Computational Communities: a Market Place for Federated Resources

Steven Newhouse[*]
Department of Computing
Imperial College of Science, Technology and Medicine
London, UK
s.newhouse@doc.ic.ac.uk

John Darlington[†]
Department of Computing
Imperial College of Science, Technology and Medicine
London, UK
jd@doc.ic.ac.uk

ABSTRACT

In this paper we define a grid middleware, comprising federated computational resources, that facilitates a globally optimal mapping of applications to the available resources and satisfies the goals of both users and resource providers. Resource providers in the community are able to specify access control policies to govern the use of their resources. Applications in the community are annotated with performance and behavioural information to enable the 'best' resources to be found automatically. A computational currency allows both the resource providers and consumers to express their requirements (e.g. completion time and resource utilisation) to support 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 that provide a ubiquitous computing environment [1]. Such infrastructures are gaining acceptance outside the traditional high performance computing community as computational and data intensive applications become commonplace in both science and commerce.

The motivations for combining otherwise disparate computing resources into an integrated environment are primarily:

- To share resources more effectively, providing both wider access to and better utilisation of resources, i.e. to increase utilisation and throughput.
- To connect geographically dispersed instruments or computational resources to provide a unified resource of greater power than would otherwise be available, i.e. meta-computing
- To provide user communities with high-level and easy-to-use access to shared computational resources and to support collaborative working practices and management of scientific intellectual property, i.e. e-science

[*]Research Director, Imperial College Parallel Computing Centre.
[†]Director,Imperial College Parallel Computing Centre.

Early experiments in Grid construction have generally involved the explicit connection of supercomputers or scientific instruments and require a high degree of expertise and involvement from both the users and resource providers [2].

Ideally, computing power should be available 'on tap' from the computational grid in the same manner as electrical power is in the power grid: a ubiquitous and transparent resource. This can only be achieved if the application and *grid middleware* (representing the underlying resources) are tightly integrated.

To support the federation of heterogeneous resources under the administrative control of different organisations, the middleware must mask any heterogeneity and provide:

- **Information.** Effective resource selection (scheduling) requires information on the available hardware, software, storage and networking resources.
- **Security and Control.** Resource providers will only be willing to contribute their resources to a wider computational community if they retain ultimate control over who can access their resources and are able to ensure the needs of their local users.
- **Effective Resource Exploitation.** The *best* resources for an application will depend both on the user's requirements (e.g. turnaround time, availability of the application software) and the resource provider's goals (e.g. revenue or utilisation maximisation).

To provide a grid computing environment that can transparently migrate applications to *better* resources as they become

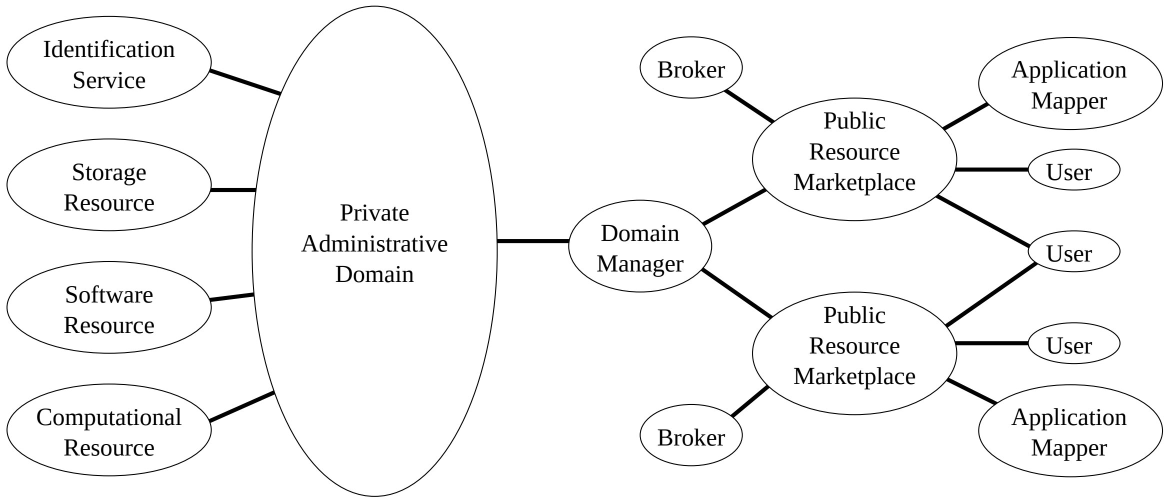

Figure 1: Building a resource marketplace through federated resources

available, and survive the failures that are commonplace in a large distributed computing systems, it is necessary to automate the resource selection process using information from the application and middleware, but to do so within the constraints and goals specified by both the user and resource provider.

With the information relating to the user's and resource provider's goals we are able to balance the resources used to execute a single application against the set of applications currently running over all the available resources in the marketplace. We use a computational currency to enumerate our goals allowing us to find the 'best' resources for our needs. 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.

In the rest of this paper we will demonstrate how this can be achieved through a software architecture that allows the effective exploitation of federated resources through a computational economy. In section 2 we describe a federated resource marketplace that allows organisations to contribute their resources to a wider computational community and how performance models and resource pricing can be used to find a globally optimal job mix across all the community's resources. Section 3 describes a prototype implementation of this architecture that uses Java and JINI while section 4 compares our approach to existing work in this field.

2. A COMPUTATIONAL ECONOMY

2.1 Overview

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

- **Organisations** that contribute resources (software, hardware, storage etc.) under a locally defined access control policy to a public resource marketplace.
- **Application Mappers** that generate a set of execution plans matching the application's requirements to the resources that are currently available in the marketplace. These plans represent feasible execution strategies for the application that are optimal with regard to performance but do not necessarily make the *best* economic use of the resources.
- **Brokers** that handle the negotiations between organisations and users in the resource marketplace to find the *best* economic execution plan from those generated by an application mapper.

An organisation's local storage, software and computational resources are managed through the Administrative Domain by a local administrator. These are made publicly available through the Domain Manager which places this information into one or more public resource marketplaces and enforces the local access control policy.

This model realises the goals of computational grids through bottom up co-operation as opposed to top down proclamation. It allows computational communities (and their resources) to federate when they see mutual benefit in doing so. Federation is achieved by advertising the 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 computing resources will only be federated into a larger computational community if the usage conditions governing remote access are explicit and strictly enforced. These conditions will include authentication and authorisation, but may also relate to current machine load and the identity of the remote user.

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 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.

2.2.1 Resources

Our computational community, like all grid environments, is built from diverse computational, storage and software resources that 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. 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 access their storage space from any resource. This is essential to allow input and output files to be automatically staged across the network to the execution location from different filespaces. Read and write access policies are defined by the user to allow authenticated individuals, groups or organisations to use their file space.
- **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.

2.2.2 Identification Service

Authentication is a primary concern in any distributed computing environment that allows users access to resources managed by other organisations. We use a public key infrastructure to recognise three distinct entities for authentication (and authorisation) purposes: an individual, a group and an organisation. Each federated organisation may act as a Certification Authority to its local user community.

All requests to use a resource have to pass through the local Domain Manager. The Domain Manager delegates authentication of the request to a trusted Identification Service. The Identification Service has a list of trusted organisations (i.e. their X509 certificates) that are generated through off-line resource sharing agreements or are a recognised Certification Authority. Having authenticated the organisation through its certificate the groups and users derived from that organisation's Certification Authority can also be authenticated.

2.2.3 Domain Manager

The Domain Manager is the only route between the private Administrative Domain containing the resources and the public computational communities. As such it has several roles in enforcing the access control polices defined by the local administrator:

- **Authentication.** Authentication of the user, their associated groups and their originating organisation is delegated to the Identification Service.
- **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. The Domain Manager is able to restrict the published information in order to hide specific resources and the details of the local access policy from particular communities. All resource requests are verified by the Domain Manager before being passed to the resource.
- **Execution.** The Domain Manager accepts validated resource requests (as part of a user's execution plan) and passes them onto the individual resources. All monitoring of the job is passed through the Domain Manager to the individual resource.

The Domain Manager allows administrators 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.

2.3 Resource Discovery and Selection

The computational community described in the previous section provides a mechanism for federating the resources of different organisations into a unified set of computational resources. Finding the *best* resources for a complex application from those that are currently available is a decision that is still generally left to the user. However, as the diversity of the heterogeneous resources increases the need for automatic or semi-automatic resource selection will also increase. Effective automatic resource selection requires information relating to the performance of the application on different resources and the requirements of the user to be made available within the computational community.

2.3.1 Execution Requirements

A source code or binary application can be easily annotated with attributes relating to its requirements (e.g. operating system, library versions etc.) enabling deployment to the target architecture. This form of simple resource selection has been demonstrated in the Condor 'matchmaking' system[5]. However, to make effective scheduling decisions (and especially for parallel applications) it is essential to have knowledge as to how the application performs on different architectures and configurations. This knowledge is

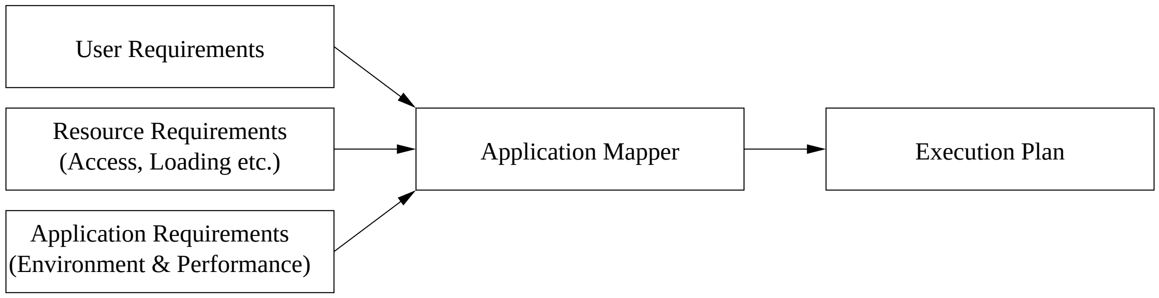
User Requirements
Resource Requirements (Access, Loading etc.)
Application Requirements (Environment & Performance)
Application Mapper
Execution Plan

Figure 2: Application Mapper

used to ensure that the allocated resources guarantee execution by a specific deadline, or in the case of a parallel execution, that optimal speedup is achieved for a particular data size and that the resources are effectively used.

Extracting this performance information automatically from the application is a non-trivial task. However, we believe this information can be expressed (either automatically or manually) through performance models and used to annotate the resource requirements and automate resource selection.

2.3.2 *Application Mapper*

The Application Mapper automates the user's selection of the *best* resources from those that are currently available within the computational community. It uses the requirements specified by the user (relating to job completion time), the application (run-time environment and execution time on different platforms) and the resource provider (governing resource access, capability and loading) to generate a set of viable execution plans for final selection by the user (see figure 2). Multiple Application Mappers can operate in the same computational community to provide the user with a variety of feasible execution plans generated using different approaches.

2.3.3 *Maintaining Optimal Execution Plans*

The selected execution plan remains optimal provided the state of the computational community remains unchanged and jobs complete as predicted. Any change in the system (either of an application or a resource) will trigger a re-evaluation of the current application mapping and a possible rescheduling of future or current operations. Applications that are capable of being remapped during execution to other resources have to provide a performance model describing the cost of the remapping operation and an interface to support it.

2.4 Computational Supply and Demand

The previous section presented a mechanism for matching federated resources to the execution requirements of an application. Although the Application Mapper allows an individual user to find the most effective resources from within the computational community for their application, as yet we have introduced no mechanisms for the resource provider to maximise their resource utilisation.

For example, suppose 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).

2.4.1 *Computational Currency*

To support our marketplace we define a computational currency, backed by a trusted organisation, as an exchange medium in our computational economy. As different marketplaces may use different currencies we foresee the need for resource providers to recognise and convert between several currencies.

2.4.2 *Resource Broker*

The Resource Broker negotiates a cost for the execution plans with the relevant resource provider (assuming the user is allowed to use the resource and it is available for use) 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. The user may have a limited budget or immovable job completion time but may be willing to reduce the job turnaround time by using a more expensive resource. 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 on each individual request.

2.5 Tools

Interaction with the architecture takes place through the following tools:

- The **Resource Manager** allows the system administrator to alter the resource's configuration (adding and removing attributes) and to define the access policy for each resource marketplace.
- The **Resource Browser** allows the user to examine the usage policies and the attributes (i.e. operating system revision and current load) of the resources in the marketplace. It can also be used for manual resource selection or automatic optimal application partitioning through the use of the application mapper.
- The **Grid Client** allows the user to define their application's requirements, publish these requirements in the resource marketplace and examine the execution plans and associated costs generated by the application mapper and the resource broker.

3. IMPLEMENTATION

3.1 Technology

An initial proof of concept prototype of this architecture has been completed at Imperial College using a Java and JINI environment [6]. The architecture described in this paper represents an extension of this work. We exploit Java's cross-platform portability (essential in any heterogeneous environment) and its rich API's to simplify many of the development tasks [7]. We have also adopted JINI as the primary service infrastructure for our architecture [8]. JINI has many desirable characteristics for a wide area grid environment. It supports dynamic registration, look-up and connection between the Java objects that represent our grid services and resources. As all grid resources are effectively transient this ability to connect and reconnect over time is a highly desirable feature. As a consequence the JINI leasing mechanism also allows unexpected failures to be gracefully handled. We use a JINI look-up server to implement the public resource marketplaces and private administrative domains defined in our architecture.

3.2 Testing

The Imperial College Parallel Computing Centre provides support for multi-disciplinary applications of high performance computing across all the constituent departments and centres of Imperial College. As such its hardware represents a realistic development test bed for our resource marketplace comprising dedicated high performance machines to commodity PC clusters. From our diverse existing user community we have identified two application areas that will immediately benefit from a grid infrastructure:

- **High Throughput Computing** applications in Particle Physics, Bioinformatics and Medical Image processing.

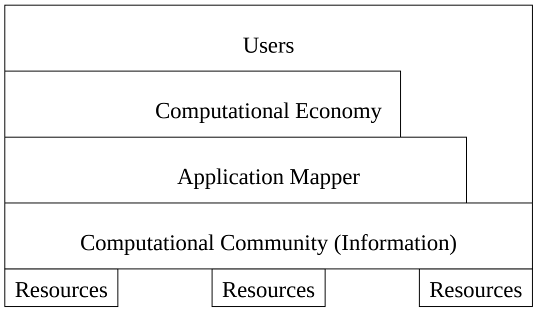

Figure 3: JINI Grid Middleware

- **Distributed High Performance Computing** applications involving the simulation of the solar coronal mass ejections and coupled fluid-structure acoustics.

4. RELATED WORK

4.1 Middleware

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 [9].

We also use an object model to provide generic functionality that can be customerised by the user while retaining the flexibility of Globus's toolkit approach. Our middleware is capable of being developed, deployed and used in increasingly sophisticated forms (see figure 3). For example, the resource browser can be just used to extract information from the computational community to display the current and projected machine load as a virtual machine room [10]. The application mapper can be used to suggest feasible mappings to the currently available resources. By defining a computational currency the user is encouraged to make effective use of the community's computational resources. This architecture is also open and extensible, allowing its components to be interchanged with the results of other grid research.

Our initial implementation indicates that the Java/JINI combination is capable of providing an extensible fault tolerant distributed infrastructure for grid computing. Work elsewhere has also demonstrated the effectiveness of JINI in providing a grid middleware [11] and the use of Java to provide a homogeneous distributed computing environment across heterogeneous resources (e.g. Javelin [12] and other projects). 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.

4.2 Application Construction

The effectiveness of our middleware relies on the ability to describe the performance and behaviour of an application. 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) [13, 14]. 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.

The parallel behaviour of sequential threads that are themselves written in conventional imperative languages can be defined through parallel structural forms [15]. This approach has shown that complex parallel algorithms specified in this manner are as efficient and scalable as the best hand written code [16, 17]. However, adoption was hindered by the use of an unorthodox language framework. Our experiences with structured coordination languages is now being used in the context of conventional software components (e.g. Java Beans). By annotating software components with XML encoded meta-data relating to how they can be used, deployed and perform we can compose an application and systematically and efficiently map the composition to the target architectures [18]. This approach yields the performance models of the overall application and its substructures, and the execution and data flow graphs needed for mapping an application to the heterogeneous resources that comprise the computational grids.

4.3 Application Scheduling

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 [19, 20, 21]. Our Application Mapper will extend this work to find an optimal execution plan (or application partitioning) that considers all the resources in the computational community and assumes an application is defined by sequence of interdependent tasks. Many jobs are defined as a network of operations (e.g. transfer input files, execute the application, transfer the output files) with associated execution and data flow graphs. If each operation provides a performance model the Application Mapper has sufficient information to find the resources that minimises the overall execution and completion time.

Consideration of input and output file staging and current computational load may mean that the fastest computational resource will not always complete the job in the shortest possible time. If the resource is heavily used it may take several hours for the job to start, while other lightly loaded, but slower resources, would complete the work in less time. As applications become increasingly dependent on larger data sets it may be quicker to execute the application on a slow computational resource that has good network connectivity rather than transferring the data to a faster computational resource.

The need for superschedulers that consider resources across administrative domains has been recognised by Scheduling Working Group of the Grid Forum [22].

4.4 Computational Economics

The Spawn system has demonstrated how different funding ratios could be used to guide resource allocation and usage [23]. Nimrod/G uses historical execution times and heterogeneous resource costs to implement fixed budget and deadline scheduling of multiple tasks [24]. The resource costs are obtained through standard auctioning techniques (e.g. English, Dutch, Hybrid and Sealed Bid auctions [25]) and incorporated into the linear programming model used by the application mapper when finding an optimal application mapping.

5. FUTURE WORK

Our experiences in using Java and JINI to produce grid middleware have to date been experimental but encouraging. The rich API's within Java and JINI's fault tolerance features eliminate many implementation issues. A JINI working group was formed at a recent Global Grid Forum meeting to co-ordinate research in this area [26]. The prototype implementation is now being re-engineered to fully conform to the model described in the 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 (licences to a particular user for a single execution). Instead of buying a fixed software licence users will pay for each use even if the application is executed on their own or another organisation's resources.

The computational economy could be extended to include the speculative purchasing of resources (futures) and other market based actions. Such an open market in resource provision would enable resource owners, be they HPC Centres, single machine owners or software component providers, to participate in dynamically evolving Computational Communities that users could access to satisfy their computational requirements.

6. CONCLUSION

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.

7. 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] 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.

[6] N. Dragios. Java Metacomputer. Master's thesis, Imperial College, Department of Computing, 2000.

[7] K. Arnold, J. Gosling, and D. Holmes. *The Java Programming Language, (Third Edition).* Addison-Wesley.

[8] http://java.sun.com/jini/.

[9] 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.

[10] http://www.ncsa.uiuc.edu/SCD/Alliance/VMR/.

[11] Z. Juhasz and L. Kesmarki. JINI-Based Prototype Metacomputing Framework. In *Euro–Par 2000*, pages 1171–1174, 2000.

[12] 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, Amsterdam, Netherlands, October 1999.

[13] J. Darlington *et al.* Parallel Programming using Skeleton Functions. In *Lecture Notes in Computer Science*, volume 694, pages 146–160.

[14] 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.

[15] http://hpc.doc.ic.ac.uk/environments/coordination/.

[16] J. Yang. *Co-ordination Based Structured Parallel Programming.* PhD thesis, Department of Computing, Imperial College, 1999.

[17] P. Au, J. Darlington, M. M. Ghanem, and Y. Guo. Co-ordinating Heterogeneous Parallel Computation. In L. Boug, P. Fraigniaud, A. Mignotte, and Y. Robert, editors, *Euro-Par '96*, pages 601–614, August 1996.

[18] S. Newhouse, A. Mayer, and J. Darlington. A Software Architecture for HPC Grid Applications. In *Euro–Par 2000*, pages 686–689, 2000.

[19] F. Berman and B. Wolski. The AppLeS project: A status report. In *Proc. NEC Symp. On Metacomputing*, 1997.

[20] F. Berman and J. M. Schopf. Stochastic scheduling. In *Supercomputing*, 1999.

[21] 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.

[22] http://www.cs.nwu.edu/~jms/sched-wg/.

[23] C. Waldsburger *et al.* Spawn: A Distributed Computational Economy. *IEEE Transactions on Software Engineering*, February 1992.

[24] 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.

[25] 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.

[26] http://www.gridforum.org/.