An Integrated Grid Environment for Component Applications

Nathalie Furmento, Anthony Mayer, Stephen McGough,
Steven Newhouse, Tony Field, and John Darlington

Imperial College Parallel Computing Centre, Department of Computing,
Imperial College of Science, Technology and Medicine, London, SW7 2BZ, UK
icpc-sw@doc.ic.ac.uk
http://www-icpc.doc.ic.ac.uk/components/ **

Abstract. Computational grids present many obstacles to their effective exploitation by non-trivial applications. We present a grid middleware, implemented using Java and Jini, that eliminates these obstacles through the intelligent use of meta-data relating to the structure, behaviour and performance of an application. We demonstrate how different problem sizes and selection criteria (minimum execution time or minimum cost) utilise different implementations for the optimal solution of a set of linear equations.

1 Introduction

Computational grids, federations of geographically distributed heterogeneous hardware and software resources, are emerging in academia, between national research labs and within commercial organisations. Eventually, these computing resources will become ubiquitous and appear transparent to the user, delivering computational power to applications in the same manner as electrical energy is distributed to appliances through national power grids [1]. This can only be achieved when an application is fully integrated with the *grid middleware* which provides access to the underlying hardware and software resources.

The grid middleware has to mask the heterogeneous nature of the resources and provide:

- A **secure execution environment** that is scalable in terms of the number of users and resources.
- **Virtual organisations**, formed through the federation of real resources. Resource owners will only contribute their resources to these federations if they are able to ensure access to their own local community [2].
- **Information** relating to the grid's resources, the application's performance and behaviour and the user's and resource provider's requirements.
- **Effective resource exploitation** by exploiting information relating to the structure, behaviour and performance of the application.

** Research supported by the EPSRC grant GR/N13371/01 on equipment provided by the HEFCE/JREI grants GR/L26100 and GR/M92455

C.A. Lee (Ed.): GRID 2001, LNCS 2242, pp. 26-37, 2001.

In this paper, we present an overview of a grid middleware that supports resource federation through locally defined access and usage policies and the exploitation of application meta-data to optimise execution. The paper makes the following contributions:

1. We describe a federated grid architecture, implemented using Java and Jini, which supports management of a collection of computational resources and which provides scalable secure access to those resources by registered users.
2. We present a component-based application development framework which associates performance meta-data with each component enabling composite performance models to be built for complete applications.
3. We show how application performance models can be used within the grid framework to determine optimal resource selection under minimum resources cost and minimum execution time constraints.

2 Federated Grid Architecture

2.1 Overview

To support our grid related research activities into application composition and resource exploitation, we have defined a middleware that supports the federation of hardware and software resources [3] as shown in Figure 1. Central to our grid architecture is the notion of a public Computational Community. It represents a virtual organisation (comprising individuals and organisations with a common goal) to which real organisations can contribute their resources. These are made available to all users, satisfying access control restrictions specified by the resource provider through the Computational Community. Information relating to the resources in the Computational Community (such as the current resource load, operating system, access and usage policies, etc.) are accessed through a published API or a GUI such as the Resource Browser by the user. At this level, the functionality of the Resource Browser is comparable to that of a federated virtual machine room [4].

The resources within an organisation are managed through a private Administrative Domain by the Resource Manager. Static and dynamic attributes on each resource provide information on demand to services in the Computational Community (e.g. the Application Mapper). This information service is critical to resource selection and updates can be pulled on demand to the services in the Computational Community. The Domain Manager acts as a gateway between the public and private areas of the grid middleware, the Administrative Domain and Computational Community, by enforcing the locally defined access and usage policies. Authentication of the resource request is delegated to the Identity Manager which is described in Section 3.1.

The Computational Community on its own provides a low-level homogeneous interface to the grid's heterogeneous resources that is similar to Globus's toolkit of services [2] and Legion's object model [5]. Our infrastructure provides a toolkit of software objects that can be extended and specialised by the user to provide

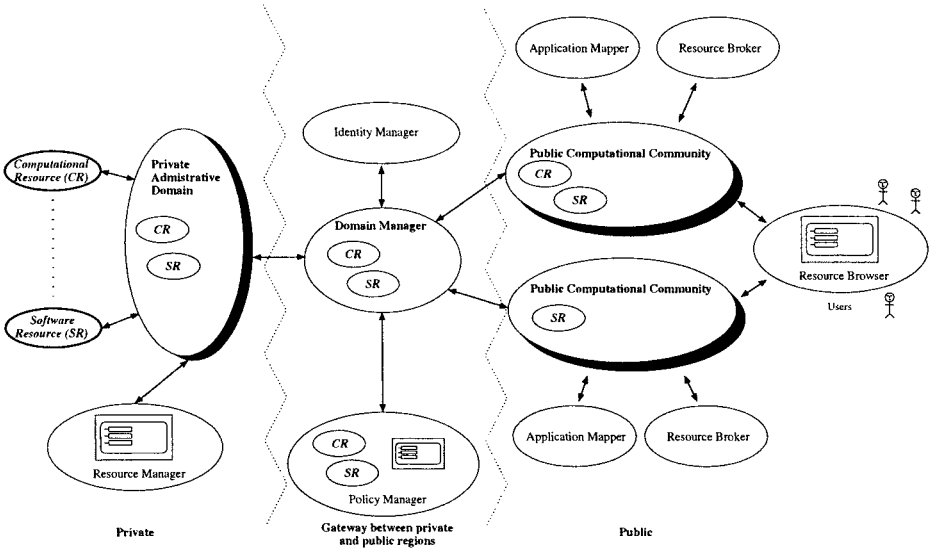

Fig. 1. Building a public Computational Economy through Federated Resources.

higher level services such as an Application Mapper (to generate optimal application decomposition) and a Resource Broker (to maximise throughput using computational economics).

2.2 Implementation

This architecture is being implemented using a combination of Java [6] and Jini [7]. Jini's leasing mechanism allows transient grid resources, represented by Java objects, to connect and re-connect over time while allowing unexpected failures to be handled gracefully. We use Jini's look-up server to provide dynamic registration of grid objects in the public Computational Communities and private Administrative Domains. This behaviour could also be achieved, for example, through the use of an LDAP server [8]. The initial stages of this work were described in [9] and are expanded in this paper. Other Java computing infrastructures such as Javelin [10], Popcorn [11] (using computational economics) and those based on Jini [12] have not addressed the usage and access policies which are essential in building virtual organisations.

3 Security Infrastructure

3.1 Authentication

All user interaction with the private Administrative Domain from the Computational Community has to pass through the Domain Manager. The authentication of the users (their identity, organisation and group memberships) is delegated

```
<identificationOrganisation publish="jini://jini.doc"
    name="ICPC" keystore="./data/security/keystore.ICPC">
  <group name="ala">
    <user name="aem3Signed"/>
    <user name="nfurSigned"/>
    <user name="sjn5Signed"/>
  </group>
  <group name="ra">
    <user name="asm100Signed"/>
    <user name="sjn5Signed"/>
  </group>
  <knownOrganisation name="rwth"/>
  <knownOrganisation name="inria"/>
</identificationOrganisation>
```

Fig. 2. Example of a Identity Manager XML configuration file.

by the Domain Manager to a trusted Identity Manager. The user's identity and organisation are encapsulated within an X509 certificate [13] associated with the resource request. No transport level security is currently implemented, however several SSL implementations exist for Java. The Identity Manager therefore has two roles: to authenticate users and to act as a Certification Authority by issuing certificates to its local user community. In doing so, it maintains a list of local users which is used to group individuals within the organisation and to mark certificates as revoked if an user leaves an organisation.

The configuration is maintained as a disk-based XML schema which lists the trusted organisations, local individuals and the membership of each group within the organisation. The example in Figure 2 shows an organisation (`ICPC`) with 4 users split into 2 groups (`ala` and `ra`). Users from two trusted non-local organisations (`rwth` and `inria`) are accepted by the Identity Manager. The certificates of the trusted organisations are exchanged through off-line resource sharing agreements or are a recognised Certification Authority.

A user's X509 certificate contains the public parts of their own key signed by a succession of other organisations (Certification Authorities). The Identity Manager compares these signatures with those of the local organisation and the known trusted organisations (see point (1) in Figure 3). If no match is found, the authentication fails. Following validation of the organisation's signature on the user's certificate by the local Identity Manager, authentication then passes to the known Identity Manager of the trusted non-local organisation to retrieve the user's group memberships and to validate if the user is still part of the organisation (see point (2) in Figure 3). These responses (see point (3) in Figure 3) are stored in a time-limited cache to reduce inter-organisation authentication traffic. Finally, the Identity Manager returns to the Domain Manager an authenticated user, groups and organisation identities (see point (4) in Figure 3). The Identity Manager sits outside the private Administrative Domain but is administered, like the Domain Manager, by the local organisation and will only interact with known and trusted Identity Managers. This ensures that no user interaction takes place with the private Administrative Domain until the user is authenticated and the request authorised.

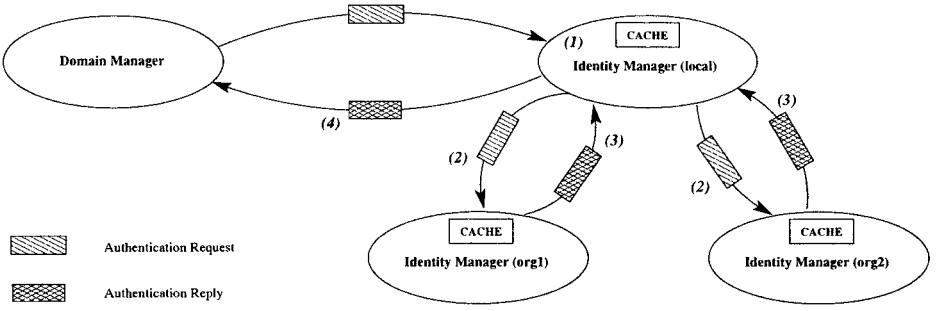

Fig. 3. Authentication Flows between the Domain Manager and the Identity Manager(s).

3.2 Authorisation

The Domain Manager uses the authenticated user, groups and organisation entities to determine if the request complies with the local access and usage policy. A simplified access control policy has already been implemented that accepts or rejects a request on the basis of the user, group or organisation [14]. The policy can be applied throughout the week or be restricted to specific periods. This functionality will be extended to include polices based on the number and identity of remote users (e.g. no more than 5 users from the Physics department) or the current resource load (e.g. if the utilisation is less than 50% then accept remote users). The Domain Manager allows the formulation of fine-grained control policies to specify how local and non-local users are permitted to use the resources.

3.3 Execution

After the Domain Manager has authenticated and authorised the request, it is passed to the resource. A request to access a resource's dynamic attribute (defined in Section 2.1) is fulfilled by the Domain Manager invoking the attribute's service function (e.g. Remote Method Invocation server to return the current queue status) and passing the result back to the Computational Community. If the request requires access to a computational resource, a session is provided through a secure shell (ssh) implementation [15]. The Domain Manager uses a pool of standard user accounts to provide access to the computational resource. A session can be mapped to a specific account (if the incoming request is from a local user) or to a generic account (from the available guest accounts) if one is available. The guest accounts are stateless with files staged in and out of the account before and after each session depending on the user's request. Any authenticated and authorised user from a remote organisation is able to gain access to a computational resource without having a specified account on each of the available resources. This is essential if any grid computing system is to scale.

On startup, all jobs have access to their own execution plan. This allows user specified configuration information to be passed into the application. If a job

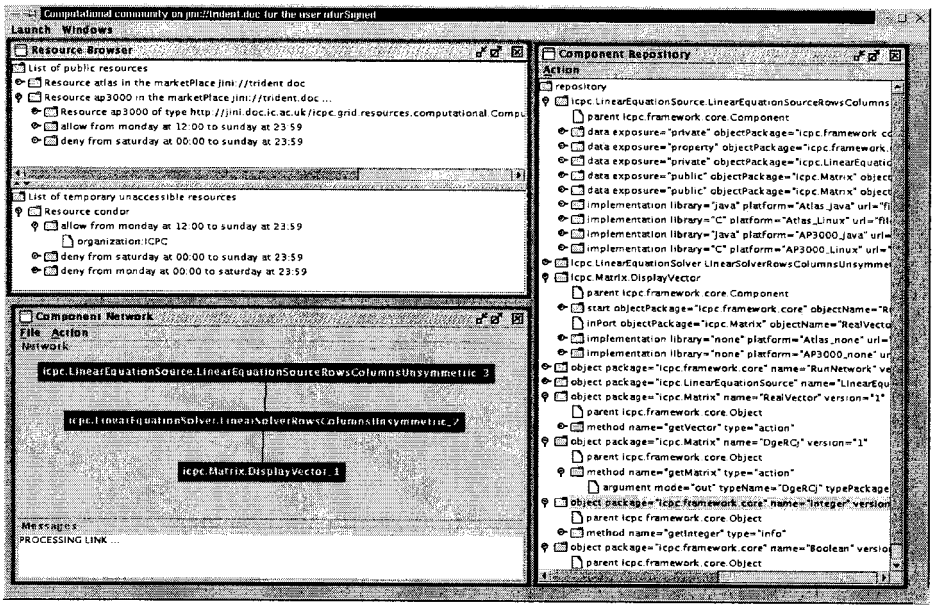

Fig. 4. Building a Component Network from the Resource Component Repositories.

spans multiple resources in potentially several Administrative Domains, it will be able to extract the location of the other distributed components from the execution plan and initiate contact with the other components.

4 The Computational Community

The Resource Browser is the primary user interface to the resources within the Computational Community. See the top left quadrant of Figure 4. On start up, the Resource Browser uses Jini's automatic discovery and join protocols to find a local Computational Community. Alternatively, the Jini URL relating to the Computational Community can be specified explicitly. Depending on the user's identity, a resource within the Computational Community will be:

1. *Available* - it will be shown with all of its public attributes and resource related information. The users are able to obtain the current value of each displayed dynamic attribute or to request a direct connection to the resource.
2. *Temporary unavailable* - the access policies relating to the resource, which the user is currently unable to satisfy, will be shown. The users therefore know when they will be allowed to access the resource.
3. *Always unavailable*, no information about this resource is shown to the users as they will never be permitted access to the resource.

For users to access resources within the Computational Community through a Resource Browser, they need a valid X509 certificate. This is obtained by:

```
<application>
 <network>
  <instance componentName="DisplayVector" componentPackage="icpc.Matrix" id="1"/>
  <instance componentName="LinearSolverRowsColumnsUnsymmetric"
        componentPackage="icpc.LinearEquationSolver" id="2"/>
  <instance componentName="LinearEquationSourceRowsColumnsUnsymmetric"
        componentPackage="icpc.LinearEquationSource" id="3">
   <property name="unknowns">10</property>
  </instance>
  <dataflow sinkComponent="2" sinkPort="unknowns"
        sourceComponent="3" sourcePort="unknowns"/>
  <dataflow sinkComponent="2" sinkPort="matrix"
        sourceComponent="3" sourcePort="matrix"/>
  <dataflow sinkComponent="2" sinkPort="vector"
            sourceComponent="3" sourcePort="vector"/>
  <dataflow sinkComponent="1" sinkPort="vector"
            sourceComponent="2" sourcePort="solution"/>
 </network>
</application>
```

Fig. 5. The XML definition of a Component Network.

1. Generating a certificate and the associated Certificate Signature Request (CSR) with a Certificate Manager tool,
2. Sending the CSR file to the administrator of their local organisation (or Certification Authority),
3. Importing the reply - a signed certificate - into their own certificate database.

The definition and membership of groups is currently controlled by the organisation's local administrator and maintained within the Identity Manager. To ensure unique group names between organisations, each name is appended with the organisations domain name e.g `staff@doc.ic.ac.uk`.

Any organisation can contribute a resource to a Computational Community (if it is already registered within their private Administrative Domain) by supplying their Domain Manager with the URL of the Jini look-up service and by defining their local access and usage policies.

5 Application Specification

5.1 Component eXtensible Markup Language

The key to the effective mapping of an application to heterogeneous computational resources is an understanding of its structure and behaviour. Application structure is exposed through the use of a component-based design pattern in its specification. To support this aspect of our work, we have defined a Component eXtensible Markup Language (CXML) that encapsulates meta-data describing the application structure. There are three distinct classes of meta-data described by the CXML.

Application Specification Document. This CXML document represents an application, consisting of component instances and connectors indicating the

composition of the instances (through a simple port mechanism) and of their properties. Figure 5 shows an application network with three component instances, one with a user customised property, that declares the data flows between components.

Repository Data: Interfaces & Behaviour. Each abstract component type is defined by its interface and behaviour. The interface indicates how the component may be composed with others, and what data is accessible. The behaviour of the component is the explicit context dependencies of the component's methods, i.e. a limited semantics of the dependencies between a component's ports. This information together with default values for properties is described in CXML and stored within the repository.

Repository Data: Implementations. Alongside the abstract component definitions is the meta-data for the component implementations. This includes binding code and performance models relating execution time and memory usage to data size for the implementation's methods.

The details of both the component mechanism and the CXML used to encapsulate the component meta-data are discussed in [14, 16]. The separation between component interface and implementation allows the composite application to be deployed across multiple platforms, and enables the selection of the most appropriate implementation for the given resource environment. The choice of implementation and platform is determined by performance modelling. In general an application is an assembly of fine grain components, it is possible to build a composite performance model of a complete application by examining the behaviour specification of the components and composing their individual performance models.

5.2 Composition

We have developed a visual programming environment that enables users to define their component application. The components and implementations installed on a local resource are made available to the Computational Community through a dynamic attribute. The current contents of each local Component Repository (provided by the dynamic attribute) are used to build a global Component Repository representing all the available and accessible components within the Computational Community. This community-wide Component Repository allows the users to build their application by dragging the components from the "Component Repository" window to the "Component Network" one. (Figure 4)

The users are able to build an application from the components in the Component Repository without having to know on which resources the implementations exist. If the same component is available with different implementations on more than one resource, the component will only appear once in the "Component Repository" window. The composed application is matched to the available implementations (and their corresponding resources by the Application Manager). This process is discussed in the following section.

```
<application>
 <network>
  <instance componentName="DisplayVector" componentPackage="icpc.Matrix" id="1">
   <implementation library="java" platform="java" url="file:."
        className="icpc.DisplayVector" resource="hotol@jini://jini.doc"/>
  </instance>
  <instance componentName="LinearSolverRowsColumnsUnsymmetric"
        componentPackage="icpc.LinearEquationSolver" id="2">
   <implementation library="C" platform="Linux" url="file:."
        className="icpc.LinearEquationSolver.DgeRCluC_EO"
        resource="hotol@jini://jini.doc" />
  </instance>
  <instance componentName="LinearEquationSourceRowsColumnsUnsymmetric"
        componentPackage="icpc.LinearEquationSource" id="3">
   <implementation library="C" platform="Linux" url="file:."
        className="icpc.LinearEquationSource.DgeRCddC_EO"
        resource="hotol@jini://jini.doc"/>
   <property name="unknowns">10</property>
  </instance>
  <dataflow> .... </dataflow>
  ....
 </network>
</application>
```

Fig. 6. The Execution Plan for a Component Network.

6 Application Execution

The preceding sections have demonstrated how we can build up meta-data relating to the resources in the Computational Community, the structure of a component application and the available component implementations. We are now able to exploit this rich meta-data to optimally distribute the application within the Computational Community.

This process starts with the user placing the CXML definition of their application (as shown in Figure 5) into the Computational Community for analysis by the Application Mapper(s). The Application Mapper constructs a global Component Repository from the resources in the Computational Community which it matches with the CXML application definition to find implementations that produce viable execution plans. The execution plan shown in Figure 6 extends the definition of the component network by adding to each instance the selected implementation. In this example, the Application Mapper has selected a C implementation of each component on the Linux machine `hotol`.

The execution plans are scored according to both the user's criteria (e.g. minimum execution time) and resource provider's criteria (e.g. resource access and loading). The users can either select one of the better execution plans generated by the Application Mapper or automatically execute the best plan.

Our initial implementation of an Application Mapper performs an exhaustive search on all feasible implementation options on all of the available resources. We find the best combination of component implementations for this specific instance of the application by incorporating information such as the specified problem size [14, 16]. An alternative approach to application partitioning, used by the AppLeS project, is to define the resource and user constraints as a linear programming problem and maximise a user supplied objective function [17].

Both approaches can also be used to configure the best implementation to yield optimal performance by selecting, for example, matrix blocking size and parallel data partitioning.

7 Demonstration

We illustrate the use of the complete grid environment with an example for solving a system of linear equations. Such computation regularly occurs in a wide range of scientific computing problems. When this application is represented as a component system, it consists of three component instances: a simple linear equation generator, a solver, and a display component. These components are connected as shown in the lower left quadrant of Figure 4.

Having defined the component network, the Application Mapper automatically selects the best algorithm, implementations and resource from those that are available within the Computational Community for various problem sizes. The AP3000 is an cluster of Sun SPARC Ultra II 300MHz processors running Solaris 2.6 with a proprietary Fujitsu interconnect. Atlas is a cluster of EV67 Alpha processors with a proprietary Quadrics interconnect running Tru64 Unix. Hotol is a 800MHz AMD processor Linux machine running Redhat 6.2.

Two cost models were evaluated. In both cases the relative costs of the different systems were based on the cost of a single processor on the AP3000. The Linux machine (`hotol`) was set to 80% of the cost of a processor on the AP3000 in both models to bias small jobs towards this system. The cost of each processor on Atlas was set to 475% (Cost Model 1) and 425% (Cost Model 2) of the cost of a processor on the AP3000. The best execution time and resource cost required to solve a system of linear equations is shown in Figures 7 and 8 (key to symbols in Table 1) for two different selection policies: minimum execution time and minimum resource cost. It can be seen in Figure 7 that different resource costs (an inevitable consequence of any demand driven computational economy) result in the selection of different implementations on different resources if the overall costs are to be minimised. The two graphs show that minimising cost produces significantly longer (on average 5 times) execution times while minimising execution time increases the resource cost (on average 1.6 times) with the current cost models.

Symbol	Source Implementation	Solver Implementation	Processors
◇	C	Hotol, BCG	1
+	Java	AP3000, BCG	4
□	Java	AP3000, BCG	9
×	Java	AP3000, BCG	16
◯	Java	Atlas, BCG	1
△	Java	Atlas, BCG	4
⋆	Java	Atlas, BCG	9
○	Java	Atlas, BCG	16
•	C	Atlas, BCG	16

Table 1. Key to Graphs.

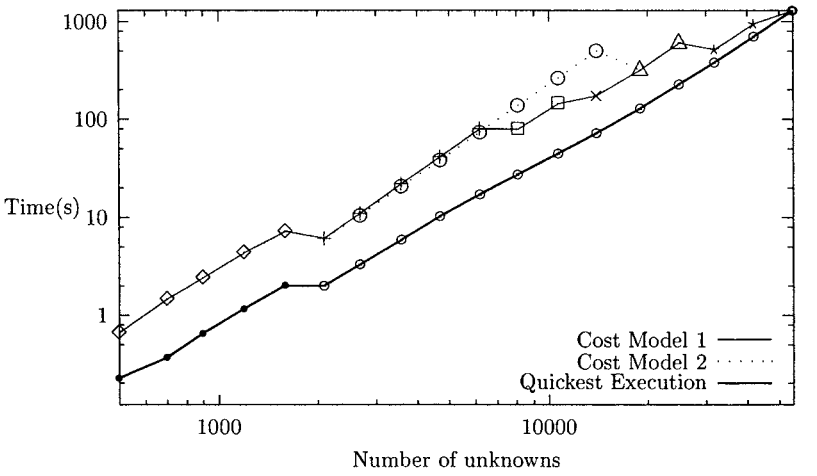

Fig. 7. Execution Time for the Composite Application Network with Different Number of Unknowns.

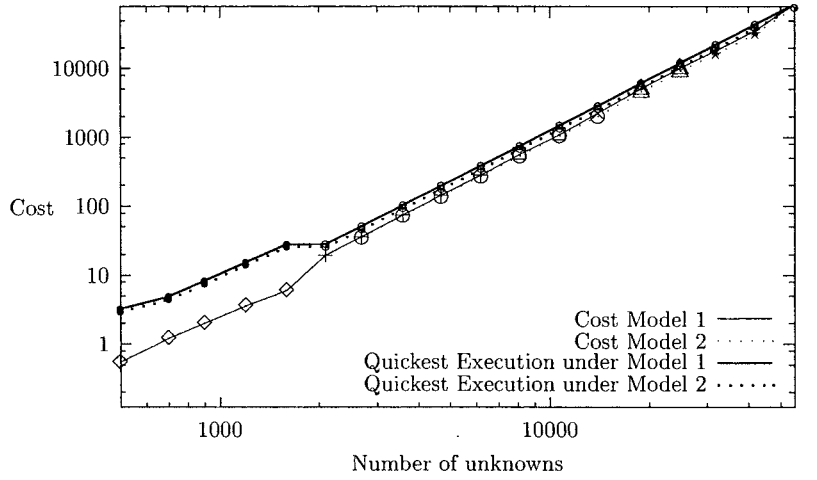

Fig. 8. Execution Cost for the Composite Application Network with Different Number of Unknowns.

8 Summary & Future Work

We have shown how the integration between the application and grid middleware is essential in the effective exploitation of grid resources from both the user's and resource provider's perspective. We use Java and Jini to build a fault tolerant grid middleware that supports the federation of resources into a Computational Community while the local organisation retains full control of access and usage policy. We have illustrated how the optimal solution of a set of linear equations

with various implementations is dependent on both the resource selection criteria and the overall problem size.

The Computational Community will be extended to include an Application Mapper based on solving the resource allocation problem as an integer linear programming problem (e.g. [17]). We will also be implementing Computational Economics through a Resource Broker and banking system to explore resource pricing within high throughput federated grid environments. This work will continue with motivating applications in high throughput computing and distributed supercomputing applications.

References

1. I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, July 1998.
2. I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Intl. J. Supercomputer Applications*, 2001.
3. S. Newhouse and J. Darlington. Computational Communities: A Market Place for Federated Resources. In *High-Performance Computing and Networking, HPCN*, volume 2110 of *LNCS*, pages 667–674, 2001.
4. The Virtual Machine Room. `http://www.ncsa.uiuc.edu/SCD/Alliance/VMR/`.
5. A. S. Grimshaw and W. A. Wulf *et al.* The Legion vision of a Worldwide Virtual Computer. *Communications of the ACM*, 40:39–45, 1997.
6. K. Arnold, J. Gosling, and D. Holmes. *The Java Programming Language.* Addison-Wesley, 3rd edition, 2000.
7. Sun Microsystems. Jini(tm) network technology. `http://java.sun.com/jini/`.
8. OpenLDAP Project. `http://www.openldap.org`.
9. N. Furmento, S. Newhouse, and J. Darlington. Building Computational Communities for Federated Resources. Accepted for Euro-Par 2001.
10. M. O. Neary, B. O. Christiansen, P. Cappello, and K. E. Schauser. Javelin: Parallel computing on the Internet. In *Future Generation Computer Systems*, volume 15, pages 659–674. Elsevier Science, October 1999.
11. O. Regev and N. Nisan. The Popcorn Market - Online Markets for Computational Resources. In *The 1st Int. Conference On Information and Computation Economies. Charleston SC*, 1998.
12. Z. Juhasz and L. Kesmarki. Jini-Based Prototype Metacomputing Framework. In *Euro-Par 2000*, pages 1171–1174, 2000.
13. Sun Microsystems. X.509 certificates. `http://java.sun.com/products/jdk/1.2/docs/guide/security/cert3.html`.
14. N. Furmento, A. Mayer, S. McGough, S. Newhouse, and J. Darlington. A Component Framework for HPC Applications. Accepted for Euro-Par 2001.
15. Mindterm, the java secure shell client. `http://www.mindbright.se/mindterm/`.
16. N. Furmento, A. Mayer, S. McGough, S. Newhouse, and J. Darlington. Optimisation of Component-based Applications within a Grid Environment. Accepted for SuperComputing 2001.
17. H. Dail, G. Obertelli, F. Berman, R. Wolski, and A. Grimshaw. Application-Aware Scheduling of a Magnetohydrodynamics Application in the Legion Metasystem. In *The 9th Heterogeneous Computing Workshop*, May 2000.