

Computational Communities: A Marketplace for Federated Resources

Steven Newhouse and John Darlington

Imperial College Parallel Computing Centre
Department of Computing, Imperial College of Science
Technology and Medicine, London, SW7 2BZ, UK

Abstract. We define a grid middleware comprising federated resources that facilitates a globally optimal mapping of applications to the available resources while satisfying the goals of both users and resource providers. Applications are annotated with performance and behavioural information to enable the ‘best’ resources to be found automatically. A computational currency is used by resource providers and consumers to express their goals (e.g. completion time, resource utilisation, etc.) enabling a globally optimal mapping of applications to resources. We describe a prototype implementation of this architecture using Java and Jini.

1 Introduction

The accelerating proliferation of high-performance computing resources and the emergence of high-speed wide area networking has led to much interest in the development of Computational Grids. A Computational Grid is defined as the combination of geographically distributed heterogeneous hardware and software resources to provide a ubiquitous transparent computing environment [1]. Such infrastructures are gaining acceptance outside the traditional high performance computing community as computational and data intensive applications become commonplace in science and commerce. Early experiments in Grid construction have generally involved the explicit connection of supercomputers or scientific instruments, requiring a high degree of expertise and involvement from resource providers and users [2].

The federation of heterogeneous resources under the administrative control of different organisations is achieved through a *grid middleware* that must mask any heterogeneity and provide:

- **Information.** Effective application scheduling requires information on the available hardware, software, storage and networking resources.
- **Security and Control.** Organisations will only federate their resources if they retain control and are able to ensure the needs of their local users.
- **Effective Resource Exploitation.** The *best* resources for an application will depend on the user’s and resource provider’s goals.

To achieve a transparent and ubiquitous grid computing environment it is necessary to automate resource selection, but to do so within the constraints and goals of the user and resource provider. We use a computational currency to enumerate these goals allowing us to achieve a balance between the needs of the individual and the community. For example, a user can choose to pay for better resources to reduce their execution time while a resource provider can select a job mix that maximises their revenues and utilisation.

2 A Computational Economy

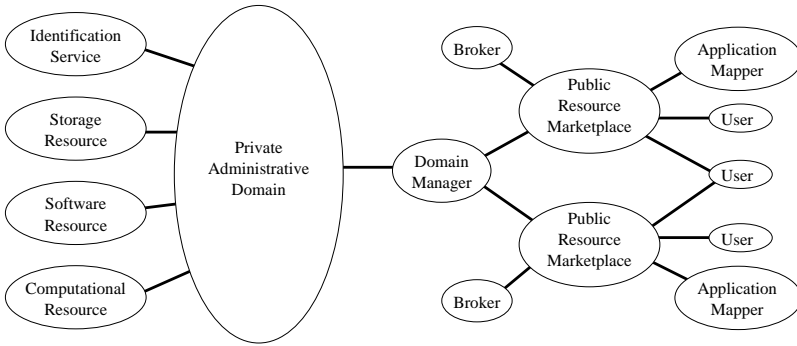


Fig. 1. Building a Resource Marketplace through Federated Resources.

2.1 Overview

Our federated computational economy has four major components that interact through a public resource marketplace (see figure 1):

- **Organisations** contribute resources under a locally defined access control policy to a public resource marketplace. Any organisation can participate by informing their local Domain Manager of the ‘well known’ URL of the marketplace’s Jini Lookup Service and the resources they wish to contribute. Within a local area network the lookup service can be discovered using Jini’s automatic protocols.
- **Users** are able to utilise the resources and services within a marketplace by connecting to its Jini Lookup Service through a ‘well known’ URL or by automatic discovery within a local area network. To access these resources the user’s organisation must have an established ‘trust’ relationship with the resource provider’s organisation. This may range from a known public key certification authority to a specific resource sharing agreement.

- **Application Mappers** generate a set of execution plans matching the application’s requirements to the resources that are currently available. These represent feasible execution strategies optimised with regard to performance but may not make the *best* economic use of the resources.
- **Brokers** negotiate between organisations and users to find the *best* economic execution plan from those generated by an application mapper.

An organisation’s resources are managed through the local Administrative Domain. These resources are made publicly available through the Domain Manager which places this information into one or more public resource marketplaces and enforces the specified local access control policy.

This model allows organisations to federate when they see mutual benefit in doing so by advertising their resources, alongside others in that community, in a common marketplace. It is important to note that the same resources may appear in different marketplaces with different constraints and that users are able to advertise their needs in several marketplaces. This ensures there is no single point of failure in the provision of resources and allows competition between different marketplaces which may have different broker or mapper implementations.

2.2 Building a Computational Community

Local resources will only be federated into a larger computational community if the usage conditions governing remote access are explicit and strictly enforced. For instance, in an academic environment staff may be given a higher priority than students but a student with an upcoming deadline may be given higher priority than most staff. Likewise, remote users may only be allowed to use the resources if they are idle but collaborators may be given priority over other remote users. Being able to express these usage policies is a key motivation for our infrastructure.

Our resources may have both static (e.g. operating system release, architecture, etc.) and dynamic attributes (e.g. current load, available licences, etc.). These attributes are advertised in the computational community and used during resource selection. The resources are registered as Java objects within the Jini lookup service. Persistence of these attributes (between power cycles and unexpected failures) is maintained through an XML syntax that describes the resource and its characteristics.

- **Computational Resources.** We currently access our own local computational hardware through a batch scheduler abstraction with implementations for NQS, PBS [3] and Condor [4]. Each computational resource executes its own segment of an XML defined execution plan passed to it by the Domain Manager.
- **Storage Resources.** The user must be able to automatically and securely access their storage space from any location.
- **Software Resources.** Our current implementation only represents unlimited use software libraries but the execution of a licensed library or application has to be scheduled in the same manner as a computational resource to ensure that a licence is available.

These resource abstractions could encapsulate access to existing infrastructures such as Globus [2] or Legion[5].

The *Domain Manager* allows organisations to contribute their resources into a federated computational community while retaining fine-grained control of how non-local users are permitted to use these resources. As the only route between the private and public areas of the middleware it provides:

- **Authentication.** Authentication of the user, groups and organisation uses a public key infrastructure and is delegated to the Identification Service. Each organisation may act as its own certification authority (CA) and define which organisations (and CA’s) it trusts to use its resources.
- **Authorisation.** Access to individual resources is controlled through conventional access control lists that recognise three entities: individuals, groups and organisations. This allows the Domain Manager to implement fine-grained access control policies governing resource usage.
- **Promotion.** Resources and their associated access control polices are published in one or more computational communities by the Domain Manager. Published information can be restricted to hide specific resources and the details of the local access policy.
- **Execution.** The Domain Manager validates all resource requests in a user’s execution plan before passing them onto the individual resources.

2.3 Resource Discovery and Selection

Effective automatic resource selection requires information relating to the performance of the application on different architectures (encapsulated within a performance model) and the requirements of the user. This is in addition to any source code or binary annotations such as those regarding execution environment that are currently used in the Condor ClassAds ‘matchmaking’ system [6]. We are extending our previous work in program composition using skeletons [7] to defining HPC applications as a network of software components which will automatically generate the application’s structure and overall performance model from its components [8].

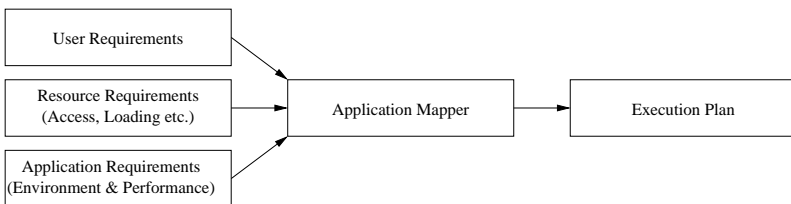


Fig. 2. Application Mapper.

The *Application Mapper* automates the user's selection of the *best* resources from those that are currently available. It uses the requirements specified by the user (e.g. job completion time), the application (e.g. run-time environment and execution time) and the resource provider (e.g. resource access and loading) to generate a set of viable execution plans for final selection by the user (see figure 2). Multiple Application Mappers can provide the user with a variety of feasible execution plans generated using different approaches. One approach to application partitioning is to define the resource and user constraints as a linear programming problem and maximise a user supplied objective function [9].

2.4 Computational Supply and Demand

The Application Mapper ensures effective resource selection and utilisation for an individual user's application but it does not ensure effective resource utilisation for the resource provider. For example, two applications request the use of a 16 processor PC cluster and that from each application's performance model the optimal number of processors is determined to be nine. If the first job starts with nine processors, the second job would have to wait until the first job has completed and nine processors become available. An alternative approach is for both applications to use eight processors and both to begin immediately. The latter situation is preferred by the resource provider as it increases utilisation but it may not always be feasible (or desirable) to adjust a user's application in this way.

The wishes of the resource provider (to maximise their resource utilisation) are elegantly expressed by charging a user different costs for different resource configurations. It may be acceptable to the user to use eight processors if the cost were considerably less than that of using nine processors (as the resource provider would be charging for the seven 'wasted' processors). To support resource trading we will use a trusted computational currency as an exchange medium. Resource providers will need to recognise and convert between several currencies which may be backed by a specific resource provider or generic micropayment scheme.

The *Resource Broker* negotiates a cost for the execution plans with all valid resource providers and presents these to the user. The resource provider prices the execution plan according to its own economic priorities (which may be dependent on the individual, group or organisation wishing to use their resource) and attempts to maximise their resource utilisation by consideration of, say, revenue stream or job throughput. Job priority, from the perspective of either a user and resource provider, is elegantly and simply expressed through these market mechanisms. By maximising the pay-off functions we can find the 'best' global allocation of resources to jobs in the computational community by balancing the needs of the users, the applications, and the resource providers over all requests rather than each individual request. Our framework makes no attempt to define what is the 'best' pay-off function other than it will be defined by the user and resource provider in terms of time and money.

3 Implementation

An initial proof of concept prototype of this architecture has been completed at Imperial College using a Java and Jini environment [10]. The architecture described in this paper represents an extension of this work. We exploit Java's portability and its rich API's to simplify many of the development tasks [11]. We use Jini as the primary service infrastructure as it supports dynamic registration, look-up and connection between the Java objects that represent our grid services and resources [12]. As all grid resources are effectively transient this ability to connect and reconnect over time is a highly desirable feature. The Jini leasing mechanism also allows unexpected failures to be handled gracefully. We use the look-up server to represent the public resource marketplaces and private administrative domains. This implementation is currently being extended to meet the needs of our user community in High Throughput Computing (e.g. Particle Physics, Bioinformatics and Medical Image) and Distributed High Performance Computing applications (e.g. solar coronal mass ejection simulations and coupled fluid-structure acoustics).

4 Related Work

Our approach is a combination and logical extension of two leading grid infrastructure projects: Globus and Legion. Globus provides a toolkit of services (information management, security, communication etc.) to integrate heterogeneous computational resources into a single infrastructure [2]. Legion uses a uniform object model for both applications and resources allowing users and administrators to subclass generic interfaces to their specific local needs [5].

Java has been used to provide a homogeneous distributed computing environment across heterogeneous resources (e.g. Javelin [13] and other projects). While Jini has been used to form a meta-computing infrastructure [14]. However, these projects have not yet addressed the policy issues regarding the access of remote users to local resources, which is fundamental in our approach.

The skeleton approach to program composition defines an application as a composition of components which are assembled using pre-defined structural forms of known semantics (e.g. pipe, farm) [7,15]. The mapping of these compositions onto target architectures is guided by analytical performance models, developed with each component, allowing decisions regarding efficient implementation to be made quantitatively and systematically. Our experiences with structured coordination languages is now being used in the context of conventional software components by their annotation with XML encoded meta-data relating to how they can be used, deployed and perform [8]. This compositional approach yields the performance models of the overall application and its substructures allowing the application to systematically and efficiently map the composition to the target architectures.

Application oriented schedulers (or mappers) such as AppLeS select the optimal number of processors from a computational resource for a particular problem

size by using static or stochastic computational and networking parameters and standard linear programming techniques [9,16]. Our Application Mapper will extend this work to find an optimal execution plan that considers all the available resources and assumes an application is defined by sequence of inter-dependent tasks, e.g. input/output file staging. Consideration of input and output file staging on a fast heavily used resource means it may be quicker to execute the application on a slower computational resource that has good network connectivity.

The Spawn system has demonstrated how different funding ratios could be used to guide resource allocation and usage [17]. Nimrod/G uses historical execution times and heterogeneous resource costs to implement fixed budget and deadline scheduling of multiple tasks [18]. The resource costs are obtained through standard auctioning techniques (e.g. English, Dutch, Hybrid and Sealed Bid auctions [19]) and incorporated into the linear programming model used by the application mapper when finding an optimal application mapping.

5 Conclusions and Future Work

Computational grids will eventually, like the Internet, change the way we work. However, to effectively exploit the computational potential of the grid we need to articulate the needs of the users, their applications and the resource providers. From this information we can automatically deploy an application to a resource that will satisfy the stated requirements of the user and the resource provider.

Our architecture, through the federation of resources to build computational communities, the use of application mappers to effectively match applications to resources and brokers to make the best economic use of the available resources, address some of the weaknesses in current grid infrastructures. To implement this system we exploit Jini's fault tolerant and decentralised infrastructure and Java's inherent cross-platform portability.

Our prototype implementation is now being re-engineered to fully conform to the model described in this paper and we foresee its deployment over our local test bed during Summer 2001. We also see scope for expanding the software resource model to provide a software service (a software and hardware combination provided by an application service provider) and even deployable single use software libraries. The computational economy could also be extended to include the speculative purchasing of resources (futures) and other market based actions.

References

1. I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann.
2. I. Foster and C. Kesselman. The Globus Project: A Status Report. In *Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop*, pages 4–18, 1998.
3. <http://www.openpbs.org>.

4. <http://www.cs.wisc.edu/condor>.
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. R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, pages 28–31, July 1998.
7. J. Darlington *et al.* Parallel Programming using Skeleton Functions. In *Lecture Notes in Computer Science*, volume 694, pages 146–160.
8. S. Newhouse, A. Mayer, and J. Darlington. A Software Architecture for HPC Grid Applications. In *Euro-Par 2000*, pages 686–689, 2000.
9. H. Dail, G. Obertelli, F. Berman, R. Wolski, and A. Grimshaw. Application-Aware Scheduling of a Magnetohydrodynamics Application in the Legion Metasystem. In *Proceedings of the 9th Heterogeneous Computing Workshop*, May 2000.
10. N. Dragios. Java Metacomputer. Master’s thesis, Imperial College, Department of Computing, 2000.
11. K. Arnold, J. Gosling, and D. Holmes. *The Java Programming Language, (Third Edition)*. Addison-Wesley.
12. <http://java.sun.com/jini/>.
13. M. O. Neary, B. O. Christiansen, P. Cappello, and K. E. Schauer. Javelin: Parallel computing on the internet. In *Future Generation Computer Systems*, volume 15, pages 659–674. Elsevier Science, Amsterdam, Netherlands, October 1999.
14. Z. Juhasz and L. Kesmarki. JINI-Based Prototype Metacomputing Framework. In *Euro-Par 2000*, pages 1171–1174, 2000.
15. J. Darlington, M. Ghanem, Y. Guo, and H. W. To. Guided Resource Organisation in Heterogeneous Parallel Computing. *Journal of High Performance Computing*, 4(10):13–23, 1997.
16. F. Berman and J. M. Schopf. Stochastic scheduling. In *Supercomputing*, 1999.
17. C. Waldsburger *et al.* Spawn: A Distributed Computational Economy. *IEEE Transactions on Software Engineering*, February 1992.
18. R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid. In *The 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000)*. IEEE Computer Society Press, USA, 2000.
19. D. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini. Economic models for allocating resources in computer systems. In Scott Clearwater, editor, *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific, Hong Kong, 1996.