

Application Deployment over Heterogeneous Grids using Distributed Ant

Wojtek Goscinski and David Abramson
{wojtekg,davida}@csse.monash.edu.au
School of Computer Science and Software Engineering,
Monash University, Victoria, Australia

Abstract

The construction of large scale e-Science grid experiments presents a challenge to e-Scientists because of the inherent difficulty of deploying applications over large scale heterogeneous grids. In spite of this, user-oriented application deployment has remained unsupported in grid middleware. This lack of support for application deployment is strongly detrimental to the usability, evolution, uptake and continual development of the grid. This paper presents our motivation, design and implementation of the Distributed Ant user-oriented application deployment system, including recent extensions to support application deployment over heterogeneous grids. We also present a significant Distributed Ant deployment case study, demonstrating how a user-oriented application deployment system enables e-Science experiments.

1. Introduction

Grid computing has been proposed as the next generation of infrastructure to support distributed applications in science, engineering and business [1]. The Grid provides mechanisms that harness computational resources, databases, high speed networks and scientific instruments, allowing users to build innovative virtual applications. Such virtual applications can be synthesized by combining multiple different pre-existing applications on multiple computational resources. A number of projects have focused on providing a suitable runtime environment for these virtual applications. For example, the Globus project defines middleware interfaces that support application invocation in a secure, distributed, environment [2]. However, a significant, unaddressed, challenge for Grid computing concerns the difficulty in *developing, distributing* and *maintaining* software over a large distributed and heterogeneous resource base. Traditional development environments support a range of important software engineering processes, such as high level specification, source code control, editing, compilation debugging and even execution. But existing environments do not understand the Grid infrastructure, in which applications need to be

*deployed*¹ to a range of different resources. As a consequence, users have to use manual techniques which are both inappropriate and un-scalable.

In response, we have developed an *automated application deployment system* with a *user-oriented* approach, Distributed Ant (DistAnt) [3]. A user-oriented approach means it is targeted at users with reasonable knowledge of the application they are deploying, but strictly limited grid computing knowledge, resource information and importantly, resource authority. DistAnt provides a simple, scalable and secure deployment service and supports a simple procedural deployment description.

This paper presents a significant advance over our earlier research; it focuses on application deployment over *heterogeneous grids* and *uncharacterized resources*, and provides a demonstrating case study. DistAnt supports application deployment over heterogeneous grids by virtualizing certain grid resource attributes to provide a common application deployment gateway, deployment description, file system structure and resource description. To manage remaining resource heterogeneity DistAnt supports *sub-grids*, to divide an unmanageable heterogeneous grid into manageable sets of like resources, categorized by resource attributes that can be queried. Sub-grids provide a framework to execute environment specific remote build routines, compile an application over a set of resource platforms and redistribute binaries to the entire grid. DistAnt also supports definition and deployment of application dependencies. As this paper will demonstrate, *DistAnt enables deployment of a complex native application over an uncharacterized heterogeneous grid, assuming nothing about grid resources.*

Furthermore, integration of DistAnt into our grid parametric sweep software, Nimrod/G [4], provides an overall environment enabling grid scale application development, deployment and execution.

In presenting this paper, our explicit goals are:

- *Present the need for DistAnt by relating our experience in conducting large scale e-Science experiments over heterogeneous grids and the lack of suitable existing deployment system (Section 2).*

¹ We define application deployment as the range of actions, including file transport, compilation, linking, installation and configuration, required to setup a program on a resource, ready for execution.

- *Present the challenges of application deployment over heterogeneous grids and infer solutions (Section 3).*
- *Discuss the implementation of our user-oriented application deployment system, DistAnt; focusing on how DistAnt enables the deployment of e-Science application over heterogeneous grids, supporting the full sequence of steps from compilation to execution (Section 4). And,*
- *Demonstrate how application deployment can enable large grid-scale e-Science experiments by presenting an e-Science deployment case study (Section 5).*

2. User-Oriented Application Deployment

The motivation for this research is based around expertise and experience in organizing and developing large scale e-Science grid experiments:

1. At the IEEE Supercomputing Conference 2003, our research group used Nimrod/G, to coordinate over 50,000 instances of the GAMESS chemistry package at up to 30 super computers in 10 countries [5].
2. At the 8th Pacific Rim Applications and Grid Middleware Assembly 2005, our research group used Nimrod/G to coordinate 2160 long running instances of the CCAM climate model at 6 super computers in 5 countries, to simulate Australian climate.

The development of both grids was done manually by computer scientists using available tools such as *ssh* and *ftp*. Application deployment represented a huge proportion of both the technical and organizational effort required to realize both experiments, demonstrating the complete lack of support for application deployment within grid middleware and the barrier it presents to scientists. In both cases, application deployment was further complicated by the heterogeneity of the experiment testbed.

This manual approach is undesirable for a number of reasons. First, deploying applications is a highly user intensive process and becomes more difficult with large grids. Tools such as *ssh* and *ftp*, which sufficed earlier, are not scalable because they are highly user intensive and specific to one resource. Second, the fundamental characteristics of e-Science applications and the grid are at tension: e-Science applications commonly have highly specific hardware and software dependencies, and highly specific installation and configuration procedures; conversely, grids are characterized as highly technically and organizationally heterogeneous. Each grid resource can present a different set of deployment problems. Third, applications compiled to native machine code require an individual build for each platform. Finally, resource access does not guarantee adequate resource authority or functionality to realize deployment requirements.

With continuous development of grid infrastructure, higher scientific reliance on computational resources and better resource availability, e-Science experiments, of the scale discussed, can become routine. Thus, providing middleware end-user support is essential, including activi-

ties which have remained highly user interactive such as application deployment. In response, we foresaw the need for a grid deployment system, coupling application development, deployment and execution.

2.1 Existing Infrastructure

Automated application deployment can be characterized from two opposing, but complimentary perspectives: The existing super-user system management perspective; and our user-oriented perspective. A super-user system management perspective implies a centralized mechanism to provide administration and configuration over systems within institutional control. Conversely, the user-oriented nature of the grid requires that users have authority to deploy applications on resources they have received access, but do not necessarily control. The grid espouses easier inter-institutional access to resources and therefore yielding a degree of central control is unavoidable. Furthermore, computational grid resource users require specific applications and relying on complete central control to provide these applications is not a grid scalable solution. Hence, our focus towards a user-oriented perspective of grid application deployment.

From a super-user system management perspective, automated application deployment is generally viewed as a part of configuration management, which encompasses the range of actions of transforming a blank machine into a functional system. A representative handful of configuration management systems are summarized below, including: a commonly used cluster management system, NPACI Rocks [7]; and research specifically targeting grid management, GridWeaver [8]. A more complete overview of such systems can be found at [3].

NPACI Rocks provides node installation from a bare machine and a configuration based on package dependencies. Using RedHat kick start files and Linux RPMs, NPACI Rocks completely reinstalls nodes from the operating system up when they have failed or when there is a requested change in their installation or configuration. Rocks is primarily regarded as a cluster management tool.

GridWeaver is the sum of two systems: The SmartFrog framework [9], which defines a framework for constructing configurable software systems; and LCFG [10], which provides a mechanism for Linux node installation from the operating system up. The two systems are mutually supportive technologies; LCFG provides a mechanism to completely rebuild a machine, while SmartFrog provides a component management framework which assumes underlying resources. Both systems are centrally managed.

These super-user-oriented configuration management systems do not cater for user requirements in a user-oriented, shared grid. For example, using NPACI Rocks, a change in the system description would result in a complete rebuild of the system, which is undesirable in a multi-user system. Nor is it appropriate for a grid environment where application testbeds are dynamic and

opportunistic. This super-user-oriented perspective, of the above described systems, is an evolution of cluster management where application deployment is largely a non issue. Users will typically only need to deploy applications once, on a shared file system, to instantiate their application on the entire cluster. In contrast, deploying an application over a loosely coupled set of grid resources requires an individual deployment, on each resource. Furthermore, an organizationally heterogeneous grid, has no one overarching super-user to control deployment issues such as dependencies, version issues or clashing deployments. Therefore, the impetus and responsibility for a successful application deployment must come from the user, with support from underlying middleware.

Neither an automated super-user perspective configuration system, nor the current user-oriented manual approach provides a solution to grid application deployment.

The Globus GRAM [2, 13] provides a single gateway for application specification and execution. It is significant because it provides a *user-oriented homogeneous* interface to underlying heterogeneous queue managers, successfully minimizing technical and organizational heterogeneity. It provides a model for our own deployment service design.

Application deployment can be viewed as an extension of application development. The entire process leading up to and including deployment requires a number of steps: compilation; linking; packaging; file transport; unpacking / installation; configuration and finally, application instantiation. The first steps, compilation, linking and packaging, are easily described and automated through a make system such as Make or Ant. Unpackaging, installation and configuration can also be easily provided through a build file system on the local machine. Ant [11] is an XML based procedural build-file system, written in Java. Ant build files consist of: *tasks*, which define an action; *targets* which procedurally group together *tasks*; *properties*; and a *project* which groups all elements. Ant targets can depend on other targets, which is used by Ant to create a workflow graph. Build file systems, such as Ant, do not extend their functionality to remote locations.

The GridAnt [12] system takes a similar approach to DistAnt; it utilizes Ant extensibility to consolidate multiple client side actions into workflows managed by the underlying Ant graph engine. GridAnt implements Ant tasks as an interface to common grid activities such as file transfer, job submission and information query. However, unlike GridAnt, our work is specifically focused towards application deployment; addressing both client and server side requirements and issues such as remote actions, configuration and resource heterogeneity.

A different approach to application deployment over heterogeneous grids is to enforce a homogeneous environment over the entire grid. Operating system level virtual machines have been suggested as a solution to providing an on demand virtual homogeneous environment [18, 19]. This approach to grid computing provides a homogeneous environment but does not attempt to solve the

inherent difficulty in deploying applications over a set of virtual or real resources.

The lack of suitable user-oriented application deployment system led us to develop DistAnt. In an earlier publication [3] on our work we demonstrated:

- A build file system extension to provide a procedural deployment description;
- The use of an application deployment service to perform deployment actions; and
- The success and viability of a user-oriented grid application deployment system.

The DistAnt architecture, presented in earlier publication [3], consists of three components:

- A flexible front end procedural deployment description to describe file transport, configuration and instantiation in the same way as compilation, linking and packaging in a Unix make file. It extends the Ant build file system with custom tasks, extending the build file paradigm to application deployment. DistAnt custom tasks provide an interface to the following *deployment actions*: file transfer, remote actions and application instantiation. For consistency *deployment actions* are detailed in Sections 4.2 and 4.3, along with newly introduced DistAnt tasks.
- A deployment client (DistAnt Client) to coordinate the deployment with remote deployment servers. It uses existing file transfer and instantiation mechanisms to coordinate deployment actions over a set of resources.
- A deployment server (DistAnt Server) which resides at remote locations servicing deployment actions. The DistAnt Server provides a homogeneous deployment gateway, regardless of underlying technical and organizational heterogeneity. It is responsible for providing and maintaining *deployment spaces*, specific directories created for user deployments. Each *deployment space* is uniquely referenced by a user defined string called a *deployment reference*. Each DistAnt Server is responsible for resolving a deployment reference to its deployment space, which might be located at different locations between resources, according to local policy. Deployment references allow a user to reference the same application deployment, over a set of resources, regardless of actual file system location.

DistAnt has been implemented independently of grid middleware, while a Globus Toolkit 4 WSRF [2] wrapper provides a middleware specific implementation. DistAnt uses GSI security [2] to authenticate and authorize users. Like GRAM, DistAnt relies on the underlying user account system and performs all deployment requests as the requesting user, using the Unix *sudo* utility.

Our earlier published work provides a framework for application deployment, but does not support application deployment over *heterogeneous grids* or unknown / uncharacterized resources. The following section discusses how a system can support application deployment over heterogeneous grids and introduces characterization of resources, and application dependencies.

3. Application Deployment System Design

This paper focuses on the broad range of non-hosted applications, including: legacy codes with legacy dependencies; native applications; or virtual machine applications such as Java or .NET codes. The following discusses the important characteristics of a system designed to deploy the range of non-hosted e-Science applications.

It is common for e-Science applications to depend on specific elements of their environment such as hardware, operating system, compiler, library or other software. This is particularly true of legacy applications, which are dependent on legacy environment attributes. This dependency structure can be termed as an applications dependency tree. e-Science applications commonly perform a typical *make configure* build step, which identifies system characteristics, locates dependency locations and makes appropriate decisions before a build or install is initiated. *make configure* is typically an environment validation step which reports missing dependencies and exits, relying on user intervention to fix missing dependencies. However, when dealing with many resources, user intervention is not scalable. It is also expected that similar dependency problems would reoccur on many resources. Therefore, an application deployment system should provide functionality to express an application's dependency tree, check for dependencies, and, if possible, proactively deploy missing dependent components or software.

It is the goal of our research, that users should be able to deploy applications with only absolutely limited information about grid resources. However, deploying an application to a completely unknown resource is counter-intuitive; some deployment decisions must be made at the client side. For example, specific platforms will require specific binaries. Therefore, the deployment system needs to perform a step which is analogous to *make configure*; querying for resource information and making appropriate deployment decisions. A homogeneous means of querying for resource information and characteristics is essential.

Many existing e-Science applications build under multiple platforms and environments. Such applications lend themselves towards grid deployment because they already contain build and deployment knowledge for different circumstances and environments. For example, an existing make file will contain knowledge to build on different platforms. Application deployment systems focusing on e-Science applications need to leverage the existing software engineering investment and deployment knowledge coded inside applications. Furthermore, the system should deploy applications with minimal changes to the existing application and its existing build file system.

e-Science applications might have a very complicated deployment and configuration sequence and it is expected that an application deployment system would allow users to write complex deployment routines. In this respect,

providing a front end build file system, with grid deployment constructs, is particularly important because it is analogous to the build file system architecture typically used to build applications.

3.1 Deployment Challenges over Heterogeneous Resources

Grids are characterized by a high level of both technical and organizational heterogeneity. Technical heterogeneity complicates application deployment due to different build and executable files, library dependencies, file systems and installation procedures. Organizational heterogeneity implies different administration, operating procedures and access policy; complicating application deployment due to different access, permissions, accounts and installation procedures. In terms of software deployment, the two most disparate approaches to dealing with both types of heterogeneity are to hide it under a virtual environment, or conversely, assume nothing about an environment and perform a complete configuration. We argue that the best solution to dealing with resource heterogeneity is to virtualize certain system attributes, while providing support to manage remaining heterogeneity.

Application deployment can benefit from a similar approach to that of Globus GRAM; providing a single gateway for application deployment and a homogeneous means of describing a deployment eliminates problems raised due to organizational heterogeneity. A homogeneous resource description and resource querying mechanism is important to discover information about resources and make appropriate deployment decisions.

File system structure and organization presents a problem for application deployment. Different operating systems organize their file systems differently, while identical platforms are commonly different due to local policies. Writing generic configuration procedures to execute on any remote resource would be impossible without virtualizing file system structure. This can be achieved by managing the file system space where applications are deployed. This allows an application directory to be addressed by a *deployment reference*, regardless of its actual file system location. It also provides security by sandboxing users within specific file system spaces.

Virtualizing the above described system attributes allows users to write and enact generic deployment routines over a range of heterogeneous resources. However, it is not in our interest to completely virtualize a grid resource. Therefore, a grid application deployment system should manage resource heterogeneity.

An effective approach to managing a heterogeneous grid is to sub-divide it into manageable *sets* of homogeneous resources, which can be treated individually. Dividing a complete grid into sets of resources (sub-grids), allows the user to write specific deployment routines for specific resource attributes. Grouping resources within sub-grids

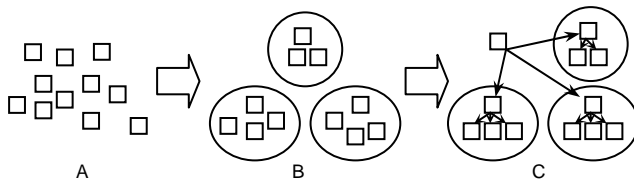


Figure 1. The logical sequence of steps to deploy an application using sub-grids: (A) showing uncharacterised resources; (B) resources grouped in logical platform specific sub-grids and finally (C) arrows showing the use of representative resources for compilation and redistribution to remaining resources.

asserts that a particular resource grouping shares attributes, important to a particular application deployment. For example, resources which share the same platform description might require a similar sequence of build steps, while all resources with an old version of a particular library might require a new library version. Applying set operators, such as intersection, to sub-grids provides additional expressive power. For example, the intersection of a set of Linux-x86 resources and resources requiring a library version upgrade produces the set of resources which require a Linux-x86 binary of the required library.

e-Science applications are commonly deployed from source code. However, a complete build from source on each individual resource is typically unnecessary. Grouping resources provides the basis of expressional power to build from source on one resource and redistribute binary to remaining resources. For example, using sub-grids a user can sub divide a resource based on platform. Defining a representative resource of each platform sub-grid means source can be compiled once on the representative resource and then redistributed to each resource (Figure 1).

4. DistAnt Implementation

This section describes the extended functionality to deploy e-Science applications over *heterogeneous grids* (Section 4.1), and an overview of all the deployment tasks supported by DistAnt (Section 4.2 and Section 4.3).

4.1 DistAnt: Deployment over Heterogeneous Grids

The following lists extensions to DistAnt functionality implemented to support e-Science deployment over *heterogeneous grids*, as discussed in Section 3:

- The ability to execute resource queries to characterise resources unknown resources. DistAnt supports two ways of retrieving resource information. First, the DistAnt Service publishes a homogeneous resource description, which can be queried. Second, remote tasks can be written to return resource properties;
- The ability to form sub-grids based on resource attributes, write deployment routines for specific sub-grids and choose representative resources;
- The ability to express and manage application dependencies; and

- A command line interface, providing a deployment interface for other systems, particularly Nimrod/G.

The functionality to query, characterise and group grid resources, and write deployment routines specific to sub-grids is provided through a set of custom Ant deployment tasks, which are discussed in the following sections. Other implementation details are detailed below.

The DistAnt Server publishes a homogeneous resource description which allows clients to write and enact queries against a resource. The resource information published by the DistAnt Server provides a description of the underlying resource and can be classified into three categories:

- General live resource information, such as platform and load information, provided by the Ganglia [14] *gmond* information gathering application.
- Deployment specific information, which includes locations of common applications and libraries. This information is gathered using a script.
- DistAnt specific information, which includes information about the user's previous deployments.

Grid middleware already offers information services which provide resource information; for example, Globus provides the Monitoring and Discover Service (MDS) [2]. However, publishing our own resource information is important for two reasons: First, we want to publish information which is specific to application deployment and this includes information which would typically be overlooked by the default information service; and second, through experience, we have learned that the default information service is commonly overlooked by resource administration, leading to frustration when resources do not support a baseline of information. DistAnt supports querying the Globus MDS, or other WSRF services, and it is expected that information published in the MDS could augment information published by DistAnt, or visa versa. The definition of a common resource description, for the purposes of application deployment, is a new approach in our research, and it is expected that the information published by DistAnt will develop and mature with use.

DistAnt has been integrated into Nimrod/G, providing an overall environment supporting the full range of activities required to create, deploy and execute and e-Science application. Nimrod/G has been augmented to dynamically resolve DistAnt deployment references, once they have been scheduled. Nimrod/G supports a new keyword, *resolve*, which interfaces to the DistAnt command line interface, providing a means of resolving global deployment references for resource specific deployment spaces.

4.2 Introduction to DistAnt Deployment Tasks

DistAnt extends the Ant build file system by defining three types of tasks: *deployment actions* which perform actions on remote resources; *descriptive tasks* which provide functionality to list resources, characterize resources, and form sub-grids; and *deployment management tasks*

which provide functionality to manage deployments.

These three types of tasks are building blocks for writing a procedural application deployment description. The relationships between the most common tasks can be summarized as follows:

- An initial set of resources is created using the *resource* and *grid* tasks;
- Sets of resources are characterized with attributes using the *property* and *queryproperty* tasks;
- Sub-grids are formed using the *subgrid* task, which executes provided *condition* tasks against resource attributes;
- *Deployment actions* are executed over a set of sub-grids, referenced by the *setgrid* task; and
- Applications are redistributed over the entire grid using the *freeze* and *melt* tasks.

4.3 Deployment Tasks

The following three actions are *deployment actions*, which were introduced in earlier publication. The remainder of this section introduces new DistAnt functionality.

deploy provides a means of transporting files to remote locations. DistAnt supports multiple file transfer mechanisms. For any particular resource, the actual file transport is chosen based on the mechanisms supported by the server and the preference list provided by the client. DistAnt currently supports GridFTP [15], Globus Reliable File Transfer [16] and Secure Copy (*scp*).

remote executes procedural actions on remote machines, allowing the user to write routines which unpack, build, install or configure applications at a remote location. All user properties and properties specific to the particular resource are passed to the remote DistAnt Service. Any properties set during the remote target execution are passed back to the client and set as resource properties. Passing Ant properties between client and server provides continuity and also allows the user to write routines which collect resource information or return a result.

remoteExe provides a mechanism for instantiating a deployed application. DistAnt supports multiple instantiation mechanisms, including the Globus GRAM and Unix command line. In each case, the DistAnt Service provides the information required to execute the deployed application. For example, when a GRAM instantiation is requested, the DistAnt Service creates and passes back a Resource Specification Language file for the client to use. Nimrod/G uses this mechanism to resolve an application deployment.

The remote user output created by *deployment actions* is streamed back to the DistAnt Client. The user has the option of displaying output with a resource identifying prefix, saving output to a resource log file, or both. Build or deployment errors are passed back to the DistAnt Client, which displays the error and reacts in two possible ways, according to user settings:

- First, an error on a single resource results in the entire

build process being aborted, or

- Second, the failed resource being dropped from the continuing deployment procedure.

Deployment actions are written within a *setgrid* task, which limits their scope to a specific referenced sub-grid. If the referenced sub-grid contains no resources, the nested actions are not performed. Alternatively, if the *setIfEmpty* attribute is set, the nested actions are *only* performed if the referenced sub-grid is empty. Therefore, the *setgrid* task allows users to write routines, specific to the presence or absence of resources which possess specific attributes.

The ability to model, characterise and group grid resources is provided by the following *descriptive tasks*:

resource provides a means of specifying, and if necessary, characterising, resources.

grid provides functionality to group listed resources or resources retrieved from an external source². The *grid* task allows the user to set a representative resource, which can then be referenced by appending the *.head* operator to the grid name, while the *.body* operator references all other resources within that grid. Using the nested task *head*, the user is able to specify conditions to find a suitable representative resource. If a grid contains no resources, or given conditional statements do not produce a *head* resource, the *setgrid setIfEmpty* attribute can be set to execute an appropriate procedure. An example of this is given in the case study in Section 5.

property defines resource specific properties. DistAnt properties extend the standard Ant property mechanism, allowing users to define a property name and value. However, unlike standard Ant properties, DistAnt properties reference resource specific properties. *property* can also be used to assign a property to an entire grid.

queryproperty is an extension of the *property* task, allowing the user to specify resource queries which are evaluated and assigned as resource properties. The basic *queryproperty* functionality resolves WSRF XPath queries against the current resource or any other specified WSRF service or resource. The *queryproperty* task is extendable to allow other query mechanisms.

The *subgrid* task allows the user to form sub-grids based on nested conditions. The *subgrid* task takes one attribute, *gridref*, the reference to an existing grid and wraps a set of *condition* tasks.

The *condition* task filters resources with specific resource properties. They can be grouped together if wrapped in Boolean operators. Condition tasks take three attributes: A name which refers to the resource property being tested; a value attribute which specifies the value being tested for; and a type attribute which refers to the type of condition. Condition types include: equal, unequal, lessthan, greaterthan, highest and lowest.

DistAnt *deployment management tasks* provide the ability to store and redistribute application spaces: The *freeze*

² For example, the *nimrodgrid* task is an extension of the standard *grid* task and retrieves a list of resources from the Nimrod/G system.

task allows the user to archive a deployment space, which is then stored on the remote resource or copied to the client machine. The *melt* task can then be used to un-archive the frozen application deployment on any particular resource. Used together, these two tasks allow the user to perform a deployment on a representative resource of a sub-grid and then redistribute the deployment to the other resources of that sub-grid. It can also be used to archive a deployment so that a fresh version is always available.

The following case study demonstrates how, *deployment actions*, *descriptive tasks*, and *deployment management tasks*, together provide the functionality to deploy an e-Science application over a heterogeneous set of resources.

5. Case Study

GAMESS is a widely available public quantum chemistry package [17]. It has been ported to a wide range of platforms, and thus is an ideal candidate for execution in the Grid. It also underpins one of the experiments described in Section 2 [5]. This section describes how the DistAnt system was used to deploy GAMESS over a grid testbed. It also describes how DistAnt integration into Nimrod enables a complete series of steps from application compilation, to application instantiation as part of a GAMESS parametric sweep experiment.

A heterogeneous deployment testbed was created and included the following resources: 5 × Linux x86; 1 × SunOS Sparc; 1 × MacOSX PowerPC and 1 × FreeBSD x86, distributed over two administrative organizations. The following details complicating characteristics of this testbed, and how the DistAnt system overcame them:

- Different platforms, each requiring a different configuration and native binaries to be built. The DistAnt build file written for this case study forms platform specific sub-grids, which each define a head resource where GAMESS is built. Native binaries are then redistributed to remaining body resources of each platform.
- Different compilers, the existing GAMESS build file uses different Fortran compilers depending on the target system. However, some resources were missing a Fortran compiler entirely, meaning a compiler must be installed. The DistAnt build file written for this case study selects the fastest resource with a Fortran compiler already installed. If no resources in a platform sub-grid have a Fortran compiler, then DistAnt chooses the fastest resource and deploys gcc-g77. This demonstrates both how DistAnt supports dependencies expression and how DistAnt can be used to select the most suitable resources for particular activities.
- Different file system structures. File system structure is virtualized by the use of deployment references to address GAMESS and the deployed gcc-g77.
- Different file transfer mechanisms. DistAnt coordinated file transfer using the available transfer method.
- Different access and firewall policies. The DistAnt Ser-

vice provides a trusted web services based deployment, mitigating access and firewall problems.

The following describes the sequence of tasks in the GAMESS deployment file, referencing Figure 2.

1. A deployment reference `GAMESS_1` is defined and a complete grid is created, `AllResources`. The entire grid is queried for resource attributes, including operating system, architecture, cpu speed and Fortran compiler, which are set as resource properties. The only grid information provided by the user is the resource domain names and grid service port numbers.
2. The resources in `AllResources` are divided into platform specific sub-grids. In Figure 2 a new sub-grid, `Linux-X86` is defined, based on the resources in `AllResources` with appropriate `operatingSystem` and `architecture` properties. The fastest resource, with a Fortran compiler, is assigned as the head resource.
3. If the previously defined head resource is nonexistent, then `gcc-g77` is deployed on the fastest platform specific sub-grid resource. This is a once only step per platform; subsequent deployments of GAMESS can be compiled using the existing `gcc-g77`.
4. The GAMESS source package is deployed to the heads of the platform specific sub-grids, where it is

```

<property name="namespace" value="\${GAMESS_1}"/>
<grid id="AllResources">
  <resource url=http://memnon.csse.monash.edu.au:8080/>
  <resource url=http://romulus.dstc.monash.edu.au:8080/>
  <queryproperty
    xPath="string(/**/*/*/METRIC[NAME='os_name']/VALUE)"
    name="operatingSystem" />
</grid>
<grid id="Linux-X86" dest="\${namespace}" head="any">
  <head>
    <and>
      <condition type="highest" property="cpuSpeed"/>
      <condition type="equal" property="fortran"/>
    </and>
  <head>
  <subgrid gridref="AllResources">
    <and>
      <condition type="equal"
        property="operatingSystem" value="Linux"/>
      <condition type="equal"
        property="architecture" value="x86"/>
    </and>
  </subgrid >
  <property name="platform" value="linux-pc"/>
</grid>
<setgrid gridref="Linux-X86.head" setifempty="true">
  <antcall target="fortran_install"/>
</setgrid>
<setgrid gridRef="Linux-X86.head ">
  <deploy src="\${package}" dest="\${namespace}"
    transport="gftp, rft" />
  </deploy>
  <remote dest="\${namespace}" target="gameSS_compile" />
</setgrid>
<target name="freeze" depends="formgrids">
  <setgrid gridRef="Linux-X86.head">
    <freeze dest="\${namespace}" file="\${platform}_freeze"/>
  </setgrid>
</target>
<setgrid gridRef="Linux-X86.body">
  <melt dest="\${namespace}" file="\${platform}_freeze"/>
</setgrid>

```

Figure 2. An excerpt of the DistAnt deployment file written to deploy GAMESS. For brevity, this figure has been strictly reduced to the Linux-X86 subgrid and most resources and properties have been removed.

pre-processed, compiled and linked. The actual compilation involves a number of steps which are initiated by the `gameSS_compile` target and are specific to the platform resource property, set in step 2.

5. Each head resource GAMESS deployment space is *frozen* to a platform specific filename at the client.
6. The previously frozen deployment space is *melted* to the remaining resources.

The deployment was performed with no changes to GAMESS code, very minimal changes to the existing GAMESS build file, and no changes to `gcc-g77` code or build file. The deployment over the complete testbed took 92 minutes. Both the MacOSX and FreeBSD platform subgrids required `gcc-g77` installed and the longest `gcc-g77` build took 54 minutes. The longest GAMESS build took 34 minutes. These times can be attributed to file copy and remote build, with an insignificant DistAnt overhead. Comparing this result against a manual deployment is difficult because it depends on the individual manual deployment scenario. One possible scenario assumes deployments are performed sequentially and the user manually installs on a representative resource and then redistributes binaries. In this case the aggregated time for this experiment would have been 212 minutes. In reality, a manual deployment could be quicker or slower depending on the difficulty and amount of concurrency a single user is able to manage. In either case, manual deployment is highly intensive and doesn't scale to large grids. On the other hand, whilst writing a DistAnt deployment file is an overhead, it enables easy non-interactive deployments and redeployments over a dynamic set of heterogeneous and uncharacterised resources.

After deployment, Nimrod/G was used to initiate a GAMESS experiment, based on a subset of that executed at Super Computing 2003. Importantly, the coordinating Nimrod/G description file (called a Nimrod plan) (Figure 3) is almost identical to one that uses manual deployment, and only requires one additional statement (*resolve*) to locate the provided *deployment reference*.

```
task main
  resolve ${GAMESS_1}
  node:execute ${GAMESS_1}/gameSS/cleanrunGMS
  cart_eth.A1=$A1.A2=$A2.B1=$B1.B2=$B2 > cart_eth.out
endtask
```

Figure 3. A subset of the Nimrod/G plan with the *resolve* keyword.

6. Conclusion

Grid middleware support of end users is essential to improve usability and promote wide scale adoption of the grid. We have demonstrated how DistAnt is able to support grid users, throughout the full sequence of events: application development, deployment and instantiation. Furthermore, this research is significant because users are able to deploy their application over a heterogeneous grid, with only absolutely minimal resource knowledge. This is particularly important in a dynamic grid environment,

where resources are added and removed on a regular basis, particularly e-Science where scientists use computational resources opportunistically.

DistAnt provides the opportunity to continue research in the area of e-Science grid management. We see deployment as one part of an overall grid runtime environment which guarantees application execution, which would guarantee application execution, and plan to continue research in this area.

References

- [1] I. Foster, C. Kesselman. "Computational Grids", Chapter 2, "The Grid: Blueprint for a New Computing Infrastructure", Morgan-Kaufman, 1999
- [2] "A Globus Primer. Describing Globus Toolkit 4", http://www-unix.globus.org/toolkit/docs/4.0/key/GT4_Primer_0.6.pdf
- [3] W. Goscinski, D. Abramson, "Distributed Ant: A System to Support Application Deployment in the Grid", Fifth IEEE/ACM International Workshop on Grid Computing, 2004
- [4] D. Abramson, J. Giddy and L. Kotler, "High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?", International Parallel and Distributed Processing Symposium (IPDPS), Cancun, Mexico, 2000, 520- 528.
- [5] W. Sudholt, K. Baldridge, D. Abramson, et al. "Parameter Scan of an Effective Group Difference Pseudopotential Using Grid Computing", New Generation Computing 22, 2004, 125-135.
- [6] J. McGregor, J. Katzfey, "NWP experiments with a variable-resolution conformal-cubic primitive equations model", CAS/JSC Working Group on Numerical Experimentation Report, Geneva, 1998.
- [7] G. Bruno, P. Papadopoulos and M. Katz., "Npaci rocks: Tools and techniques for easily deploying manageable linux clusters". Cluster 2001, 2001.
- [8] P. Anderson, P. Goldsack, J. Paterson, "SmartFrog meets LCFG - Autonomous Reconfiguration with Central Policy Control", 2002 Large Installations Systems Administration Conference, 2003
- [9] P. Goldsack, J. Guijarro, A. Lain, et al, "SmartFrog: Configuration and Automatic Ignition of Distributed Applications", HP Labs, UK, 2003, <http://www.hpl.hp.com/research/smartfrog/>
- [10] P. Anderson and A. Scobie. "LCFG: The Next Generation", UKUUG Winter Conference, 2002.
- [11] "The Apache Ant Project", <http://ant.apache.org/>
- [12] G. von Laszewski, B. Alunkal, K. Amin, S Hampton, and S. Nijssure. GridAnt-Client-side Workflow Management with Ant, 2002, <http://www.unix.globus.org/cog/projects/gridant/>
- [13] K. Czajkowski, I. Foster, et al, "Resource Management Architecture for Metacomputing Systems". IPPS/SPDP Workshop on Job Scheduling Strategies for Parallel Proc, 1998.
- [14] M. Massie, B. Chun, D. Culler, "The ganglia distributed monitoring system: design, implementation, and experience", Parallel Computing, July 2004.
- [15] B. Allcock, J. Bester, et al. "Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing", IEEE Mass Storage Conference, 2001.
- [16] W.E. Allcock, I. Foster, R. Madduri. "Reliable Data Transport: A Critical Service for the Grid", Building Service Based Grids Workshop, Global Grid Forum 11, June 2004.
- [17] Schmidt, M.W., Baldridge, et al. "General Atomic and Molecular Electronic-Structure System", J. Comput. Chem. 14, 1993, 1347-1363, <http://www.msg.ameslab.gov/gameSS/gameSS.html>
- [18] R. Figueiredo, P. A. Dinda, J. A. B. Fortes, "A Case for Grid Computing on Virtual Machines", Proc. International Conference on Distributed Computing Systems, May 2003
- [19] S. Adabala, V. Chadha, et al., "From virtualized resources to virtual computing grids: the In-VIGO system", Future Generation Computer Systems 21, Elsevier, June 2005