state:** in order to manage a fabric, we require knowledge of its current configuration state. This monitoring function is non-trivial for large fabrics in which it is effectively impossible to determine a single configuration state, since the fabric is constantly changing. Nevertheless, we must have a mechanism that is capable of offering useful information about fabric state, at appropriate levels of detail.
- **Supporting configuration workflows:** there is often a requirement to sequence a set of configuration changes across the fabric. There are many configuration changes that require this, and it also permits changes to be made without taking down an entire fabric. Today, this is typically done by hand, and is therefore inefficient and error-prone. The goal is to replace manual methods with others, enabling to describe the sequence in which a set of changes must be made; describe the trigger conditions that permit moving to the next step in the sequence; describe any failure recovery steps; and then to automatically have the set of changes applied. A simple example is applying a security patch, which could roll through a fabric one node at a time.
- **Transactional application of configuration changes:** a “single” logical change to a fabric configuration can involve changes to many nodes on the network. If such a change fails part way through, the configuration system should be capable of recovering the fabric back to a known, consistent, running state.
- **Automatic reconfiguration:** we would like our systems to automatically reconfigure themselves in response to certain conditions. A typical example would be a reconfiguration to accommodate a failure – for example, connecting each client to a different print server if the primary server fails. Another canonical example is the flexing of server resources to adapt to changing demand. These require that we be able to:

1. Detect conditions in the fabric at runtime, e.g. to detect that a server is no longer available, and furthermore to ensure that the detection of this state is shared by all relevant components;
  2. Automatically execute and orchestrate specified configuration changes from directly within the runtime configuration itself.
- Security: there are myriad security challenges associated with ensuring that configuration changes can be made only by those who are authorised to do so. Furthermore, we also want to protect the configuration system from attack by the applications running on the fabric.
  - Reverse engineering of existing systems and nodes: the ideal configuration system would be able to interrogate parts of fabrics that it does not currently manage and produce a baseline model that can then be used to bring these parts within the control of the configuration and management system.
  - Practical issues: there are several issues related to creating a practical, useful deployment system. While these are not research challenges *per se*, solutions are nevertheless required. This category of problem includes:
    - Installation of nodes from the “bare-metal”: ideally, the configuration system will provide seamless support of tasks from the low-level installation of BIOS version, OSes, etc., right up to pan-node, complex distributed services.
    - Multi-platform support: support for all common operating systems and hardware platforms.
    - Broad device and application support: in order to configure devices or software components, the runtime deployment system must know how to configure these elements. Typically, this requires that components be written that are capable of interpreting configuration parameters for a device or application, and making the appropriate configuration changes.

### 4.3 GridWeaver Research Focus

GridWeaver will not solve, nor even work on, all of the research challenges listed above. We are focusing on a subset of interesting and challenging problems, including:

- Development of a suitable language system (syntax and language tools) for representing system configurations;
- Development of an experimental architecture for the runtime configuration deployment system;
- Design and development of a small number of system components to explore the use of the language and to test the architecture;

- Demonstration of the use of this approach by configuring a fabric to support a challenging application. This will probably be to show that we can automatically configure a Globus-enabled fabric automatically from bare nodes.



## Chapter 5

# Appendix: Case Study Questions

This appendix pulls together a set of questions that act as a “concept road-map” for exploring how large-scale system configuration is done in a variety of contexts. We used this as a tool for a number of the case studies in this report.

### 5.1 Physical composition and scale of the fabric

#### 5.1.1 Background

Can you outline the background (in terms of use and history) of the fabric? Does the fabric serve a dedicated purpose/application, or does it provide more generic functionality?

What type of users are supported by the fabric? Do these users have diverse and critical requirements for applications and configuration of resources?

#### 5.1.2 Scale

How many nodes are there in the fabric?

How many separate sites make up the fabric?

How many staff are allocated to maintaining the fabric configuration? Are they committed full time, part time, or as and when required?

Is the size of the fabric fairly static, or are nodes added and removed from the fabric regularly?

#### 5.1.3 Diversity

What diversity of hardware is in use? (in terms of purpose/manufacturer/age)

What diversity of software is in use? For example, operating systems, applications, in-house software.

Is there a significant diversity in the configuration of different nodes? For example, are nodes collected into clusters sharing the same basic configuration information?

Are hardware/software purchasing decisions influenced by configuration management requirements and restrictions?

#### **5.1.4 Dynamism**

On average, how frequently are configuration changes applied?

Are configuration changes collected together and distributed in a batch at a specified frequency, or are changes applied in a more dynamic manner?

How frequently are OS upgrades applied to nodes on the fabric?

How frequently are new nodes installed onto the network?

How frequently are nodes re-built from scratch?

Does the fabric include volatile connections, such as laptop machines.

## **5.2 Current configuration management tools used**

What configuration management tools do you currently utilise? Include imported tools (for example, commercial technologies) and home-grown solutions.

What issues influenced your choice of these technologies?

If you use an "off the shelf" solution, is the technology implemented as is, or are local "hacks" included in order to tailor the system to local requirements, or fill gaps in the functionality of the technology?

Do you have experience with any other configuration technologies/techniques?

What type of user interface is provided for interacting with the technology? Does the technology provide a high-level representation of aspects of the configuration of the fabric?

## **5.3 Current configuration management strategy**

### **5.3.1 Scope of configuration**

Can both servers and compute nodes (that is client machines) be configured automatically? Alternatively, are server nodes deployed manually?

What type of day-to-day involvement is required to maintain the state of the fabric?

Does configuration cover low-level aspects of network configuration?

Does configuration cover services such as storage and backup strategies?

### 5.3.2 Delegation of tasks

Is responsibility for configuration of the fabric allocated to one person, or more than one person?

If more than one, how is responsibility delegated? For example, do you allocate specific aspects of fabric configuration to different people, or different groups of machines, or is there a more ad hoc approach to allocation of responsibility?

### 5.3.3 Tasks which require manual intervention

Which tasks (if any) lead to the greatest drain in staff time?

Which aspects of the fabric's configuration do you consider to be lacking in support?

Are there individual personnel who are critical to the maintenance of the fabric?

### 5.3.4 Monitoring and problem detection

Do you have facilities for monitoring the actual configuration of the fabric?

Have you developed any metrics for quantifying attributes such as the level of service availability, the response time for configuration change, or the level of configuration rot?

How are unsolicited changes to the configuration of the fabric resolved?

What facilities are in place for monitoring failures within the fabric? Some examples of failures might be:

- failure of configuration change deployment, either due to failed dependencies or poor specification of configuration changes;
- hardware failure;
- lack of authorisation to perform the configuration changes.

How do you deal with failures in the fabric? For example, are failures resolved manually?

Do you encounter user requests that are difficult or impossible to deal with?

## 5.4 Future plans and problems envisaged for the future

Are there any significant changes in the scale/use of the fabric planned in the near future?

If the scale of the fabric were to be increased by a factor of 5 to 10, can you speculate as to what new problems and issues would you face? For example, would the increase in size of the fabric, increase the amount of staff time required?

## 5.5 Wishes for future configuration management tools

Do you have a wish list of functionality and features you would like to see in future technologies? This might be a reflection of current problems or future expectations (or a combination of the two).

Some examples are:

- a good GUI for performing high-level configuration operations on the fabric;
- improved monitoring of the actual configuration in order to facilitate the validation of actual configurations against the target state of the fabric.

# Bibliography

- [ABK<sup>+</sup>02] Paul Anderson, George Beckett, Kostas Kavoussanakis, Guillaume Meche-neau, and Peter Toft. Technologies for large-scale configuration management. Technical report, The GridWeaver Project, December 2002.  
<http://www.gridweaver.org/WP1/report1.pdf>.
- [arg] Argonne National Laboratory. Web page.  
<http://www.anl.gov/>.
- [AS02] Paul Anderson and Alastair Scobie. LCFG - the Next Generation. In *UKUUG Winter Conference*. UKUUG, 2002.  
<http://www.lcfg.org/doc/ukuug2002.pdf>.
- [Bur95] Mark Burgess. Cfengine: a site configuration engine. *USENIX Computing systems*, 8(3), 1995.  
<http://www.iu.hioslo.no/~mark/research/cfarticle/cfarticle.html>.
- [cera] CERN. Web page.  
<http://public.web.cern.ch/public/>.
- [CERb] CERN. SUE. Web page.  
<http://wwwinfo.cern.ch/pdp/ose/sue/doc/sue.html>.
- [DIC] Distributed Informatics Computing Environment. Web page.  
<http://www.dice.informatics.ed.ac.uk/>.
- [Edi] Edinburgh School of Informatics, EPCC and HP Labs. The GridWeaver Project. Web page.  
<http://www.gridweaver.org>.
- [Eva01] Remy Evard. *Chiba City: A Case Study of a Large Linux Cluster*. MIT Press, 2001.
- [Ham] Martin Hamilton. RedHat Linux Kickstart HOWTO. Web page.  
[http://metalab.unc.edu/pub/Linux/docs/HOWTO/other-formats/html\\_single/KickStart-HOWTO.html](http://metalab.unc.edu/pub/Linux/docs/HOWTO/other-formats/html_single/KickStart-HOWTO.html).
- [inf] School of Informatics, University of Edinburgh. Web page.  
<http://www.informatics.ed.ac.uk/>.
- [Mic97] Sun Microsystems. Preparing custom Jumpstart installations. In *Solaris 2.6 Advanced Installation Guide*, pages 71–126. Sun Microsystems, 1997.

- [WP4] DataGRID WP4. EU dataGRID WP4: Fabric management. Web page.  
<http://hep-proj-grid-fabric.web.cern.ch/hep-proj-grid-fabric/>.