SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2002; (in press) (DOI: 10.1002/spe.488)

SP&E

Grids and Grid technologies for wide-area distributed computing

Mark Baker[1], Rajkumar Buyya[2,*,†] and Domenico Laforenza[3]

[1] *School of Computer Science, University of Portsmouth, Mercantile House, Portsmouth, U.K.*
[2] *Grid Computing and Distributed Systems Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Melbourne, Australia*
[3] *Centro Nazionale Universitario di Calcolo Elettronico (CNUCE), Instituto del Consiglio Nazionale delle Ricerche (CNR), Area della Ricerca CNR, Pisa, Italy*

SUMMARY

The last decade has seen a substantial increase in commodity computer and network performance, mainly as a result of faster hardware and more sophisticated software. Nevertheless, there are still problems, in the fields of science, engineering, and business, which cannot be effectively dealt with using the current generation of supercomputers. In fact, due to their size and complexity, these problems are often very numerically and/or data intensive and consequently require a variety of heterogeneous resources that are not available on a single machine. A number of teams have conducted experimental studies on the cooperative use of geographically distributed resources unified to act as a single powerful computer. This new approach is known by several names, such as metacomputing, scalable computing, global computing, Internet computing, and more recently peer-to-peer or Grid computing. The early efforts in Grid computing started as a project to link supercomputing sites, but have now grown far beyond their original intent. In fact, many applications can benefit from the Grid infrastructure, including collaborative engineering, data exploration, high-throughput computing, and of course distributed supercomputing. Moreover, due to the rapid growth of the Internet and Web, there has been a rising interest in Web-based distributed computing, and many projects have been started and aim to exploit the Web as an infrastructure for running coarse-grained distributed and parallel applications. In this context, the Web has the capability to be a platform for parallel and collaborative work as well as a key technology to create a pervasive and ubiquitous Grid-based infrastructure. This paper aims to present the state-of-the-art of Grid computing and attempts to survey the major international efforts in developing this emerging technology. Copyright © 2002 John Wiley & Sons, Ltd.

KEY WORDS: grid computing; middleware; resource management; scheduling; distributed applications

*Correspondence to: Rajkumar Buyya, Grid Computing and Distributed Systems Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, SEECS Building, 221 Boulevard St., Carlton, Melbourne, Australia.
†E-mail: raj@cs.mu.oz.au

Received 4 October 2001
Revised 1 August 2002
Accepted 1 August 2002

Figure 1. Towards Grid computing: a conceptual view.

1. INTRODUCTION

The popularity of the Internet as well as the availability of powerful computers and high-speed network technologies as low-cost commodity components is changing the way we use computers today. These technology opportunities have led to the possibility of using distributed computers as a single, unified computing resource, leading to what is popularly known as Grid computing [1]. The term Grid is chosen as an analogy to a power Grid that provides consistent, pervasive, dependable, transparent access to electricity irrespective of its source. A detailed analysis of this analogy can be found in [2]. This new approach to network computing is known by several names, such as metacomputing, scalable computing, global computing, Internet computing, and more recently peer-to-peer (P2P) computing [3].

Grids enable the sharing, selection, and aggregation of a wide variety of resources including supercomputers, storage systems, data sources, and specialized devices (see Figure 1) that are geographically distributed and owned by different organizations for solving large-scale computational and data intensive problems in science, engineering, and commerce. Thus creating virtual organizations [4] and enterprises [5] as envisioned in [6]—as a temporary alliance of enterprises or organizations that come together to share resources and skills, core competencies, or resources in order to better respond to business opportunities or large-scale application processing requirements, and whose cooperation is supported by computer networks.

The concept of Grid computing started as a project to link geographically dispersed supercomputers, but now it has grown far beyond its original intent. The Grid infrastructure can benefit many applications, including collaborative engineering, data exploration, high-throughput computing, and distributed supercomputing.

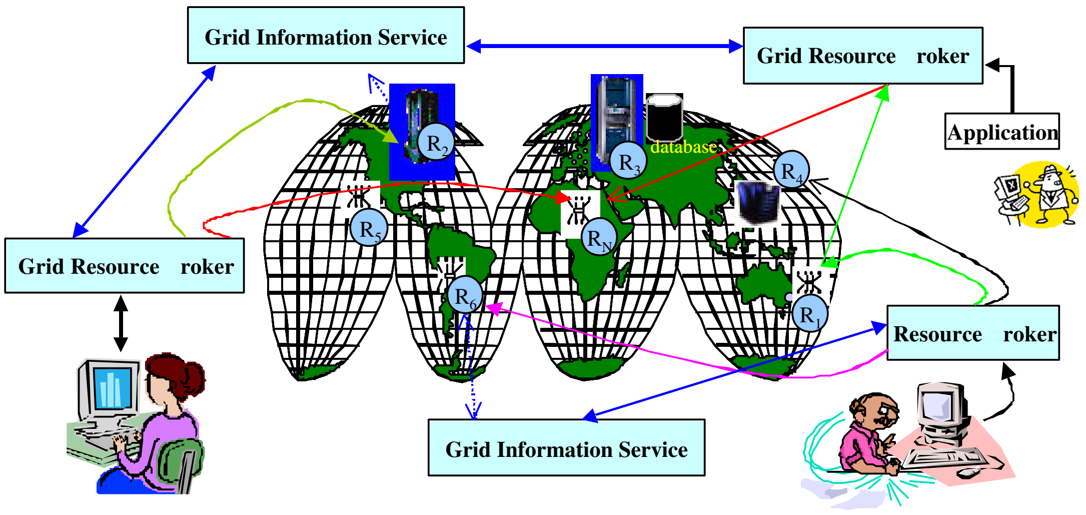

Figure 2. A high-level view of the Grid and interaction between its entities.

A Grid can be viewed as a seamless, integrated computational and collaborative environment (see Figure 1) and a high-level view of activities within the Grid is shown in Figure 2. The users interact with the Grid resource broker to solve problems, which in turn performs resource discovery, scheduling, and the processing of application jobs on the distributed Grid resources. From the end-user point of view, Grids can be used to provide the following types of services.

- *Computational services*. These are concerned with providing secure services for executing application jobs on distributed computational resources individually or collectively. Resources brokers provide the services for collective use of distributed resources. A Grid providing computational services is often called a computational Grid. Some examples of computational Grids are: NASA IPG [7], the World Wide Grid [8], and the NSF TeraGrid [9].
- *Data services*. These are concerned with proving secure access to distributed datasets and their management. To provide a scalable storage and access to the data sets, they may be replicated, catalogued, and even different datasets stored in different locations to create an illusion of mass storage. The processing of datasets is carried out using computational Grid services and such a combination is commonly called data Grids. Sample applications that need such services for management, sharing, and processing of large datasets are high-energy physics [10] and accessing distributed chemical databases for drug design [11].
- *Application services*. These are concerned with application management and providing access to remote software and libraries transparently. The emerging technologies such as Web services [12] are expected to play a leading role in defining application services. They build on computational and data services provided by the Grid. An example system that can be used to develop such services is NetSolve [13].

- *Information services*. These are concerned with the extraction and presentation of data with meaning by using the services of computational, data, and/or application services. The low-level details handled by this are the way that information is represented, stored, accessed, shared, and maintained. Given its key role in many scientific endeavors, the Web is the obvious point of departure for this level.
- *Knowledge services.* These are concerned with the way that knowledge is acquired, used, retrieved, published, and maintained to assist users in achieving their particular goals and objectives. Knowledge is understood as information applied to achieve a goal, solve a problem, or execute a decision. An example of this is data mining for automatically building a new knowledge.

To build a Grid, the development and deployment of a number of services is required. These include security, information, directory, resource allocation, and payment mechanisms in an open environment [1,14,15]; and high-level services for application development, execution management, resource aggregation, and scheduling.

Grid applications (typically multidisciplinary and large-scale processing applications) often couple resources that cannot be replicated at a single site, or which may be globally located for other practical reasons. These are some of the driving forces behind the foundation of global Grids. In this light, the Grid allows users to solve larger or new problems by pooling together resources that could not be easily coupled before. Hence, the Grid is not only a computing infrastructure, for large applications, it is a technology that can bond and unify remote and diverse distributed resources ranging from meteorological sensors to data vaults, and from parallel supercomputers to personal digital organizers. As such, it will provide pervasive services to all users that need them.

This paper aims to present the state-of-the-art of Grid computing and attempts to survey the major international efforts in this area. A set of general principles and design criteria that can be followed in the Grid construction are given in Section 2. Some of the current Grid technologies, selected as representative of those currently available, are presented in Section 3. In Section 4, we note a few scientific applications of Grids. We conclude and then discuss future trends in Section 5.

2. GRID CONSTRUCTION: GENERAL PRINCIPLES

This section briefly highlights some of the general principles that underlie the construction of the Grid. In particular, the idealized design features that are required by a Grid to provide users with a seamless computing environment are discussed. Four main aspects characterize a Grid.

- *Multiple administrative domains and autonomy*. Grid resources are geographically distributed across multiple administrative domains and owned by different organizations. The autonomy of resource owners needs to be honored along with their local resource management and usage policies.
- *Heterogeneity*. A Grid involves a multiplicity of resources that are heterogeneous in nature and will encompass a vast range of technologies.
- *Scalability*. A Grid might grow from a few integrated resources to millions. This raises the problem of potential performance degradation as the size of Grids increases. Consequently,

applications that require a large number of geographically located resources must be designed to be latency and bandwidth tolerant.

- *Dynamicity or adaptability*. In a Grid, resource failure is the rule rather than the exception. In fact, with so many resources in a Grid, the probability of some resource failing is high. Resource managers or applications must tailor their behavior dynamically and use the available resources and services efficiently and effectively.

The steps necessary to realize a Grid include:

- the integration of individual software and hardware components into a combined networked resource (e.g. a single system image cluster);
- the deployment of:
  - low-level middleware to provide a secure and transparent access to resources;
  - user-level middleware and tools for application development and the aggregation of distributed resources;
- the development and optimization of distributed applications to take advantage of the available resources and infrastructure.

The components that are necessary to form a Grid (shown in Figure 3) are as follows.

- *Grid fabric*. This consists of all the globally distributed resources that are accessible from anywhere on the Internet. These resources could be computers (such as PCs or Symmetric Multi-Processors) running a variety of operating systems (such as UNIX or Windows), storage devices, databases, and special scientific instruments such as a radio telescope or particular heat sensor.
- *Core Grid middleware*. This offers core services such as remote process management, co-allocation of resources, storage access, information registration and discovery, security, and aspects of Quality of Service (QoS) such as resource reservation and trading.
- *User-level Grid middleware*. This includes application development environments, programming tools, and resource brokers for managing resources and scheduling application tasks for execution on global resources.
- *Grid applications and portals*. Grid applications are typically developed using Grid-enabled languages and utilities such as HPC++ or MPI. An example application, such as parameter simulation or a grand-challenge problem, would require computational power, access to remote data sets, and may need to interact with scientific instruments. Grid portals offer Web-enabled application services, where users can submit and collect results for their jobs on remote resources through the Web.

In attempting to facilitate the collaboration of multiple organizations running diverse autonomous heterogeneous resources, a number of basic principles should be followed so that the Grid environment:

- does not interfere with the existing site administration or autonomy;
- does not compromise existing security of users or remote sites;
- does not need to replace existing operating systems, network protocols, or services;
- allows remote sites to join or leave the environment whenever they choose;
- does not mandate the programming paradigms, languages, tools, or libraries that a user wants;

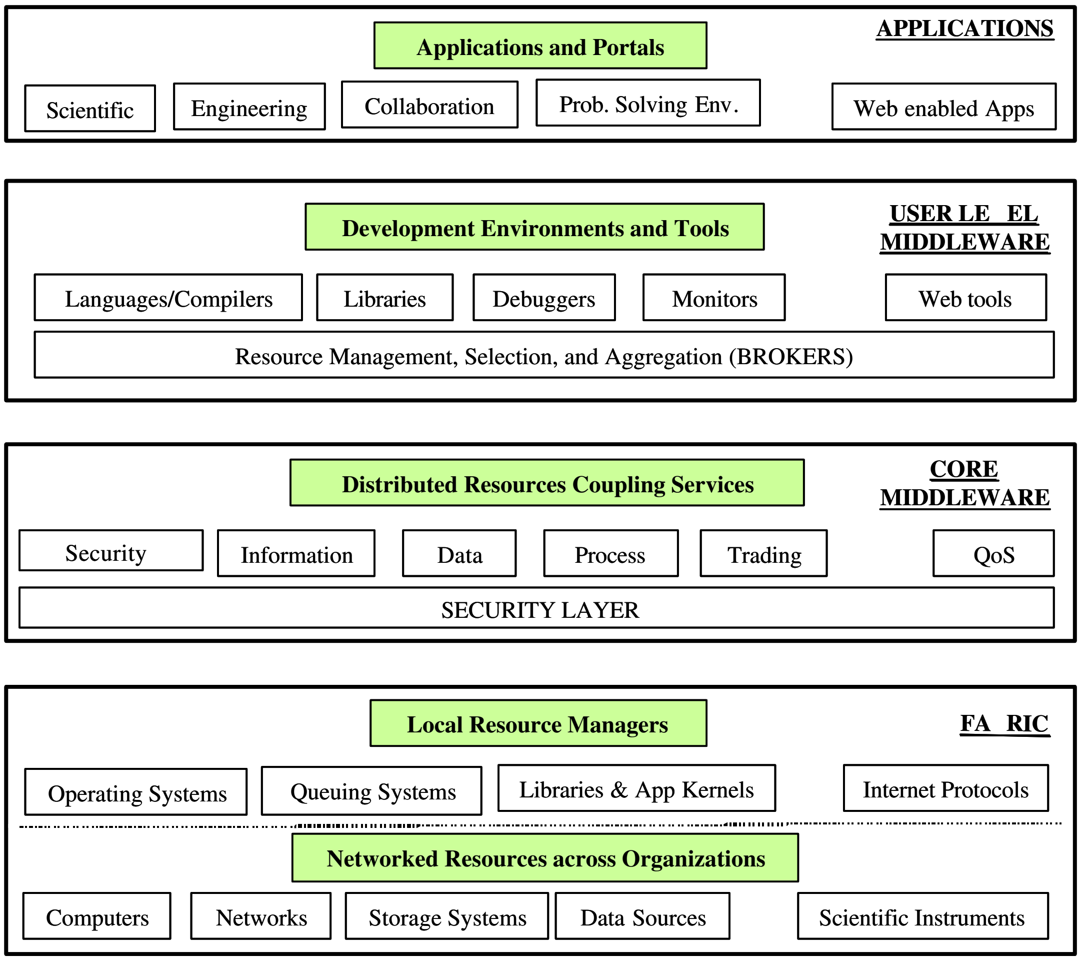

Figure 3. A layered Grid architecture and components.

- provides a reliable and fault tolerant infrastructure with no single point of failure;
- provides support for heterogeneous components;
- uses standards, and existing technologies, and is able to interact with legacy applications;
- provides appropriate synchronization and component program linkage.

As one would expect, a Grid environment must be able to interoperate with a whole spectrum of current and emerging hardware and software technologies. An obvious analogy is the Web. Users of the Web do not care if the server they are accessing is on a UNIX or Windows platform. From the client browser's point of view, they 'just' want their requests to Web services handled quickly and efficiently. In the same way, a user of a Grid does not want to be bothered with details of its underlying hardware and software infrastructure. A user is really only interested in submitting their application to the appropriate resources and getting correct results back in a timely fashion.

An ideal Grid environment will therefore provide access to the available resources in a seamless manner such that physical discontinuities, such as the differences between platforms, network protocols, and administrative boundaries become completely transparent. In essence, the Grid middleware turns a radically heterogeneous environment into a virtual homogeneous one.

The following are the main design features required by a Grid environment.

- *Administrative hierarchy*. An administrative hierarchy is the way that each Grid environment divides itself up to cope with a potentially global extent. The administrative hierarchy determines how administrative information flows through the Grid.
- *Communication services*. The communication needs of applications using a Grid environment are diverse, ranging from reliable point-to-point to unreliable multicast communications. The communications infrastructure needs to support protocols that are used for bulk-data transport, streaming data, group communications, and those used by distributed objects. The network services used also provide the Grid with important QoS parameters such as latency, bandwidth, reliability, fault-tolerance, and jitter control.
- *Information services*. A Grid is a dynamic environment where the location and types of services available are constantly changing. A major goal is to make all resources accessible to any process in the system, without regard to the relative location of the resource user. It is necessary to provide mechanisms to enable a rich environment in which information is readily obtained by requesting services. The Grid information (registration and directory) services components provide the mechanisms for registering and obtaining information about the Grid structure, resources, services, and status.
- *Naming services*. In a Grid, like in any distributed system, names are used to refer to a wide variety of objects such as computers, services, or data objects. The naming service provides a uniform name space across the complete Grid environment. Typical naming services are provided by the international X.500 naming scheme or DNS, the Internet's scheme.
- *Distributed file systems and caching*. Distributed applications, more often than not, require access to files distributed among many servers. A distributed file system is therefore a key component in a distributed system. From an applications point of view it is important that a distributed file system can provide a uniform global namespace, support a range of file I/O protocols, require little or no program modification, and provide means that enable performance optimizations to be implemented, such as the usage of caches.
- *Security and authorization*. Any distributed system involves all four aspects of security: confidentiality, integrity, authentication, and accountability. Security within a Grid environment is a complex issue requiring diverse resources autonomously administered to interact in a manner that does not impact the usability of the resources or introduces security holes/lapses in individual systems or the environments as a whole. A security infrastructure is the key to the success or failure of a Grid environment.
- *System status and fault tolerance*. To provide a reliable and robust environment it is important that a means of monitoring resources and applications is provided. To accomplish this task, tools that monitor resources and application need to be deployed.

- *Resource management and scheduling*. The management of processor time, memory, network, storage, and other components in a Grid is clearly very important. The overall aim is to efficiently and effectively schedule the applications that need to utilize the available resources in the Grid computing environment. From a user's point of view, resource management and scheduling should be transparent; their interaction with it being confined to a manipulating mechanism for submitting their application. It is important in a Grid that a resource management and scheduling service can interact with those that may be installed locally.
- *Computational economy and resource trading*. As a Grid is constructed by coupling resources distributed across various organizations and administrative domains that may be owned by different organizations, it is essential to support mechanisms and policies that help in regulate resource supply and demand [16,17]. An economic approach is one means of managing resources in a complex and decentralized manner. This approach provides incentives for resource owners, and users to be part of the Grid and develop and using strategies that help maximize their objectives.
- *Programming tools and paradigms*. Grid applications (multi-disciplinary applications) couple resources that cannot be replicated at a single site even or may be globally located for other practical reasons. A Grid should include interfaces, APIs, utilities, and tools to provide a rich development environment. Common scientific languages such as C, C++, and Fortran should be available, as should application-level interfaces such as MPI and PVM. A variety of programming paradigms should be supported, such as message passing or distributed shared memory. In addition, a suite of numerical and other commonly used libraries should be available.
- *User and administrative GUI*. The interfaces to the services and resources available should be intuitive and easy to use. In addition, they should work on a range of different platforms and operating systems. They also need to take advantage of Web technologies to offer a view of portal supercomputing. The Web-centric approach to access supercomputing resources should enable users to access any resource from anywhere over any platform at any time. That means, the users should be allowed to submit their jobs to computational resources through a Web interface from any of the accessible platforms such as PCs, laptops, or Personal Digital Assistant, thus supporting the ubiquitous access to the Grid. The provision of access to scientific applications through the Web (e.g. RWCPs parallel protein information analysis system [18]) leads to the creation of science portals.

3. GRID COMPUTING PROJECTS

There are many international Grid projects worldwide, which are hierarchically categorized as integrated Grid systems, core middleware, user-level middleware, and applications/application driven efforts (see Table I). Selected ones are further grouped into country/continents wise as listed in Tables II–V. A listing of majority of projects in Grid computing worldwide along with pointers to their Web sites can be found in [19–21]. A description of two community driven forums, Global Grid Forum (GGF) and Peer-to-Peer (P2P) Working Group promoting wide-area distributed computing technologies, applications, and standards is given in Table VI. This section discusses some of the current Grid projects representative of the Grid technology approaches.

Table I. Hierarchical organization of major Grid efforts.

Category	Project	Organization	Remarks
Integrated Grid systems	NetSolve	University of Tennessee	A programming and runtime system for accessing high-performance libraries and resources transparently
	Ninf	Tokyo Institute of Technology	Functionality is similar to NetSolve
	ST-ORM	UPC, Barcelona	A scheduler for distributed batch systems
	SILVER	PPNL and University of Utah	A scheduler for distributed batch systems
	Albatross	Vrije University	An object-oriented programming system
	PUNCH	Purdue University	A portal computing environment and service for applications
	Javelin	UCSB	A Java-based programming and runtime system
	XtremWeb	Paris-Sud University	A global computing environment
	MILAN	Arizona and NY	Aims to provide end-to-end services for transparent utilization and management of networked resources
	DISCWorld	University of Adelaide	A distributed information-processing environment
	Unicore	Germany	A Java-based environment for accessing remote supercomputers
Core middleware	Cosm	Mithral	A toolkit building P2P applications
	Globus	ANL and ISI	Provides uniform and secure environment for accessing remote computational and storage resources
	Gridbus	University of Melbourne	Provides technologies that support end-to-end computational economy-driven resource sharing, management, and scheduling
	GridSim	Monash University	A toolkit for Grid simulation
	JXTA	Sun Microsystems	A Java-based framework and infrastructure for P2P computing
	Legion	University of Virginia	A Grid operating system providing transparent access to distributed resources
	P2P Accelerator	Intel	A basic infrastructure for creating P2P applications for the .NET platform
User-level middleware: *Schedulers*	AppLeS	UCSD	An application specific scheduler
	Condor-G	University of Wisconsin	A wide-area job processing system
	Nimrod-G	Monash University	An economic-based Grid resource broker for parameter sweep/task farming applications

Table I. *Continued.*

Category	Project	Organization	Remarks
User-level middleware: *Programming environments*	MPICH-G	Northern Illinois University	MPI implementation on Globus
	Nimrod parameter programming tools	Monash University	A declarative language parametric programming
	MetaMPI	HLRS, Stuttgart	An MPI programming and runtime environment
	Cactus	Max Planck Institute for Gravitational Physics	A framework for writing parallel applications. It was developed using MPICH-G and Globus
	GrADS	Rice University	Grid application development tools
	GridPort	SDSC	Tools for creating computing portals
Applications and application-driven Grid efforts	European DataGrid	CERN	High-energy physics, earth observation, biology
	GriPhyN	UCF and ANL	High-energy physics
	PPDG	Caltech and ANL	High-energy physics
	Virtual Laboratory	Monash University and WEHI	Molecular modeling for drug design
	HEPGrid	Melbourne University	High-energy physics
	NEESGrid	NCSA	Earthquake engineering
	Geodise	Southampton University	Aerospace design optimization
	Fusion Grid	Princeton/ANL	Magnetic fusion
	IPG	NASA	Aerospace
	ActiveSheets	Monash, QUT, and DSTC	Spread sheet processing
	Earth System Grid	LLNL, ANL, and NCAR	Climate modeling
	Virtual Instruments	UCSD	Neuroscience
	National Virtual Observatory	Johns Hopkins University and Caltech	Access to distributed astronomical databases and processing
	Brain Activity analysis	Osaka University and the University of Melbourne	Analysis of human brain's activity data gathered through magneto-encephalography (MEG) sensors to identify symptoms of diseases

Table II. Some Australian Grid efforts.

Project	Focus and technologies developed	Category
ActiveSheets	ActiveSheets enables the transparent processing of spreadsheet applications modeled in the Microsoft Excel package on distributed computers using Nimrod-G task processing and scheduling services—www.dstc.edu.au/Research/activesheets-ov.html	Application portal
Compute Power Market	CPM aims to develop market-based resource management and scheduling tools and technologies for P2P style computing—www.computepower.com	Middleware
DISCWorld	DISCWorld is an infrastructure for service-based metacomputing across LANs and WANs. DISCWorld allows remote users to login over the Web and request access to data, and invoke services or operations on the available data—dhpc.adelaide.edu.au/Projects/DISCWorld/	Integrated application and middleware system
Gridbus	A Grid toolkit for enabling Grid computing and Business for service-oriented computing. The key objective of the Gridbus project is to develop fundamental, next-generation cluster and grid technologies that support true utility-driven service-oriented computing	Middleware
GridSim	GridSim is Java-based toolkit for modelling and simulation of computational resources for design and evaluation of schedulers and scheduling algorithms for network based high-performance cluster and Grid computing—www.buyya.com/gridsim/	Grid simulation toolkit
Nimrod/G and GRACE	Nimrod/G & Grace are brokers for resource management and scheduling of parameter sweep (coarse-grained, data parallel) applications using computational economy and QoS constraints. The brokers dynamically lease Grid resources/services at runtime depending on their capability, cost, and availability to meet user objectives—www.buyya.com/ecogrid	Grid scheduler and resource trader
Virtual Laboratory	VL provides an application development and execution environment for solving large-scale data intensive applications such as molecular modeling for drug design—www.buyya.com/vlab	Application modeling and execution environment
World Wide Grid (WWG)	WWG is a large-scale testbed for Grid computing. It has heterogeneous computational resources distributed across multiple organization in five continents—www.buyya.com/ecogrid/wwg	Grid testbed

3.1. Globus

Globus [23] provides a software infrastructure that enables applications to handle distributed heterogeneous computing resources as a single virtual machine. The Globus project is a U.S. multi-institutional research effort that seeks to enable the construction of computational Grids. A computational Grid, in this context, is a hardware and software infrastructure that provides dependable, consistent, and pervasive access to high-end computational capabilities, despite the geographical distribution of both resources and users. Globus provides basic services and capabilities that are required to construct a computational Grid. The toolkit consists of a set of components

Table III. Some European Grid efforts.

Project	Focus and technologies developed	Category
UNICORE	The UNiform Interface to Computer Resources aims to deliver software that allows users to submit jobs to remote high-performance computing resources—www.unicore.org	A portal and middleware
MOL	Metacomputer OnLine is a toolbox for the coordinated use of WAN/LAN connected systems. MOL aims at utilizing multiple WAN-connected high-performance systems for solving large-scale problems that are intractable on a single supercomputer—www.uni-paderborn.de/pc2/projects/mol	User level middleware
Cactus	The Cactus toolkit is a set of arrangements providing a general computational infrastructure for many different applications. It contains modules (called thorns) that can be plugged into a core code (called the flesh) that contains the APIs and infrastructure to glue the thorns together. Applications need to create thorns for solving problems—www.cactuscode.org	Application development toolkit
Globe	Globe is a research project aiming to study and implement a powerful unifying paradigm for the construction of large-scale wide-area distributed systems: distributed shared objects—www.cs.vu.nl/∼steen/globe	Object-based operating environment/ middleware system
DataGrid	This project aims to develop middleware and tools necessary for the data-intensive applications of high-energy physics—www.eu-dataGrid.org/	DataGrid middleware and applications
MetaMPI	MetaMPI supports the coupling of heterogeneous MPI systems, thus allowing parallel applications developed using MPI to be run on Grids without alteration—http://www.hlrs.de/organization/pds/projects/metodis/	Programming environment
U.K. eScience	The thrust of the U.K. eScience programme is to develop tools and applications that enable scientists and engineers to transparently access remote computational machines and instruments—www.nesc.ac.uk	applications
XtremWeb	XtremWeb is a Java-based toolkit for developing a cycle stealing infrastructure for solving large-scale applications—www.xtremweb.net	Middleware environment

that implement basic services, such as security, resource location, resource management, and communications.

It is necessary for computational Grids to support a wide variety of applications and programming paradigms. Consequently, rather than providing a uniform programming model, such as the object-oriented model, the Globus provides a bag of services which developers of specific tools or applications can use to meet their own particular needs. This methodology is only possible when the services are distinct and have well-defined interfaces (APIs) that can be incorporated into applications or tools in an incremental fashion.

Globus is constructed as a layered architecture in which high-level global services are built upon essential low-level core local services. The Globus toolkit is modular, and an application can exploit

Table IV. Some Japanese Grid efforts.

Project	Focus and technologies developed	Category
Ninf	Ninf allows users to access computational resources including hardware, software and scientific data distributed across a wide-area network with an easy-to-use interface—http://ninf.apgrid.org/	Development and execution environment
Bricks	Bricks is a performance evaluation system that allows analysis and comparison of various scheduling schemes in a typical high-performance global computing setting—ninf.is.titech.ac.jp/bricks/	Simulation/ performance evaluation system
Grid Datafarm	Grid Datafarm focuses on developing large distributed data storage management and processing technologies for peta-scale data intensive computing [22]—datafarm.apGrid.org/	Middleware

Globus features, such as resource management or information infrastructure, without using the Globus communication libraries. The Globus toolkit supports the following:

- Grid Security Infrastructure (GSI);
- GridFTP;
- Globus Resource Allocation Manager (GRAM);
- Metacomputing Directory Service (MDS-2);
- Global Access to Secondary Storage (GASS);
- data catalogue and replica management;
- Advanced Resource Reservation and Allocation (GARA);

Globus can be viewed as a Grid computing framework based on a set of APIs to the underlying services. Globus provides application developers with a pragmatic means of implementing a range of services to provide a wide-area application execution environment.

3.2. Legion

Legion [24] is an object-based metasystem developed at the University of Virginia. Legion provides the software infrastructure so that a system of heterogeneous, geographically distributed, high-performance machines can interact seamlessly. Legion attempts to provide users, at their workstations, with a single, coherent, virtual machine. In the Legion system the following apply.

- *Everything is an object*. Objects represent all hardware and software components. Each object is an active process that responds to method invocations from other objects within the system. Legion defines an API for object interaction, but not the programming language or communication protocol.
- *Classes manage their instances*. Every Legion object is defined and managed by its own active class object. Class objects are given system-level capabilities; they can create new instances, schedule them for execution, activate or deactivate an object, as well as provide state information to client objects.

Table V. Some U.S. Grid efforts.

Initiative	Focus and technologies developed	Category
Globus	The Globus project is developing a basic software infrastructure for computations that integrate geographically distributed computational and information resources—www.globus.org	Core middleware and toolkit
Legion	Legion is an object-based metasystem. Legion supports transparent scheduling, data management, fault tolerance, site autonomy, and a wide range of security options—legion.virginia.edu	Core middleware and toolkit
Javelin	Javelin provides Internet-based parallel computing using Java—www.cs.ucsb.edu/research/javelin/	Middleware system
AppLeS	This is an application-specific approach to scheduling individual parallel applications on production heterogeneous systems —apples.ucsd.edu	Grid scheduler
NASA IPG	The Information Power Grid (IPG) is a testbed that provides access to a Grid—a widely distributed network of high-performance computers, stored data, instruments, and collaboration environments—www.ipg.nasa.gov	Application testbed
Condor	The Condor project is developing and deploying, and evaluating mechanisms and policies that support high-throughput computing on large collections of distributed resources—www.cs.wisc.edu/condor/	Middleware and scheduling system
Harness	Harness builds on the concept of the virtual machine and explores dynamic capabilities beyond what PVM can supply. It focuses on parallel plug-ins, P2P distributed control, and multiple virtual machines—www.epm.ornl.gov/harness/	Programming environment and runtime system
NetSolve	NetSolve is an Remote Procedure Call-based client/agent/server system that allows one to remotely access both hardware and software components—www.cs.utk.edu/netsolve/	Programming environment and runtime system
Gateway	Gateway offers a programming paradigm implemented over a virtual Web of accessible resources www.npac.syr.edu/users/haupt/WebFlow/demo.html	Web portal
WebFlow	WebFlow is an extension of the Web model that can act as a framework for wide-area distributed computing	Application runtime system
GridPort	The Grid Portal Toolkit (GridPort) is a collection of technologies designed to aid in the development of science portals on computational Grids: user portals, applications interfaces, and education portals—gridport.npaci.edu	Portal development environment
GrADS	The Grid Application Development Software (GrADS) is an adaptive programming and runtime environment—hipersoft.cs.rice.edu/grads/	User level middleware
JXTA	JXTA from Sun provides core infrastructure that are essential for creating P2P computing services and applications—www.jxta.org	Core middleware

Table VI. Grid related community forums.

Initiative	Focus and technologies developed
Global Grid Forum	This is a community-initiated forum of individual researchers and practitioners working on distributed computing, or 'Grid' technologies. This forum focuses on the promotion and development of Grid technologies and applications via the development and documentation of 'best practices', implementation guidelines, and standards with an emphasis on rough consensus and running code—www.gridforum.org
Peer-to-Peer Working Group (P2PWG)	The P2PWG is organized to facilitate and accelerate the advance of best practices for a P2P computing infrastructure. The group promotes best practice based on P2P computing. As computers become ubiquitous, ideas for implementation and use of P2P computing are developing rapidly and gaining prominence—www.p2pwg.org

- *Users can define their own classes*. As in other object-oriented systems users can override or redefine the functionality of a class. This feature allows functionality to be added or removed to meet a user's needs.

Legion core objects support the basic services needed by the metasystem. The Legion system supports the following set of core object types.

- *Classes and metaclasses*. Classes can be considered managers and policy makers. Metaclasses are classes of classes.
- *Host objects*. Host objects are abstractions of processing resources, they may represent a single processor or multiple hosts and processors.
- *Vault objects*. Vault objects represents persistent storage, but only for the purpose of maintaining the state of Object Persistent Representation (OPR).
- *Implementation objects and caches*. Implementation objects hide the storage details of object implementations and can be thought of as equivalent to executable files in UNIX. Implementation cache objects provide objects with a cache of frequently used data.
- *Binding agents*. A binding agent maps object IDs to physical addresses. Binding agents can cache bindings and organize themselves into hierarchies and software combining trees.
- *Context objects and context spaces*. Context objects map context names to Legion object IDs, allowing users to name objects with arbitrary-length string names. Context spaces consist of directed graphs of context objects that name and organize information.

Legion objects are independent, active, and capable of communicating with each other via unordered non-blocking calls. Like other object-oriented systems, the set of methods of an object describes its interface. The Legion interfaces are described in an Interface Definition Language (IDL). The Legion system uses an object-oriented approach, which potentially makes it ideal for designing and implementing complex distributed computing environments. However, using an object-oriented methodology does not come without a raft of problems, many of these being tied-up with the need for Legion to interact with legacy applications and services.

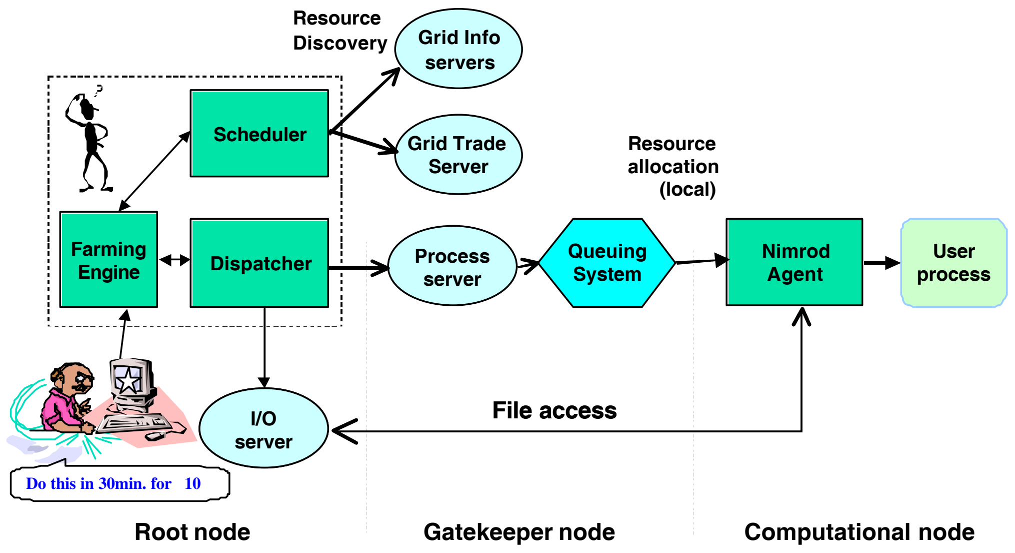

Figure 4. Nimrod-G Grid resource broker architecture.

The software developed under the Legion project has been commercialized by a spin-off company call Avaki Corporation [25]. Avaki has enhanced and developed Legion to take advantage of the emerging Grid and P2P technologies [26].

3.3. Nimrod-G and GRACE

Nimrod-G is a Grid resource broker that performs resource management and scheduling of parameter sweep, task-farming applications on worldwide Grid resources [27,28]. It consists of four key components: a task-farming engine, a scheduler, a dispatcher, and agents (see Figure 4 for the Nimrod-G broker architecture). A Nimrod-G persistent and programmable *task-farming engine* (TFE) enables 'plugging' of user-defined schedulers and customized applications or problem-solving environments (e.g. ActiveSheets [29]) in place of default components. The dispatcher uses the Globus services to deploy Nimrod-G agents on remote resources in order to manage the execution of assigned jobs. The local resource management system (e.g. queuing system or forking service) starts the execution of the Nimrod-G agent that interacts with the I/O server running on the user home/root-node to fetch a task script assigned to it (by the Nimrod-G scheduler) and executes the Nimrod commands specified in the script. The Nimrod-G scheduler has the ability to lease Grid resources and services depending on their capability, cost, and availability driven by user QoS requirements. It supports resource discovery, selection, scheduling, and transparent execution of user jobs on remote resources. The users can set the deadline by which the results are needed; the Nimrod/G broker then tries to find the cheapest computational resources available on the Grid and use them so that the user deadline is met and the cost of computation is kept to a minimum.

Table VII. Nimrod-G deadline and budget constrained scheduling algorithms.

Algorithm/strategy	Execution time (deadline, D)	Execution cost (budget, B)
Cost optimization	Limited by D	Minimize
Cost–time optimization	Minimize when possible	Minimize
Time optimization	Minimize	Limited by B
Conservative-time optimization	Minimize	Limited by B, but all unprocessed jobs have guaranteed minimum budget

Specifically, Nimrod-G supports user-defined deadline and budget constraints for schedule optimizations and manages the supply and demand of resources in the Grid using a set of distributed computational economy and resource trading services called GRACE (Grid Architecture for Computational Economy) [16]. The deadline and budget constrained (DBC) scheduling algorithms with four different optimization strategies [30,31]—cost optimization, cost-time optimization, time optimization, and conservative-time optimization—supported by the Nimrod-G resource broker for scheduling applications on the worldwide distributed resources are shown in Table VII. The *cost optimization* scheduling algorithm uses the cheapest resources to ensure that the deadline can be meet and the computational cost is minimized. The *time optimization* scheduling algorithm uses all the affordable resources to process jobs in parallel as early as possible. The *cost-time optimization* scheduling is similar to cost optimization, but if there are multiple resources with the same cost, it applies the time optimization strategy while scheduling jobs on them. The *conservative time optimization* scheduling strategy is similar to the time-optimization scheduling strategy, but it guarantees that each unprocessed job has a minimum budget-per-job. The Nimrod-G broker with these scheduling strategies has been used to solve large-scale data-intensive computing applications such as the simulation of ionization chamber calibration [27] and the molecular modeling for drug design [11].

3.4. GridSim

GridSim [32] is a toolkit for modeling and simulation of Grid resources and application scheduling. It provides a comprehensive facility for the simulation of different classes of heterogeneous resources, users, applications, resource brokers, and schedulers. It has facilities for the modeling and simulation of resources and network connectivity with different capabilities, configurations, and domains. It supports primitives for application composition, information services for resource discovery, and interfaces for assigning application tasks to resources and managing their execution. These features can be used to simulate resource brokers or Grid schedulers to evaluate performance of scheduling algorithms or heuristics. The GridSim toolkit has been used to create a resource broker that simulates Nimrod-G for design and evaluation of DBC scheduling algorithms with cost and time optimizations [31].

The GridSim toolkit resource modeling facilities are used to simulate the worldwide Grid resources managed as time- or space-shared scheduling policies. In GridSim-based simulations, the broker and user entities extend the GridSim class to inherit the ability to communicate with other entities. In GridSim, application tasks/jobs are modeled as *Gridlet* objects that contain all the information related to the job and the execution management details, such as job length in MI (million instructions), disk I/O operations, input and output file sizes, and the job originator. The broker uses GridSim's job management protocols and services to map a Gridlet to a resource and manage it throughout its lifecycle.

3.5. Gridbus

The Gridbus (GRID computing and BUSiness) toolkit project is engaged in the design and development of cluster and grid middleware technologies for service-oriented computing [33]. It provides end-to-end services to aggregate or lease services of distributed resources depending on their availability, capability, performance, cost, and users' QoS requirements. The key objective of the Gridbus project is to develop fundamental, next-generation cluster and grid technologies that support utility computing. The following initiatives are being carried out as part of the Gridbus project.

- At grid-level, the project extends our previous work on grid economy and scheduling to support: (a) different application models; (b) different economy models; (c) data models; and (d) architecture models—both grids and P2P networks.
- At resource level, the project supports a QoS-driven resource scheduler (e.g. economy-driven cluster scheduler [34]); it helps the user to negotiate and establish service-level agreement (SLA) with the resource scheduler in advance.
- A GridBank (GB) mechanism supports a secure Grid-wide accounting and payment handling to enable both cooperative and competitive economy models for resource sharing.
- The GridSim simulator is being extended to support simulation of these concepts for performance evaluation.
- GUI tools are being developed to enable distributed processing of legacy applications.
- The technologies are being applied to various application domains (high-energy physics, brain activity analysis, drug discovery, data mining, GridEmail, automated management of e-commerce).

3.6. UNICORE

UNICORE (UNiform Interface to COmputer REsources) [35] is a project funded by the German Ministry of Education and Research. It provides a uniform interface for job preparation, and seamless and secure access to supercomputer resources. It hides the system and site-specific idiosyncrasies from the users to ease the development of distributed applications. Distributed applications within UNICORE are defined as multipart applications where the different parts may run on different computer systems asynchronously or they can be sequentially synchronized. A UNICORE job contains a multipart application augmented by the information about the destination systems, the resource requirements, and the dependencies between the different parts. From a structural viewpoint a UNICORE job is a recursive object containing job groups and tasks. Job groups themselves consist of other job groups and tasks. UNICORE jobs and job groups carry the information of the destination system for the included tasks. A task is the unit, which boils down to a batch job for the destination system.

The design goals for UNICORE include a uniform and easy to use GUI, an open architecture based on the concept of an abstract job, a consistent security architecture, minimal interference with local administrative procedures, exploitation of existing and emerging technologies, a zero-administration user interface through a standard Web browser and Java applets, and a production ready prototype within two years. UNICORE is designed to support batch jobs, it does not allow for interactive processes. At the application level asynchronous metacomputing is supported, allowing for independent and dependent parts of a UNICORE job to be executed on a set of distributed systems. The user is provided with a unique UNICORE user-ID for uniform access to all UNICORE sites.

3.7. Information Power Grid

The NAS Systems Division is leading the effort to build and test NASA's IPG [7], a network of high-performance computers, data storage devices, scientific instruments, and advanced user interfaces. The overall mission of the IPG is to provide NASA's scientific and engineering communities with a substantial increase in their ability to solve problems that depend on the use of large-scale and/or distributed resources. The project team is focused on creating an infrastructure and services to locate, combine, integrate, and manage resources from across the NASA centers. An important goal of the IPG is to produce a common view of these resources, and at the same time provide for distributed management and local control. The IPG team at NAS is working to develop:

- independent but consistent tools and services that support a range of programming environments used to build applications in widely distributed systems;
- tools, services, and infrastructure for managing and aggregating dynamic collections of resources: processors, data storage/information systems, communications systems, real-time data sources and instruments, as well as human collaborators;
- facilities for constructing collaborative, application-oriented workbenches and problem-solving environments across NASA, based on the IPG infrastructure and applications;
- a common resource management approach that addresses areas such as:
    - systems management;
    - user identification;
    - resource allocations;
    - accounting;
    - security;
- an operational Grid environment that incorporates major computing and data resources at multiple NASA sites in order to provide an infrastructure capable of routinely addressing larger scale, more diverse, and more transient problems than is currently possible.

3.8. WebFlow

WebFlow [36] is a computational extension of the Web model that can act as a framework for the wide-area distributed computing. The main goal of the WebFlow design was to build a seamless framework for publishing and reusing computational modules on the Web so that end-users, via a Web browser, can engage in composing distributed applications using WebFlow modules as visual components and editors as visual authoring tools. WebFlow has a three-tier Java-based architecture that can be considered a visual dataflow system. The front-end uses applets for authoring, visualization, and control of the environment. WebFlow uses servlet-based middleware layer to manage and interact with back-end modules such as legacy codes for databases or high-performance simulations. WebFlow is analogous to the Web. Web pages can be compared to WebFlow modules and hyperlinks that connect Web pages to inter-modular dataflow channels. WebFlow content developers build and publish modules by attaching them to Web servers. Application integrators use visual tools to link outputs of the source modules with inputs of the destination modules, thereby forming distributed computational graphs (or compute-Webs) and publishing them as composite WebFlow modules. A user activates these compute-Webs by clicking suitable hyperlinks, or customizing the computation either in terms of available parameters or by employing some high-level commodity tools for visual graph authoring. The high-performance back-end tier is implemented using the Globus toolkit:

- MDS is used to map and identify resources;
- GRAM is used to allocate resources;
- GASS is used for a high-performance data transfer.

With WebFlow, new applications can be composed dynamically from reusable components just by clicking on visual module icons, dragging them into the active WebFlow editor area, and linking them by drawing the required connection lines. The modules are executed using Globus components combined with the pervasive commodity services where native high-performance versions are not available. The prototype WebFlow system is based on a mesh of Java-enhanced Web servers (Apache), running servlets that manage and coordinate distributed computation. This management infrastructure is implemented by three servlets: Session Manager, Module Manager, and Connection Manager. These servlets use URL addresses and can offer dynamic information about their services and current state. Each management servlet can communicate with others via sockets. The servlets are persistent and application-independent. Future implementations of WebFlow will use emerging standards for distributed objects and take advantage of commercial technologies, such as the CORBA [37] as the base distributed object model. WebFlow takes a different approach to both Globus and Legion. It is implemented in a hybrid manner using a three-tier architecture that encompasses both the Web and third-party back-end services. This approach has a number of advantages, including the ability to 'plug-in' to a diverse set of back-end services. For example, many of these services are currently supplied by the Globus toolkit, but they could be replaced with components from CORBA or Legion. WebFlow also has the advantage that it is more portable and can be installed anywhere a Web server supporting servlets is capable of running.

3.9. NetSolve

NetSolve [13] is a client/server application designed to solve computational science problems in a distributed environment. The Netsolve system is based around loosely coupled distributed systems, connected via a LAN or WAN. Netsolve clients can be written in C and Fortran, and use Matlab or the Web to interact with the server. A Netsolve server can use any scientific package to provide its computational software. Communications within Netsolve is via sockets. Good performance is ensured by a load-balancing policy that enables NetSolve to use the computational resources available as efficiently as possible. NetSolve offers the ability to search for computational resources on a network, choose the best one available, solve a problem (with retry for fault-tolerance), and return the answer to the user.

3.10. Ninf

The Ninf [38] is a client/server-based system for global computing. It allows access to multiple remote computational and database servers. Ninf clients can semitransparently access remote computational resources from languages such as C and Fortran. Global computing applications can be built easily by using the Ninf remote libraries as it hides the complexities of the underlying system.

3.11. Gateway—desktop access to high-performance computational resources

The Gateway system offers a programming paradigm implemented over a virtual Web of accessible resources [39]. A Gateway application is based around a computational graph visually edited by end-users, using Java applets. A module developer, a person who has only limited knowledge of the system on which the modules will run, writes modules. They need not concern themselves with issues such as: allocating resources, how to run the modules on various machines, creating inter-module connections, sending and receiving data between modules, or how to run several modules concurrently on a single machine. This is handled by WebFlow [36]. The Gateway system hides the configuration, management, and coordination mechanisms from the developers, allowing them to concentrate on developing their modules.

The goals of the Gateway system are:

- to provide a problem-oriented interface (a Web portal) to more effectively utilize high-performance computing resources from the desktop via a Web browser;
- this ‘point & click’ view hides the underlying complexities and details of the resources, and creates a seamless interface between the user’s problem description on their desktop system and the heterogeneous resources;
- the high-performance computing resources include computational resources such as supercomputers or workstation clusters, storage, such as disks, databases, and backing store, collaborative tools, and visualization servers.

Gateway is implemented as a three-tier system, as shown in Figure 5. Tier 1 is a high-level *front-end* for visual programming, steering, runtime data analysis and visualization, as well as collaboration. This tier is built on top of the Web and object-oriented commodity standards. Tier 2 is middleware and is based on distributed, object-based, scalable, and reusable Web servers and object brokers. Tier 3 consists of back-end services, such as those shown in Figure 5. The middle tier of the architecture is based on a network of Gateway servers. The user accesses the Gateway system via a portal Web page emanating from the secure gatekeeper Web server. The portal implements the first component of the Gateway security, user authentication and generation of the user credentials that is used to grant access to resources. The Web server creates a session for each authorized user and gives permission to download the front-end applet that is used to create, restore, run, and control user applications.

The main functionality of the Gateway server is to manage user sessions. A session is established automatically after the authorized user is connected to the system by creating a user context that is basically an object that stores the user applications. The application consists of one or more Gateway modules.

The Gateway modules are CORBA objects conforming to the JavaBeans model. The application’s functionality can be embedded directly into the body of the module or, more typically, the module serves as a proxy for specific back-end services. The Gateway servers also provide a number of generic

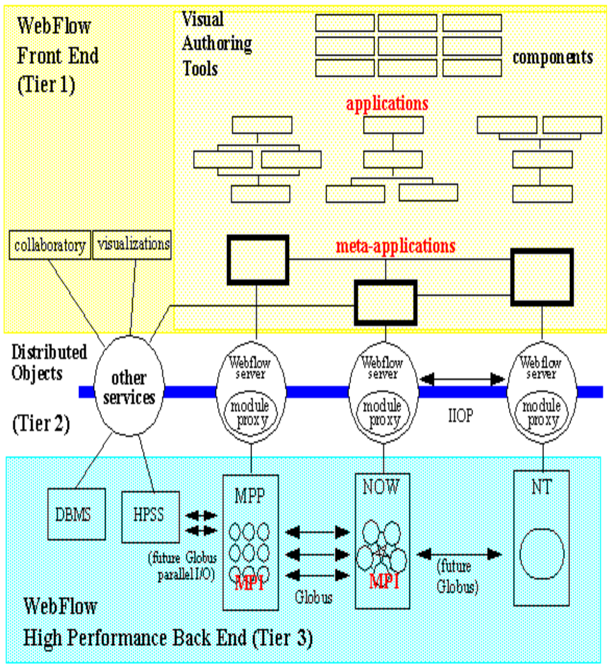

Figure 5. Three-tier architecture of Gateway.

services, such as access to databases and remote file systems. The most prominent service is the job service that provides secure access to high-performance computational servers. This service is accessed through a metacomputing API, such as the Globus toolkit API.

To interoperate with Globus there must be at least one Gateway node capable of executing Globus commands. To enable this interaction at least one host will need to run a Globus and Gateway server. This host serves as a 'bridge' between two domains. Here, Globus is an optional, high-performance (and secure) back-end, while Gateway serves as a high-level Web accessible visual interface and a job broker for Globus.

3.12. GridPort

The Grid Portal Toolkit (GridPort) is a collection of technologies designed to aid in the development of science portals on computational Grids: user portals, applications interfaces, and education portals [40]. The two key components of GridPort are the Web portal services and the application APIs. The Web portal module runs on a Web server and provides secure (authenticated) connectivity to the Grid. The application APIs provide a Web interface that help in developing customized science portals by end-users (without having the knowledge of the underlying portal infrastructure). The system is designed to allow execution of portal services and the client applications on separate Web servers.

The GridPort toolkit modules have been used to develop science portals for the applications areas such as molecular modeling, cardiac physiology, and tomography.

The GridPort modules are based on commodity Internet and Web technologies as well as existing Grid services and applications. The technologies used include HTML, JavaScript, Perl, CGI, SSH, SSL, FTP, GSI, and Globus. As Web technologies are easy to use and pervasive, client portals based on GridPort are accessible through any Web browser irrespective of location. By using the GridPort toolkit, application programmers can extend the functionality supported by the HotPage computational resource portal. A user can also customize Web pages and program portal services with a minimal knowledge of Web technologies.

3.13. DataGrid

The European DataGrid project, led by CERN, is funded by the European Union with the aim of setting up a computational and data-intensive Grid of resources for the analysis of data coming from scientific exploration [10]. The primary driving application of the DataGrid project is the Large Hadron Collider (LHC), which will operate at CERN from about 2005 to 2015. The LHC represents a leap forward in particle beam energy, density, and collision frequency. This leap is necessary in order to produce some examples of previously undiscovered particles, such as the Higgs boson or perhaps super-symmetric quarks and leptons. The LHC will present a number of challenges in terms of computing.

The main goal of the DataGrid initiative is to develop and test the infrastructure that will enable the deployment of scientific 'collaboratories' where researchers and scientists will perform their activities regardless of their geographical location. These collaboratories will allow personal interaction as well as the sharing of data and instruments on a global basis. The project is designing and developing scalable software solutions and testbeds in order to handle many petabytes of distributed data, tens of thousands of computing resources (processors, disks, etc.), and thousands of simultaneous users from multiple research institutions.

The DataGrid project is divided into 12 work packages (WPs) distributed over four working groups: testbed and infrastructure, applications, computational and DataGrid middleware, management and dissemination. Figure 6 illustrates the structure of the project and the interactions between the work packages (source [41]). The work has an emphasis on enabling the distributed processing of data-intensive applications in the area of high-energy physics, earth observation, and bio-informatics.

The main challenge facing the project is providing the means to share huge amounts of distributed data over the current network infrastructure. The DataGrid relies upon emerging Grid technologies that are expected to enable the deployment of a large-scale computational environment consisting of distributed collections of files, databases, computers, scientific instruments, and devices.

3.14. The Open Grid Services Architecture framework

The Open Grid Services Architecture (OGSA) framework [42], the Globus-IBM vision for the convergence of Web services and Grid computing was presented at the Global Grid Forum (GGF) meeting held in Toronto in February 2002. The GGF has set up an Open Grid Services working group to review and refine the Grid services architecture and documents that form the technical specification.

The OGSA supports the creation, maintenance, and application of ensembles of services maintained by virtual organizations (VOs). Here a service is defined as a network-enabled entity that provides some capability, such as computational resources, storage resources, networks, programs, and databases.

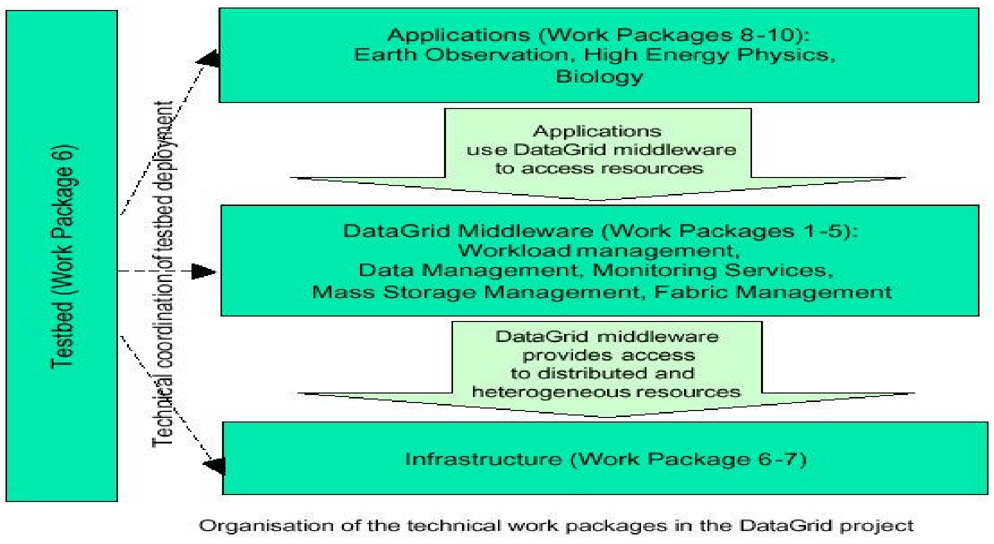
Testbed (Work Package 6)
Technical coordination of testbed deployment
Applications (Work Packages 8-10): Earth Observation, High Energy Physics, Biology
Applications use DataGrid middleware to access resources
DataGrid Middleware (Work Packages 1-5): Workload management, Data Management, Monitoring Services, Mass Storage Management, Fabric Management
DataGrid middleware provides access to distributed and heterogeneous resources
Infrastructure (Work Package 6-7)
Organisation of the technical work packages in the DataGrid project

Figure 6. The DataGrid organization.

The Web Services standards used in the OGSA include SOAP, WSDL, and WS-Inspection.

- The Simple Object Access Protocol (SOAP) provides a means of messaging between a service provider and a service requestor.
- The Web Services Description Language (WSDL) is an XML document for describing Web services as a set of endpoints operating on messages containing either document-oriented (messaging) or RPC payloads.
- WS-Inspection comprises a simple XML language and related conventions for locating service descriptions published by a service provider.

Web services enhance the OGSA in a number of ways. One of the main ones is that Web services has the ability to support the dynamic discovery and composition of services in heterogeneous environments. Web Services have mechanisms for registering and discovering interface definitions, endpoint descriptions, and for dynamically generating service proxies. WDSL provides a standard mechanism for defining interface definitions separately from their embodiment within a particular binding (transport protocol and data encoding format). In addition, Web services are being widely adopted, which means that their adoption will allow a greater level of interoperability and the capability to exploit new and emerging tools and services such as Microsoft .NET and Apache Axis.

The parts of Globus that are impacted on most by the OGSA are:

- the GRAM protocol;
- the information infrastructure, MDS-2, used for information discovery, registration, data modeling, and a local registry;
- GSI, which supports single sign-on, restricted delegation, and credential mapping.

The standard interfaces defined in OGSA are as follows.

- *Discovery*. Clients require mechanisms for discovering available services and for determining the characteristics of those services so that they can configure themselves and their requests to those services appropriately.
- *Dynamic service creation*. This is a standard interface (factory) and semantics that any service creation service must provide.
- *Lifetime management*. In a system that incorporates transient and stateful service instances, mechanisms must be provided for reclaiming services and state associated with failed operations.
- *Notification*. A collection of dynamic, distributed services must be able to notify each other asynchronously of interesting changes to their state.
- *Manageability*. The operations relevant to the management and monitoring of large numbers of Grid service instances are provided.
- *Simple hosting environment*. A simple execution environment is a set of resources located within a single administrative domain and supporting native facilities for service management: for example, a J2EE application server, Microsoft .NET system, or Linux cluster.

It is expected that the future implementation of Globus toolkit will be based on the OGSA architecture. Core services will implement the interfaces and behavior described in the Grid Service specification. Base services will use the Core services to implement both existing Globus capabilities, such as resource management, data transfer, and information services, as well as new capabilities such as resource reservation and monitoring. A range of higher level services will use the Core and Base services to provide data management, workload management, and diagnostics services.

4. GRID APPLICATIONS

A Grid platform could be used for many different types of applications. In [1], Grid-aware applications are categorized into five main classes:

- distributed supercomputing (e.g. stellar dynamics);
- high-throughput (e.g. parametric studies);
- on-demand (e.g. smart instruments);
- data intensive (e.g. data mining);
- collaborative (e.g. collaborative design).

A new emerging class of application that can benefit from the Grid is:

- service-oriented computing (e.g. application service provides and the users' QoS requirements driven access to remote software and hardware resources [15]).

There are several reasons for programming applications on a Grid, for example:

- to exploit the inherent distributed nature of an application;
- to decrease the turnaround/response time of a huge application;
- to allow the execution of an application which is outside the capabilities of a single (sequential or parallel) architecture;

- to exploit the affinity between an application component and Grid resources with a specific functionality.

Although wide-area distributed supercomputing has been a popular application of the Grid, a large number of other applications can benefit from the Grid [43,44]. Applications in these categories come from science, engineering, commerce, and educational fields.

The existing applications developed using the standard message-passing interface (e.g. MPI) for clusters, can run on Grids without change, since an MPI implementation for Grid environments is available. Many of the applications exploiting computational Grids are embarrassingly parallel in nature. The Internet computing projects, such as SETI@Home [45] and Distributed.Net [46], build Grids by linking multiple low-end computational resources, such as PCs, across the Internet to detect extraterrestrial intelligence and crack security algorithms, respectively. The nodes in these Grids work simultaneously on different parts of the problem and pass results to a central system for post-processing.

Grid resources can be used to solve grand challenge problems in areas such as biophysics, chemistry, biology, scientific instrumentation [27], drug design [11,47], tomography [48], high-energy physics [49], data mining, financial analysis, nuclear simulations, material science, chemical engineering, environmental studies, climate modeling [50], weather prediction, molecular biology, neuroscience/brain activity analysis [51], structural analysis, mechanical CAD/CAM, and astrophysics.

In the past, applications were developed as *monolithic* entities. A monolithic application is typically the same as a single executable program that does not rely on outside resources and cannot access or offer services to other applications in a dynamic and cooperative manner. The majority of the scientific and engineering (S&E) as well as business-critical applications of today are still monolithic. These applications are typically written using just one programming language. They are generally computational intensive, batch processed, and their elapsed times are measured in several hours or days. Good examples of applications in the S&E area are: Gaussian [52], PAM-Crash [53], and Fluent [54].

Today, the situation is rapidly changing and a new style of application development based on components has become more popular. With component-based applications, programmers do not start from scratch but build new applications by reusing existing *off-the-shelf* components and applications. Furthermore, these components may be distributed across a wide-area network. Components are defined by the public interfaces that specify the functions as well as the protocols that they may use to communicate with other components. An application in this model becomes a dynamic network of communicating objects. This basic distributed object design philosophy is having a profound impact on all aspects of information processing technology. We are already seeing a shift in the software industry towards an investment in software components and away from *handcrafted*, *stand-alone* applications. In addition, within the industry, a technology war is being waged over the design of the component composition architecture [55].

Meanwhile, we are witnessing an impressive transformation of the ways that research is conducted. Research is becoming increasingly interdisciplinary; there are studies that foresee future research being conducted in virtual laboratories in which scientists and engineers routinely perform their work without regard to their physical location. They will be able to interact with colleagues, access instrumentation, share data and computational resources, and access information in digital libraries. All scientific and technical journals will be available on-line, allowing readers to download documents and other forms of information, and manipulate it to interactively explore the published research [56]. This exciting vision has a direct impact on the next generation of computer applications and on the way that they will be designed and developed. The complexity of future applications will grow rapidly, and the time-to-market pressure will mean that applications can no longer be built *from scratch*. Hence, mainly for cost reasons, it is foreseeable that no single company or organization would be able to, for example, create by itself complex and diverse software, or hire and train all the necessary expertise necessary to build an application. This will heighten the movement towards component frameworks, enabling rapid construction from third-party ready-to-use components.

In general, such applications tend to be multidisciplinary multimodular, written by several development teams using several programming languages, using multisource heterogeneous data, which can be mobile and interactive. Their execution will take a few minutes or hours [57]. In particular, future S&E applications, for example, will be multidisciplinary. They will be composed of several different *disciplinary modules* coupled into a single modeling system (fluids and structures in an aeronautics code, e.g. [58,59]), or composed of several different *levels of analysis* combined within a single discipline (e.g. linear, Euler, and Navier–Stokes aerodynamics [60,61]. Some of these components will be characterized by high-performance requirements. Thus, in order to achieve better performance, the challenge will be to map each component onto the best candidate computational resource available on the Grid that has the highest degree of affinity with that software component. There are several examples of such integrated multidisciplinary applications reported in the literature, in several science and engineering fields including aeronautics (e.g. simulation of aircraft), geophysics (e.g. environmental and global climate modeling), biological systems, drug design, and plasma physics. In all these areas, there is a strong interest in developing increasingly sophisticated applications that couple ever more advanced simulations of very diverse physical systems. In [62,63] several fields where parallel and distributed simulation technologies have been successfully applied are reported. In particular, some applications belonging to areas such as the design of complex systems, education and training, entertainment, military, social and business collaborations, telecommunications, transportation, etc., are described.

5. CONCLUSIONS AND FUTURE TRENDS

There are currently a large number of projects and a diverse range of new and emerging Grid developmental approaches being pursued. These systems range from Grid frameworks to application testbeds, and from collaborative environments to batch submission mechanisms.

It is difficult to predict the future in a field such as information technology where the technological advances are moving very rapidly. Hence, it is not an easy task to forecast what will become the ‘dominant’ Grid approach. Windows of opportunity for ideas and products seem to open and close in the ‘blink of an eye’. However, some trends are evident. One of those is growing interest in the use of Java [64] and Web services [12] for network computing.

The Java programming language successfully addresses several key issues that accelerate the development of Grid environments, such as heterogeneity and security. It also removes the need to install programs remotely; the minimum execution environment is a Java-enabled Web browser. Java, with its related technologies and growing repository of tools and utilities, is having a huge impact on the growth and development of Grid environments. From a relatively slow start, the developments in Grid computing are accelerating fast with the advent of these new and emerging technologies. It is very hard to ignore the presence of the Common Object Request Broker Architecture (CORBA) [37] in the background. We believe that frameworks incorporating CORBA services will be very influential on the design of future Grid environments.

The two other emerging Java technologies for Grid and P2P computing are Jini [65] and JXTA [66]. The Jini architecture exemplifies a network-centric service-based approach to computer systems. Jini replaces the notions of peripherals, devices, and applications with that of network-available services. Jini helps break down the conventional view of what a computer is, while including new classes of services that work together in a federated architecture. The ability to move code from the server to its client is the core difference between the Jini environment and other distributed systems, such as CORBA and the Distributed Common Object Model (DCOM) [67].

Whatever the technology or computing infrastructure that becomes predominant or most popular, it can be guaranteed that at some stage in the future its star will wane. Historically, in the field of computer research and development, this fact can be repeatedly observed. The lesson from this observation must therefore be drawn that, in the long term, backing only one technology can be an expensive mistake. The framework that provides a Grid environment must be adaptable, malleable, and extensible. As technology and fashions change it is crucial that Grid environments evolve with them.

In [1], Smarr observes that Grid computing has serious social consequences and is going to have as revolutionary an effect as railroads did in the American midWest in the early 19th century. Instead of a 30–40 year lead-time to see its effects, however, its impact is going to be much faster. Smarr concludes by noting that the effects of Grids are going to change the world so quickly that mankind will struggle to react and change in the face of the challenges and issues they present. Therefore, at some stage in the future, our computing needs will be satisfied in the same pervasive and ubiquitous manner that we use the electricity power grid. The analogies with the generation and delivery of electricity are hard to ignore, and the implications are enormous. In fact, the Grid is analogous to the electricity (power) Grid and the vision is to offer (almost) dependable, consistent, pervasive, and inexpensive access to resources irrespective of their location for physical existence and their location for access.

ACKNOWLEDGEMENTS

The authors would like to acknowledge all developers of the systems or projects described in this article. We have had intellectual communication and exchanged views on this upcoming technology with David Abramson (Monash), Fran Berman (UCSD), David C. DiNucci (Elepar), Jack Dongarra (UTK), Ian Foster (ANL), Geoffrey Fox (Syracuse), Wolfgang Gentzsch (Sun), Jon Giddy (DSTC), Al Geist (ORNL), and Tom Haupt (Syracuse). We thank them for sharing their thoughts.

REFERENCES

1. Foster I, Kesselman C (eds.). *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann: San Francisco, CA, 1999.
2. Chetty M, Buyya R. Weaving computational Grids: How analogous are they with electrical Grids? *Journal of Computing in Science and Engineering (CiSE)* 2001; (July–August).
3. Oram A (ed.). *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly Press: U.S.A., 2001.
4. Foster I, Kesselman C, Tuecke S. The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications* 2001. To appear.

5. Buyya R, Stockinger H, Giddy J, Abramson D. Economic models for management of resources in peer-to-peer and Grid computing. *SPIE International Conference on Commercial Applications for High-Performance Computing*, Denver, CO, August 20–24, 2001.
6. Camarinha-Matos L, Afsarmanesh H (eds.). *Infrastructure For Virtual Enterprises: Networking Industrial Enterprises*. Kluwer Academic Press: Norwell, MA, 1999.
7. Johnston W, Gannon D, Nitzberg B. Grids as production computing environments: The engineering aspects of NASA's information power grid. *Eighth IEEE International Symposium on High Performance Distributed Computing*, Redondo Beach, CA, August 1999. IEEE Computer Society Press: Los Alamitos, CA, 1999.
8. Buyya R. The World-Wide Grid. http://www.buyya.com/ecogrid/wwg/.
9. NSF Tera-Grid. http://www.teraGrid.org/.
10. Hoschek W, Jaen-Martinez J, Samar A, Stockinger H, Stockinger K. Data management in an international data Grid project. *Proceedings of the 1st IEEE/ACM International Workshop on Grid Computing (Grid'2000)*, Bangalore, India, 17–20 December 2000. Springer: Berlin, 2000.
11. Buyya R. The Virtual Laboratory Project: Molecular modeling for drug design on Grid. *IEEE Distributed Systems Online* 2001; **2**(5). http://www.buyya.com/vlab/.
12. W3C. Web services activity. http://www.w3.org/2002/ws/.
13. Casanova H, Dongarra J. NetSolve: A network server for solving computational science problems. *International Journal of Supercomputing Applications and High Performance Computing* 1997; **11**(3).
14. Baker M, Fox G. Metacomputing: Harnessing informal supercomputers. *High Performance Cluster Computing: Architectures and Systems*, vol. 1, Buyya R (ed.). Prentice-Hall: Englewood Cliffs, NJ, 1999.
15. Buyya R, Giddy J, Abramson D. A case for economy grid architecture for service-oriented Grid computing. *10th IEEE International Heterogeneous Computing Workshop (HCW 2001), in Conjunction with IPDPS 2001*, San Francisco, CA, April 2001. IEEE Computer Society Press: Los Alamitos, CA, 2001.
16. Buyya R, Abramson D, Giddy J. Economy driven resource management architecture for computational power grids. *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2000)*, Las Vegas, NV, 2000. CSREA Press: Athens, GA, 2000.
17. Buyya R. Economic-based distributed resource management and scheduling for Grid computing. *PhD Thesis*, Monash University, Melbourne, Australia, April 2002.
18. PAPIA: Parallel protein information analysis system. http://www.rwcp.or.jp/papia/.
19. Gentzsch W (ed.). Special issue on metacomputing: From workstation clusters to internet computing. *Future Generation Computer Systems* 1999; **15**.
20. Buyya R (ed.). Grid Computing Info Centre. http://www.Gridcomputing.com/.
21. Baker M (ed.). Grid computing. IEEE DS Online. http://computer.org/dsonline/gc/.
22. Matsuoka S. Grid RPC meets DataGrid: Network enabled services for data farming on the Grid. *First IEEE/ACM International Conference on Cluster Computing and the Grid (CCGrid 2001)*, Brisbane, Australia, May 2001. IEEE Computer Society Press: Los Alamitos, CA, 2001.
23. Foster I, Kesselman C. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications* 1997; **11**(2):115–128.
24. Grimshaw A, Wulf W. The Legion vision of a worldwide virtual computer. *Communications of the ACM* 1997; **40**(1).
25. Avaki Corporation. http://www.avaki.com/.
26. Avaki Architecture. http://www.avaki.com/papers/AVAKI_concepts_architecture.pdf.
27. Abramson D, Giddy J, Kotler L. High performance parametric modeling with Nimrod/G: Killer application for the global Grid? *International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society Press: Los Alamitos, CA, 2000.
28. Buyya R, Abramson D, Giddy J. Nimrod/G: An architecture for a resource management and scheduling system in a global computational Grid. *The 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia'2000)*, Beijing, China, 2000. IEEE Computer Society Press: Los Alamitos, CA, 2000.
29. Abramson D, Roe P, Kotler L, Mather D. ActiveSheets: Super-computing with spreadsheets. *2001 High Performance Computing Symposium (HPC'01), Advanced Simulation Technologies Conference*, April 2001. SCS Press: San Diego, CA, 2001.
30. Buyya R, Giddy J, Abramson D. An evaluation of economy-based resource trading and scheduling on computational power Grids for parameter sweep applications. *The Second Workshop on Active Middleware Services (AMS 2000), in Conjunction with HPDC 2001*, 1 August 2000, Pittsburgh, PA. Kluwer Academic Press, 2000.
31. Buyya R, Murshed M, Abramson D. A deadline and budget constrained cost-time optimization algorithm for scheduling task farming applications on global Grids. *Technical Report*, Monash University, March 2002. http://www.buyya.com/gridsim/.
32. Buyya R, Murshed M. GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. *Concurrency and Computation: Practice and Experience* 2002; (May). To appear.

33. Buyya R. The Gridbus Toolkit: Enabling Grid computing and business. http://www.gridbus.org.
34. Sherwani J, Ali N, Lotia N, Hayat Z, Buyya R. Libra: An economy driven job scheduling system for clusters. *Technical Report*, The University of Melbourne, July 2002.
35. Almond J, Snelling D. UNICORE: Uniform access to supercomputing as an element of electronic commerce. *Future Generation Computer Systems* 1999; **15**:539–548.
36. Akarsu E, Fox G, Furmanski W, Haupt T. WebFlow—high-level programming environment and visual authoring toolkit for high performance distributed computing. *SC98: High Performance Networking and Computing*, Orlando, FL, 1998.
37. Object Management Group. Common Object Request Broker: Architecture and specification. *OMG Document No. 91.12.1*, 1991.
38. Sato A, Nakada H, Sekiguchi S, Matsuoka S, Nagashima U, Takagi H. Ninf: A network based information library for a global world-wide computing infrastructure. *High-Performance Computing and Networking* (*Lecture Notes in Computer Science*, vol. 1225). Springer, 1997.
39. Akarsu E, Fox G, Haupt T, Kalinichenko A, Kim K, Sheethaalnath P, Youn C. Using Gateway system to provide a desktop access to high performance computational resources. *The 8th IEEE International Symposium on High Performance Distributed Computing (HPDC-8)*, Redondo Beach, CA, August 1999.
40. Thomas M, Mock S, Boisseau J. Development of Web toolkits for computational science portals: The NPACI HotPage. *The 9th IEEE International Symposium on High Performance Distributed Computing (HPDC 2000)*, Pittsburgh, PA, 1–4 August, 2000.
41. The DataGrid project. http://eu-datagrid.web.cern.ch/.
42. Foster I, Kesselman C, Nick J, Tuecke S. The physiology of the Grid: An open Grid services architecture for distributed systems integration. http://www.globus.org/research/papers.html#OGSA [January 2002].
43. Leinberger W, Kumar V. Information Power Grid: The new frontier in parallel computing? *IEEE Concurrency* 1999; **7**(4).
44. Brown MD *et al*. The International Grid (iGrid): Empowering global research community networking using high performance international Internet services. http://www.globus.org/research/papers.html [April 1999].
45. SETI@Home. http://setiathome.ssl.berkeley.edu/.
46. Distributed.Net. http://www.distributed.net/.
47. Buyya R, Branson K, Giddy J, Abramson D. The Virtual Laboratory: A toolset to enable distributed molecular modelling for drug design on the World-Wide Grid. *Concurrency and Computation: Practice and Experience* 2002.
48. Smallen S *et al*. Combining workstations and supercomputers to support Grid applications: The parallel tomography experience. *The 9th Heterogenous Computing Workshop (HCW 2000, IPDPS)*, Cancun, Mexico, April 2000.
49. Holtman K. CMS DataGrid system overview and requirements. *The Compact Muon Solenoid (CMS) Experiment Note 2001/037*, CERN, Switzerland, 2001.
50. Allcock B, Foster I, Nefedova V, Chervenak A, Deelman E, Kesselman C, Lee J, Sim A, Shoshani A, Drach B, Williams D. High-performance remote access to climate simulation data: A challenge problem for data Grid technologies. *Proceedings of SC2001 Conference*, Denver, CO, November 2001.
51. Date S, Buyya R. Economic and on demand brain activity analysis on the Grid. *The 2nd Pacific Rim Application and Grid Middleware Assembly Workshop*, Seoul, Korea, July 2002.
52. Gaussian. http://www.gaussian.com.
53. Pam Crash. http://www.esi-group.com/products/crash/overview.htm.
54. Fluent. http://www.fluent.com.
55. Gannon D. Component architectures for high performance, distributed meta-computing. http://www.objs.com/workshops/ws9801/papers/paper086.html.
56. PITAC. Information technology: Transforming our society. *Information Technology Advisory Committee Interim Report to the U.S. President*, August 1998. http://www.ccic.gov/ac/report/section_1.html.
57. Darema F. Next generation software research directions. http://www.cise.nsf.gov/eia/NGS-slides/sld001.htm.
58. NASA's Numerical Propulsion System Simulation (NPSS). http://cict.grc.nasa.gov/npssintro.shtml.
59. Perez C, Priol T. JACO3: A Grid environment that supports the execution of coupled numerical simulation. http://www.ercim.org/publication/Ercim/_News/enw45/priol.html.
60. Fox GC, Williams RD, Messina PC. *Parallel Computing Works*. Morgan Kaufmann, 1994.
61. Foster I, Zang T. Building multidisciplinary applications. http://www-fp.mcs.anl.gov/hpcc/section2.6.5.html.
62. Fujimoto RM. *Parallel and Distributed Simulation Systems*. Wiley, 2000; 300.
63. Laforenza D. Programming high performance applications in Grid environments. *Invited Talk at EuroPVM/MPI Conference*, Greece, April 2002. http://www.gridforum.org/7_APM/aps.htm.
64. Arnold K, Gosling J. *The Java Programming Language*. Addison-Wesley–Longman: Reading, MA, 1996.
65. Waldo J. The JINI architecture for network-centric computing. *Communications of the ACM* 1999; **42**(7).
66. Gong L. Project JXTA: A technology overview. *Sun Whitepaper*, August 2001. http://www.jxta.org/.
67. Rogerson D. *Inside COM*. Microsoft Press: Washington State, U.S.A., 1997.