Chapter 8
The Role of Grid Computing Technologies in Cloud Computing

David Villegas, Ivan Rodero, Liana Fong, Norman Bobroff, Yanbin Liu, Manish Parashar, and S. Masoud Sadjadi

Abstract The fields of Grid, Utility and Cloud Computing have a set of common objectives in harnessing shared resources to optimally meet a great variety of demands cost-effectively and in a timely manner Since Grid Computing started its technological journey about a decade earlier than Cloud Computing, the Cloud can benefit from the technologies and experience of the Grid in building an infrastructure for distributed computing. Our comparison of Grid and Cloud starts with their basic characteristics and interaction models with clients, resource consumers and providers. Then the similarities and differences in architectural layers and key usage patterns are examined. This is followed by an in depth look at the technologies and best practices that have applicability from Grid to Cloud computing, including scheduling, service orientation, security, data management, monitoring, interoperability, simulation and autonomic support. Finally, we offer insights on how these techniques will help solve the current challenges faced by Cloud computing.

8.1 Introduction

Cloud computing exploits the advances in computing hardware and programming models that can be brought together to provide utility solutions to large-scale computing problems. At the hardware level, the last half century has seen prolific progress in computing power. This results from many improvements at the processor

D. Villegas (✉) and S. Masoud Sadjadi
CIS, Florida International University, Miami, FL, USA
e-mails: {dvill013; sadjadi}@cis.fiu.edu

I. Rodero and M. Parashar
NSF CAC, Rutgers University, Piscataway, NJ, USA
e-mails: {irodero; parashar}@rutgers.edu

L. Fong, N. Bobroff, and Y. Liu
IBM Watson Research Center, Hawthorne, NY, USA
e-mails: {llfong; bobroff; ygliu}@us.ibm.com

B. Furht, A. Escalante (eds.), *Handbook of Cloud Computing*,
DOI 10.1007/978-1-4419-6524-0_8,

level, and in recent years the availability of low cost multi-core circuits. Additional progress in high speed, low latency interconnects, has allowed building large-scale local clusters for distributed computing, and the extension to wide-area collaborating clusters in the Grid. Now, the recent availability of hardware support for platform virtualization on commodity machines provides a key enabler for Cloud based computing.

Software models move in lockstep to match advances in hardware. There is a considerable practical experience implementing distributed computing solutions and in supporting parallel programming models on clusters. These models now work to leverage the concurrency provided by multi-core and multi-systems. Additionally, there are two other areas of software evolution that are moving quickly to support the Cloud paradigm: one is the improving maturity and capability of software to manage virtual machines, and the other is the migration from a monolithic approach in constructing software solutions to a service approach in which complex processes are composed of loosely coupled components.

These latest steps in the evolution of hardware and software models have led to Grid and Cloud Computing as paradigms that reduce the cost of software solutions. Harnessing shared computing resources from federated organizations to execute applications is the key concept of Grid Computing, as proposed by Foster, Kesselman, and Tuecke (2001): "Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations...The sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs."

Evolving from the technologies of Grid computing, Utility Computing is "a business model in which computing resources are packaged as metered services" (Foster, Zhao, Raicu, & Lu, 2008) to meet on demand resource requirements. The metered resource usage is similar to electric and water utility in delivery and the payment model is pay-as-you-go. The Utility Computing projects also introduced and demonstrated the ideas of dynamic provisioning of computing resources (Appleby et al., 2001).

Armbrust et al. (2009) defined Cloud Computing as providing application software delivered as services over the Internet, and the software and hardware infrastructure in the data centers that provide those services using business models that are similar to Utility Computing. Metering of services to support pay-as-you-go business models is suitable to application software (i.e., software as services, SaaS), platform (i.e., platform as services, PaaS), and infrastructure (i.e., infrastructure as services, IaaS). Moreover, Cloud Computing leverages emerging technologies such as the Web 2.0 for application services, and virtualization and dynamic provisioning support for platform services (Buyya, Yeo, Srikumar Venugopal, & Brandic, 2009).

Grid Computing, Utility Computing and Cloud Computing differ in aspects as their architectures, the types of coordinated institutions, the types of resources shared, the cost/business models, and the technologies used to achieve their objectives. However, all these computing environments have the common objectives in harnessing shared resources to optimally meet a variety of demands cost-effectively

and at timely manner. Since Grid Computing started its technological journey about a decade earlier than Cloud Computing, are there lessons to learn and technologies to leverage from Grid to Cloud? In this chapter, we would like to explore the experiences learnt in Grid and the role of Grid technologies for Cloud computing.

The rest of this chapter is organized as follows:

- Introductory discussion on the basics of Grid and Cloud computing, and their respective interaction models between client, resource consumer and provider (Section 8.2)
- Comparison of key processes in Grid and Cloud computing (Section 8.3)
- Core Grid technologies and their applicability to Cloud computing (Section 8.4)
- Concluding remarks on the future directions of Grid and Cloud computing (Section 8.5).

8.2 Basics of Grid and Cloud Computing

8.2.1 Basics of Grid Computing

Grid Computing harnesses distributed resources from various institutions (resource providers), to meet the demands of clients consuming them. Resources from different providers are likely to be diverse and heterogeneous in their functions (computing, storage, software, etc.), hardware architectures (Intel x86, IBM PowerPC, etc.), and usage policies set by owning institutions. Developed under the umbrella of Grid Computing, information services, name services, and resource brokering services are important technologies responsible for the aggregation of resource information and availability, selection of resources to meet the clients' specific requirements and the quality of services criteria while adhering to the resource usage policies.

Figure 8.1 shows an exemplary relationship of resource providers and consumers for a collaborative Grid computing scenario. Clients or users submit their requests for application execution along with resource requirements from their home domains. A Resource broker selects a domain with appropriate resources to acquire from and to execute the application or route the application to domain for execution with results and status returning to the home domain.

8.2.2 Basics of Cloud Computing

IDC[1] defined two specific aspects of Clouds: Cloud Services and Cloud Computing. Cloud Services are "consumer and business products, services and solutions that are delivered and consumed in real-time over the Internet" while Cloud Computing is "an emerging IT development, deployment and delivery model, enabling real-time

[1] http://blogs.idc.com/ie/?p=190

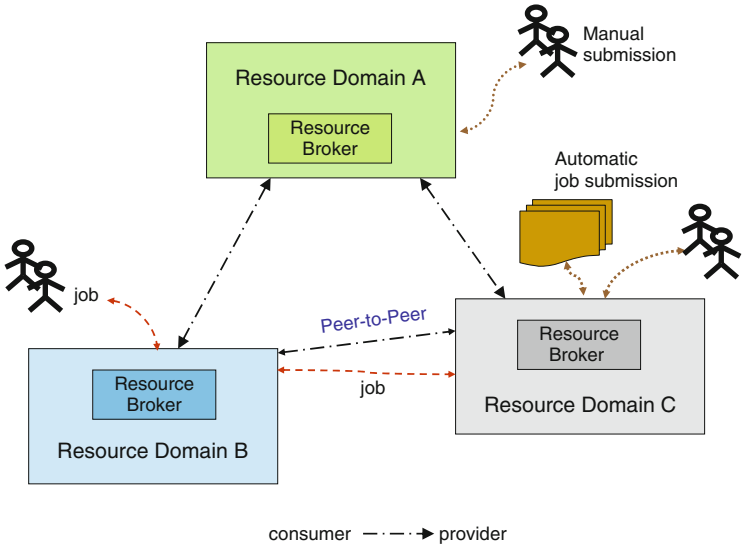

Fig. 8.1 Grid collaborating domains

delivery of products, services and solutions over the Internet (i.e., enabling Cloud services)". In this chapter, we will focus the computing infrastructure and platform aspects of the Cloud.

Amazon's Elastic Compute Cloud[2] popularized the Cloud computing model by providing an on-demand provisioning of virtualized computational resources as metered services to clients or users. While not restricted, most of the clients are individual users that acquire necessary resources for their own usage through EC2's APIs without cross organization agreements or contracts. Figure 8.2 illustrates possible usage models from clients C1 and C2 for resources/services of Cloud providers. As Cloud models evolve, many are developing the hybrid Cloud model in which enterprise resource brokers may acquire additional needed resources from external Cloud providers to meet the demands of submitted enterprise workloads (E1) and client work requests (E2). Moreover, the enterprise resource domain and Cloud providers may all belong to one corporation and thus form a private Cloud model.

8.2.3 Interaction Models of Grid and Cloud Computing

One of the most scalable interaction models of Grid domains is peer-to-peer, where most of the Grid participating organizations are both consumers and providers. In practice, there are usually agreements of resource sharing among the

[2] http://www.amazon.com/ec2

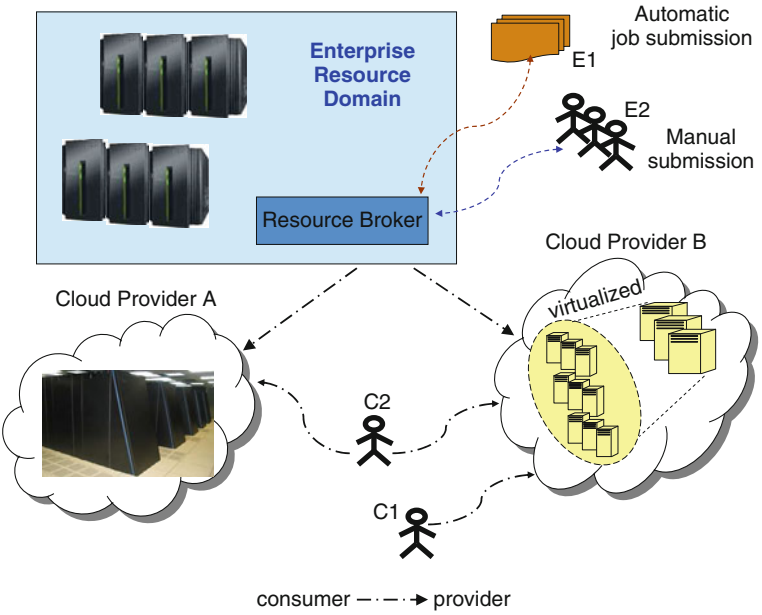

Fig. 8.2 Cloud usage models

peers. Furthermore, clients of consumer organizations in Grids use heterogeneous resources from more than one resource provider belonging to the same Virtual Organization (VO) to execute their applications. It is important for participating resource providers and consumers to have common information models, interaction protocols, application execution states, etc. The organization of Open Grid Forum (OGF)[3] has the goal of establishing relevant and necessary standards for Grid computing. Some proposed standards include Job Submission Description Language (JSDL), Basic Execution Service (BES) and others.

Currently, most of the Cloud providers offer their own proprietary service protocols and information formats. As Cloud computing becomes mature and widely adopted, clients and consumer organizations would likely interact with more than one provider for various reasons, including finding the most cost effective solutions or acquiring a variety of services from different providers (e.g., compute providers or data providers). Cloud consumers will likely demand common protocols and standardized information formats for ease of federated usage and interoperability. The Open Virtualization format (OVF) of the Distributed Management Task Force (DMTF)[4] is an exemplary proposal in this direction. Modeled after similar formations in the Grid community, OGF officially launched a workgroup, named

[3] http://www.ogf.org/

[4] http://www.dmtf.org/standards/

the Open Cloud Computing Interface Working Group (OCCI-WG)[5] to develop the necessary common APIs for the lifecycle management of Cloud infrastructure services. More standardization activities related to Cloud can be found in the wiki of Cloud-Standards.org.[6]

8.2.4 Distributed Computing in the Grid and Cloud

The Grid encompasses two areas of distributed system activity. One is operational with an objective of administrating and managing an interoperable collection of distributed compute resource clusters on which to execute client jobs, typically scientific/HPC applications. The procedures and protocols required to support clients from complex services built on distributed components that handle job submission, security, machine provisioning, and data staging. The Cloud has similar operational requirements for supporting complex services to provide clients with services on different levels of support such application, platform and infrastructure. The Grid also represents as a coherent entity a collection of compute resources that may be under different administrative domains, such as universities, but inter-operate transparently to form virtual organizations. Although interoperability is not a near term priority, there is a precedent for commercial Clouds to move in this direction similarly to how utilities such as power or communication contract with their competitors to provide overflow capacity.

The second aspect of distributed computing in the Grid is that job themselves are distributed, typically running on tightly coupled nodes within a cluster and leveraging middleware services such as MPICH. Jobs running in the Grid are not typically interactive, and some may be part of more complex services such as e-science workflows. Workloads in Clouds usually consist of more loosely coupled distributed jobs such as map/reduce, and HPC jobs written to minimize internode communication and leverage concurrency provided by large multi-core nodes. Service instances that form components of a larger business process workflow are likely to be deployed in the Cloud. These workload aspects of jobs running in the Cloud or Grid have implications for structuring the services that administer and manage the quality of their execution.

8.3 Layered Models and Usage patterns in Grid and Cloud

There are many similarities in Grid and Cloud computing systems. We compare the approaches by differentiating three layers of abstraction in Grid: Infrastructure, Platform and Application. Then we map these three layers to the Cloud services of IaaS, PaaS, and SaaS. An example of the relations among layers can be seen in Fig. 8.3.

[5] http://www.occi-wg.org/doku.php?id=start

[6] http://cloud-standards.org/

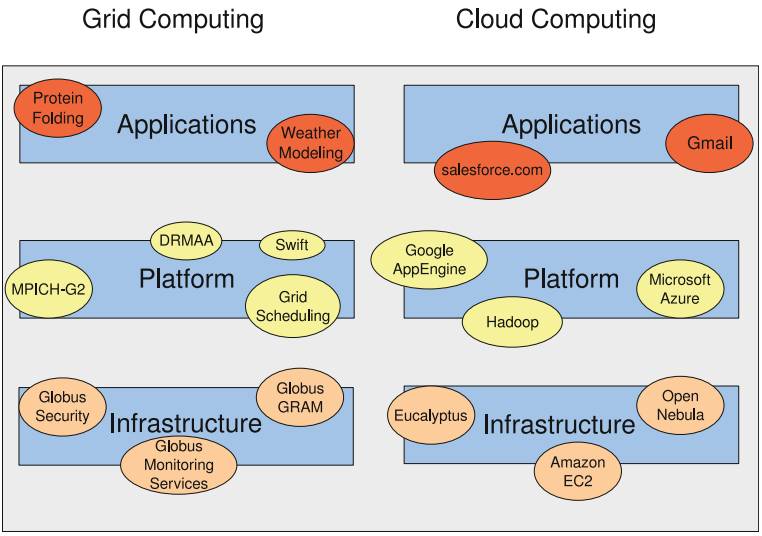

Fig. 8.3 Grid and cloud layers

8.3.1 Infrastructure

This is the layer in which Clouds share most characteristics with the original purpose of Grid middleware. Some examples are Eucalyptus (Nurmi et al., 2009), OpenNebula,[7] or Amazon EC2. In these systems users can provision execution environments in the form of virtual machines through interfaces such as APIs or command line tools. The act of defining an execution environment and sending a request to the final resource has many similarities with scheduling a job in the Grid. The main steps, shared by all of the cited Cloud environments are discussed below. We use Globus as the reference Grid technology.

- The user needs to be authorized to use the system. In Grid systems this is managed through the Community Authorization System (CAS) or by contacting a Certificate Authority that is trusted by the target institution, which issues a valid certificate. Clouds usually offer web forms to allow the registration of new users, and have additional web applications to maintain databases of customers and generate credentials, such as the case of Eucalyptus or Amazon.
- Once the user has a means of authenticating he needs to contact a gateway that can validate him and process his request. Different mechanisms are employed to carry users' requests, but Web Services are the most common of them. Users either write a custom program that consumes the WS offered by providers, or use available tools. Examples include the Amazon API tools for Amazon EC2, the euca2ools for Eucalyptus or the OpenNebula command line interface. Similarly,

[7] http://www.opennebula.org

Globus offers a set of console-based scripts that facilitate communication with the Grid.

- As part of the request for resource usage, users need to specify the action or task to be executed on the destination resources. Several formats are available for this purpose. Globus supports a Resource Specification Language (RSL) and a Job Submission Description Language (JSDL) that can define what process is to be run on the target machine, as well as additional constraints that can be used by a matchmaking component to restrict the class of resources to be considered, based on machine architecture, processor speed, amount of memory, etc. Alternatively, Clouds require different attributes such as the size of the execution environment or the virtual machine image to be used.
- After the job execution or the environment creation requests are received, there is a match-making and scheduling phase involved. The GRAM component from Globus is specially flexible in this regard, and multiple adapters allow different treatments for jobs: for example, the simplest job manager just performs a *fork* call to spawn a new process on the target machine. More advanced and widely used adapters transfer job execution responsibility to a local resource manager such as Condor, LoadLeveler or Sun Grid Engine. These systems are able of multiplexing jobs that are sent to a site into multiple resources. Cloud systems have simpler job management strategies, since the type of jobs are homogeneous and don't need to be adapted to a variety of resources such as in the case of the Grid. For example, Eucalyptus uses a Round Robin scheduling technique to alternate among machines. OpenNebula implements a Rank Scheduling Policy to choose the most adequate resource for a request, and supports more advance features such as advance reservations through Haizea (Sotomayor, Keahey, & Foster, 2008).
- One of the common phases involved in job submission is transferring the necessary data to and from the execution machine. The first of them, usually called *stage-in*, involves retrieving the input data for the process from a remote destination, such a GridFTP server. When the amount of data is large, a mapping service such as a Replica Location Service (RLS) can be used to translate a logical file name to a location. The second part of the process, *stage-out*, consists in either transferring the output data to the user's machine or to place it in a repository, possibly using the RLS. In the case of Cloud computing, the most important data that has to be transferred is the definition of an execution environment, usually in terms of Virtual Machine images. Users upload the data describing the operating system and packages needed to instantiate the VM and later reference it to perform operations such as booting a new machine. There is no standard method for transferring data in Cloud systems, but it is worth noting Amazon's object storage solution, the Simple Storage Service (S3), which allows users to move entities from 1 byte to 5 GB in size.
- Finally, Grid and Cloud systems need to offer users a method to monitor their jobs, as well as their resource usage. This facility can also be used by site administrators to implement usage accounting in order to track resource utilization and enforce user quotas. In the context of Globus, there are two

modules that can be used for this purpose, the first is GRAM itself, which allows user to query previously submitted jobs' status. The second method of acquiring information about the Grid's resources is provided by the Monitoring and Discovery Service (MDS), which is in charge of aggregating resources' data and making it available to be queried. High-level monitoring tools have been developed on top of existing Cloud management systems such as Amazon CloudWatch.

8.3.2 Platform

This layer is built on top of the physical infrastructure and offers a higher level of abstraction to users. The interface provided by a PaaS solution allows developers to build additional services without being exposed to the underlying physical or virtual resources. These facts enable additional features to be implemented as part of the model, such as presenting seemingly infinite resources to the user or allowing elastic behavior on demand. Examples of Cloud solutions that present these features are Google App Engine,[8] Salesforce's force.com[9] or Microsoft Azure.[10]

Several solutions that can be compared to the mentioned PaaS offerings exist in the Grid, even though this exact model cannot be exactly replicated. We define Platform level solutions as those containing the following two aspects:

8.3.2.1 Abstraction from Physical Resources

The Infrastructure layer provides users with direct access to the underlying infrastructure. While this is required for the lower levels of resource interaction, in the Platform level a user should be isolated from them. This allows developers to create new software that is not susceptible to the number of provisioned machines or their network configuration, for example.

8.3.2.2 Programming API to Support New Services

The Platform layer allows developers to build new software that takes advantage of the available resources. The choice of API directly influences the programs that can be built on the Cloud, therefore each PaaS solution is usually designed with a type of application in mind.

With these characteristics Grid systems allow developers to produce new software that take advantage of the shared resources in order to compare them with PaaS solutions.

---

[8] http://code.google.com/appengine/

[9] http://www.force.com/

[10] http://www.microsoft.com/windowsazure/

- Libraries are provided by Grid middleware to access resources programmatically. The Globus Java Commodity Grid (CoG) Kit (Laszewski et al., 2001) is an example. The CoG Kit allows developers to access the Grid functionality from a higher level. However, resources have to be independently addressed, which makes programs tied to the destination sites. Additionally, it is linked to Globus and makes applications dependent on a specific middleware.
- SAGA (Goodale et al., 2006) and DRMAA[11] are higher level standards that aim to define a platform independent set of Grid operations. While the former offers a wide range of options such as job submission, security handling or data management, the later focuses on sending and monitoring jobs. These solutions provide a higher level of abstraction than the previous example, but are still tied to the Grid concept of jobs as programs that are submitted to remote resources.
- An example of an API that bypasses the underlying Grid model to offer programmers a different paradigm to develop new software is MPICH-G2 (Karonis, Toonen, & Foster, 2003). It consists of a library that can be linked to a program that uses the Message Passing Interface (MPI) to transparently enable the application to work on the Grid. The programmer can think in familiar terms even though applications are Grid enabled.
- GridSuperscalar (Badia et al., 2003) is a programming paradigm to enable applications to run on the Grid. Programmers identify the functions of their code which can be run on remote resources, then specify the data dependencies for each of those functions, and after writing the code a runtime module determines the data dependencies and places each of the tasks in the Grid, transferring data accordingly so that each task can be completed.
- Another programming paradigm aimed at building new functionality on top of the Grid is SWIFT (Zhao et al., 2007). It provides a language to define computations and data dependencies, and is specially designed to efficiently run very large numbers of jobs while easing the task of defining the order of execution or the placing of data produced from one job to be consumed by another.

Probably the main difference in Cloud PaaS paradigms compared to the described options is that Grid models need to use the lowest common denominator when implementing new services. The reason for this is that the degree of compatibility with the middleware is directly related to the number of resources available: if a user's service does not make any assumptions on the remote resources, it will be able to use all of them normally; on the other hand, services requiring additional software to be installed on the target machines would have considerably fewer candidates to for execution.

In the case of Clouds, this requirement is not as stringent for two main reasons: the first one is that PaaS solutions are deeply tied to Cloud vendors, and therefore they are designed in hand with the rest of the infrastructure, and the second is that provisioning resources with the required libraries is much easier than in Grid computing, allowing new nodes to be spawned with the required environment. In

[11] http://www.drmaa.org/

the case of Grids, having the required software installed in the execution resources usually involves having a human operator do it, making the process more costly.

8.3.3 Applications

There is no clear distinctions between applications developed on Grids and those that use Clouds to perform execution and storage. The choice of platform should not influence the final result, since the computations delegated to the underlying systems can take different shapes to accommodate to the available APIs and resources.

On the other hand, it is undeniable that the vast majority of Grid applications fall in the realm of scientific software, while software running in Clouds has leaned towards commercial workloads. Here we try to identify some possible causes for the different levels of adoption of these technologies for the development of applications:

- **Lack of business opportunities in Grids.** Usually Grid middleware is installed only in hardware intended for scientific usage. This phenomenon has not successfully produced business opportunities that could be exploited by industry. Conversely, Clouds are usually backed up by industry which have had better ways to monetize their investments.
- **Complexity of Grid tools.** Perhaps due to the goal of providing a standardized, one-size-fits-all solution, Grid middleware is perceived by many as complex and difficult to install and manage. On the other hand, Cloud infrastructures have usually been developed by providers to fit their organization's needs and with a concrete purpose in mind, making them easier to use and solution oriented.
- **Affinity with target software**. Most Grid software is developed with scientific applications in mind, which is not true for the majority of Cloud systems. Scientific programs need to get the most performance from execution resources and many of them cannot be run on Clouds efficiently, for example because of virtualization overhead. Clouds are more targeted to web applications. These different affinities to distinct paradigms make both solutions specially effective for their target applications.

8.4 Techniques

Here we discuss the impact of techniques used in Grid computing that can be applied in Clouds. From the time the concept of Grid was introduced, a variety of problems had to be solved in order to enable its wide adoption. Some examples of these are user interfacing (Section 8.4.1), data transfer (Section 8.4.2), resource monitoring (Section 8.4.3) or security (Section 8.4.7). These basic techniques for the enablement of Grids were designed to fulfill its main goals, namely, to allow the sharing of heterogeneous resources among individuals belonging to remote administrative domains. These goals determine the areas of application of the described techniques

in Clouds, therefore we will find the most valuable set of improvements to be in the field of Cloud interoperability.

Clouds can not only benefit from the most fundamental techniques in Grid computing: additional techniques that arose on top of these building blocks to bring new functionality to Grids are also good candidates to be applied to Clouds. Among these we can find Autonomic Computing (Section 8.4.4), Grid scheduling (Section 8.4.5), interoperation (Section 8.4.6) or simulation (Section 8.4.8).

The techniques discussed in this section are therefore spread through various levels of the Grid architecture: some of them can be found in the lower layers, giving common services to other components, and others are built from the former and extend them. Following the classification discussed in Section 8.3, we find that some techniques belong to the Infrastructure layer, this is, have the main objective of resource management, and others are spread through the Infrastructure and Platform layers, such as the Metascheduling techniques described in the scheduling section.

8.4.1 Service Orientation and Web Services

The Cloud is both a provider of services (e.g. IaaS, PaaS, and SaaS) and a place to host services on behalf of clients. To implement the former operational aspects while maintaining flexibility, Cloud administrative functions should be constructed from software components. The Grid faced similar challenges in building a distributed infrastructure to support and evolve its administrative functions such as security, job submission, and creation of Virtual Organizations. The architectural principle adopted by the Grid is Service Orientation (SO) with software components connected by Web Services (WS). This section summarizes contributions of the Open Grid Forum (OGF) to SO in distributed computing and and how they apply to the Cloud. SO as an architecture, and Web Services as a mechanism of inter-component communication are explored here in the context of similarities between Grid and Cloud requirements.

Grid designers realized the advantage of the loosely-coupled client and service model being appropriately deployed in the distributed computing environments. The original Grid approach to SO was Open Grid Services Infrastructure (OGSI). OGSI was built on top of the emerging Web Services standards for expressing interfaces between components in a language neutral way based on XML schemas. While WS is an interface, OGSI attempted to make it object oriented by adding required methods. Subsequently, the Grid community worked within the WS standards to extend WS specification based on experience using a SOA. This lead to the introduction of Open Grid Services Architecture (OGSA), implemented in version 3 of the Globus toolkit. OGSA contains key extensions to the WS standard which are now described.

In Grid and Cloud the most typical service components such as provisioning an OS image, starting a virtual machine, or dispatching a job are long running. A service composition of these components requires an asynchronous programming

model. A consumer service component invokes a WS provider and is immediately acknowledged so the caller does not hold his process open on a communication link. The provider component asynchronously updates the consumer as the state changes. Grid architects recognized the importance of supporting the asynchronous model and integrated this approach into Web Services through the *WS-Addressing* and *WS-Notify* extensions. WS-Addressing specifies how to reference not just service endpoints, but objects within the service endpoint. Notification is based on WS-Addressing which specifies the component to be notified on a state change.

Related to the long lived service operation and asynchronous communication model is the requirement to maintain and share state information. There are many ways to achieve statefulness, none simple, especially when multiple services can update the same state object. In principle, a WS interface is stateless although of course there are many ways to build applications on top of WS that pass state through the operation messages. The challenge is to integrate the WS specification with a standard for statefulness that does not disturb the stateless intent of WS interface model. The OGF achieved this goal, developing the *Web Service Resource Framework* (WSRF). WSRF allows factory methods in the WS implementation to create objects, which are referenced remotely using the WS-Addressing standard. Persistent resource properties are exposed to coupled services through XML. Introducing state potentially adds enormous complexity to a distributed system, and the distribution of stateful data to multiple service components has the potential for data coherence problems which would require distributed locking mechanisms. The approach introduced by the Grid passes WS endpoints to the resources so that synchronized access is provided by the service implementation.

One path to leveraging Grid technology experiences in the Cloud is to consider building operation support services with a SO. The component services interconnect using the suite of WS standards. The logic of composing the services is built with modern business process design tools which produce a workflow. The design workflow is exported in the form such as the Business Process Execution Language (BPEL) and executed by a workflow engine. This implementation path of using BPEL with WSRF to build a SOA has been demonstrated in an e-Science context (Ezenwoye & Sadjadi, 2010; Ezenwoye, Sadjadi, Carey, & Robinson, 2007).

There is already some experience using WS and WSRF in the Cloud domain. The Nimbus project[12] uses the WS and WSRF model as an interface for clients to access its Cloud workspaces.

8.4.2 Data Management

In Grid computing, data-intensive applications such as the scientific software in domains like high energy physics, bio-informatics, astronomy or earth sciences

[12] http://www.nimbusproject.org/

involve large amounts of data, sometimes in the scale of PetaBytes (PB) and beyond (Moore, Prince, & Ellisman, 1998).

Data management techniques to discover and access information are essential for this kind of applications. Network bandwidth, transfer latency and storage resources are as important as computational resources to determine the tasks' latency and performance. For example, a data-intensive application will preferably be run at a site that has an ample and fast network channel to its dataset so that the network overhead can be reduced, and if it generates a large amount of data, we would also prefer a site that has enough storage space close to it.

Many technologies are applied in Grid computing to address data management problems. Data Grids (Chervenak, Foster, Kesselman, Salisbury, & Tuecke, 2001) have emerged in scientific and commercial settings to specifically optimize data management. For example, one of the services provided by Data Grids is replica management (Chervenak et al., 2002; Lamehamedi, Szymanski, Shentu, & Deelman, 2002; Samar & Stockinger, 2001; Stockinger et al., 2001). In order to retrieve data efficiently and also avoid hot spots in a distributed environment, Data Grids often keep replicas, which are either complete or partial copies of the original datasets. Replica management services are responsible for creating, registering, and managing replicas. Usually, a replica catalog such as Globus Replica Catalog (Allcock et al., 2001) is created and managed to contain information of replicas that can be located by users.

Besides data replication, caching is an effective method to reduce latency at the data consumer side (Karlsson & Mahalingam, 2002). Other technologies such as streaming, pre-staging, high-speed data movement, or optimal selection of data sources and sinks are applied in Data Grids too. These data management technologies are also used in data sharing and distribution systems such as Content Delivery Networks, Peer-to-Peer Networks and Distributed Databases. In Venugopal, Buyya, and Ramamohanarao (2006), the author suggests a taxonomy of Data Grids and compares Data Grids with other related research areas.

Standards for data services have been proposed in the Grid community. The Open Grid Services Architecture (OGSA), which is adopted by the Global Grid Forum (GGF), defines OGSA Data Services (Foster, Tuecke, & Unger, 2008) which include data transfer, data access, storage resource management, data cache, data replication, data federation, and metadata catalogues services. The Database Access and Integration Services Working Group (DAIS-WG) (Antonioletti, Krause, & Paton) at GGF is also developing standards of data services with an emphasis on database management systems, which have a central role in data management such as data storage, access, organization, authorization, etc. There are other groups at GGF that work on data management in Grid computing such as Grid File System Working Group, Grid Storage Management Working Group or GridFTP Working Group. Among them, GridFTP Working Group works on improving the performance of FTP and GripFTP (Allcock, 2003). GridFTP is an extension of FTP and it supports parallel and striped data transfer and partial file transfer. FTP and GridFTP are the most widely-used transport protocols when transferring bulk data for Grid applications.

The Globus Toolkit provides multiple data management solutions including GridFTP, the Global Access to Secondary Storage(GASS), the Reliable File Transfer (RFT), the Replica Location Service (RLS) and a higher-level Data Replication Service (DRS) based on RFT and RLS. Specifically, GASS is a light-weight data access mechanism for remote storage systems. It enables pre-staging and post-staging of files and is integrated into the Globus Resource Access and Monitoring (GRAM) to stage in executables and input data and if necessary, stage out the output data and logs.

In the current state of Cloud computing, storage is usually close to computation and therefore data management is simpler than in Grids, where the pool of execution and storage resources is considerably larger and therefore efficient and scalable methods are required for placement of jobs and data location and transfer. Still, there is the need to take data access into consideration to provide better application performance. An example of this is Hadoop,[13] which schedules computation close to data to reduce transfer delays.

Same as Grid computing, Clouds need to provide scalable and efficient techniques for transferring data. For example, we may need to move virtual machine images, which are used to instantiate execution environments in Clouds, from users to a repository and from the repository to hosting machines. Techniques for improved transfer rates such as GridFTP would result in lower times for sites that have high bandwidth, since they can optimize data transfer by parallelizing the sending streams. Also, catalog services could be leveraged to improve distributed information sharing among multiple participants such that the locating of user data and data repositories is more efficient. The standards developed from Grid computing practice can be leveraged to improve interoperability of multiple Clouds. Finally, better integration of data management with the security infrastructure would enable groups of trusted users. An application of this principle could be used in systems such as Amazon EC2 where VM images are shared by individuals with no assurances about their provenance.

8.4.3 Monitoring

Although some Cloud monitoring tools have already been developed, they provide high level information and, in most cases, the monitoring functionality is embedded in the VM management system following specific mechanisms and models. The current challenge for Cloud monitoring tools is providing information from the Clouds and application/service requests with sufficient level of detail in nearly real time in order to take effective decisions rather than providing a simple and graphical representation of the Cloud status. To do this, different Grid monitoring technologies can be applied to Clouds, specially those of them that are capable to provide monitoring data in aggregate form due to the large scale and dynamic behavior of Clouds.

[13] http://hadoop.apache.org/

The experiences gained with the research of Grid monitoring standardization can drive the definition of unified and standard monitoring interfaces and data models to enhance interoperability among different Clouds.

Grid monitoring is a complex task, since the nature of the Grid means heterogeneous systems and resources. However, monitoring is essential in the Grid to allow resource usage to be accounted for and to let users know whether and how their jobs are running. This is also an important aspect for other tasks such as scheduling.

The OGF Performance Working Group developed a model for Grid monitoring tools called Grid Monitoring Architecture (GMA) (Tierney et al., 2002). The architecture they propose is designed to address the characteristics of Grid platforms. Performance information has a fixed, often short, lifetime of utility. Performance data is often more frequently updated than requested, whereas usual database programs are firstly designed for queries. This means that permanent storage is not always necessary, and that the tools must be able to answer quickly before the data is obsolete. A Grid performance monitoring tool also needs to handle many different types of resources and should be able to adapt when communication links or other resources go down. Thus, monitoring systems should be distributed to suit these requirements. In fact, a monitoring tool should find a good tradeoff between the following characteristics: low latency for delivering data, high data rate, scalability, security policies, and minimum intrusiveness.

The GMA is based on three types of components: producers, consumers and the directory service (see Fig. 8.4). A producer is any component that can send events to a consumer, using the producer interface (accepting subscription, queries and ability to notify). In a monitoring tool, every sensor is encapsulated in a producer; however a producer can be associated to many different sources: sensors, monitoring systems or databases, for example. A consumer is any component that can receive event data from a producer. The consumer interface contains subscription/unsubscription routines and query mechanisms. To exchange data events, producers and consumers have a direct connection, but to initiate the dialog, they need the directory service.

Several monitoring tools have been developed for Grid systems. Balaton et al. (2004) provide a description and categorization of existing performance monitoring and evaluation tools, and Serafeim et al. (Zanikolas & Sakellariou, 2005) propose a taxonomy of Grid monitoring systems, which is employed to classify a wide range of projects and frameworks. Some of these approaches are discussed below.

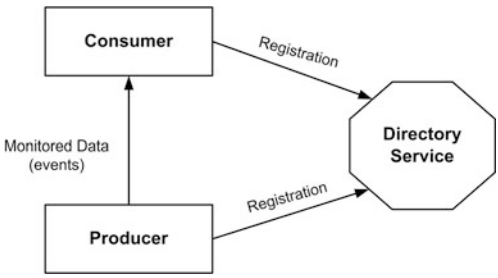

Fig. 8.4 Grid Monitoring Architecture components

Ganglia (Massie, Chun, & Culler, 2004) is a scalable distributed monitoring system for high-performance computing environments such as clusters and Grids. It is based on a hierarchical design targeted at federations of clusters, relies on a multicast-based listen/announce protocol to monitor state within clusters and uses a trace of point-to-point connections amongst representative cluster nodes to federate clusters and aggregate their state. Data is represented in XML and compressed using XDR. The Ganglia Web Frontend can be used to inspect for example CPU utilization in the last hour or last month. Ganglia has been deployed in many HPC infrastructures including supercomputing facilities and large Grid systems.

Network Weather Service (NWS) (Nolski, Spring, & Hayes, 1999) is a distributed system for producing short-term performance forecasts based on historical performance measurements. NWS provides a set of system sensors for periodically monitoring end-to-end TCP/IP performance (bandwidth and latency), available CPU percentage, and available non-paged memory. Based on collected data, NWS dynamically characterizes and forecasts the performance of network and computational resources.

Mercury (Balaton & Gombas, 2003) was designed to satisfy requirements of Grid performance monitoring: it provides monitoring data represented as metric via both pull and push access semantics and also supports steering by controls. It supports monitoring of Grid entities such as resources and applications in a generic, extensible and scalable way. Its design follows the recommendations of the OGF GMA described previously.

OCM-G (Balis et al., 2004) is an OMIS-compliant application monitor developed within the CrossGrid project. It provides configurable online monitoring via a central manager which forwards information requests to the local monitors. However, OCM-G has a distributed architecture.

The Globus Monitoring and Discovery System (MDS) (Czajkowski, Fitzgerald, Foster, & Kesselman, 2001) is another widely used monitoring tool that provides information about the available resources on the Grid and their status. It is based on the GLUE schema,[14] which is used to provide a uniform description of resources and to facilitate interoperation between Grid infrastructures. Other approaches for large-scale systems have been developed such as MonALISA (Newman et al., 2003), which is an extensible monitoring framework for hosts and networks in large-scale distributed systems, and Palantir (Guim, Rodero, Tomas, Corbalan, & Labarta, 2006) that was designed to unify the access to different monitoring and information systems for large scale resource-sharing across different administrative domains, thus providing general ways for accessing all this information. Furthermore, different Grid portal frameworks incorporate monitoring functionalities such as in the HPC-Europa Single Point of Access (Guim et al., 2007) and the P-GRADE Portal (Podhorszki & Kacsuk, 2001).

[14] http://forge.ogf.org/sf/projects/glue-wg.

Several data centers that provide resources to Cloud systems have adopted Ganglia as a monitoring tool. However, virtualized environments have more specific needs that have motivated Cloud computing technology providers to develop their own monitoring system. Some of them are summarized below:

Amazon CloudWatch[15] is a web service that provides monitoring for Amazon Web Services Cloud resources such as Amazon EC2. It collects raw data from Amazon Web Services and then processes the information into readable metrics that are recorded for a period of two weeks. It provides the users with visibility into resource utilization, operational performance, and overall demand patterns - including metrics such as CPU utilization, disk reads and writes, and network traffic.

Windows Azure Diagnostic Monitor[16] collects data in local storage for every diagnostic type that is enabled and can transfer the data it gathers to an Azure Storage account for permanent storage. It can be scheduled to push the collected data to storage at regular intervals or it can be requested an on-demand transfer whenever this information is required.

The **OpenNebula Information Manager** (IM) is in charge of monitoring the different nodes in a Cloud. It comes with various sensors, each one responsible for different aspects of the compute resource to be monitored (CPU, memory, hostname). Also, there are sensors prepared to gather information from different hypervisors.

The monitoring functionality of **Aneka** (Vecchiola, Chu, & Buyya, 2009) is implemented by the core middleware, which provides a wide set of services including also negotiation of the quality of service, admission control, execution management, accounting and billing. To help administrators to tune the overall performance of the Cloud, the Management Studio provides aggregated dynamic statistics.

Nimsoft Monitoring Solution[17] (NMS), built on the Nimsoft Unified Monitoring Architecture, delivers monitoring functionality to any combination of virtualized data center, on hosted or managed infrastructure, in the Cloud on IaaS or PaaS or delivered as SaaS services. Specifically, it provides unified monitoring for data centers, private Clouds and public Clouds such as Amazon WS, including service level and response time monitoring, visualization and reporting.

Hyperic CloudStatus[18] provides open source monitoring and management software for all types of web applications, whether hosted in the Cloud or on premise, including Amazon Web Services and Google App Engine. CloudStatus gives users real-time reports and weekly trends on infrastructure metrics.

[15] http://aws.amazon.com/cloudwatch/

[16] http://www.microsoft.com/windowsazure

[17] http://www.nimsoft.com/solutions/

[18] http://www.hyperic.com/

8.4.4 Autonomic Computing

Inspired by the the autonomic nervous system, autonomic computing aims at designing and building self-managing systems and has emerged as a promising approach for addressing the challenges due to software complexity (Jeffrey & Kephart, 2001). An autonomic system is able to make decisions to respond to changes in operating condition at runtime using high-level policies that are typically provided by an expert. Such a system constantly monitors and optimizes its operation and automatically adapts itself to changing conditions so that it continues to achieve its objectives.

There are several important and valuable milestones to reach fully autonomic computing: first, automated functions will merely collect and aggregate information to support decisions by human users. Later, they will serve as advisors, suggesting possible courses of action for humans to consider.

Self-management is the essence of autonomic computing and has been defined in terms of the following four aspects of self-management (Jeffrey & Kephart, 2001).

- *Self configuration*: Autonomic systems will configure themselves automatically in accordance with high-level policies representing business-level objectives that, for example, specify what is desired and not how it is to be accomplished. When a component is introduced, it will incorporate itself seamlessly, and the rest of the system will adapt to its presence.
- *Self optimization*: Autonomic systems will continually seek ways to improve their operation, identifying and seizing opportunities to make themselves more efficient in performance and/or cost. Autonomic systems will monitor, experiment with, and tune their own parameters and will learn to make appropriate choices about keeping functions or outsourcing them.
- *Self healing*: Autonomic computing systems will detect, diagnose, and repair localized problems resulting from bugs or failures in software and hardware.
- *Self protection*: Autonomic systems will be self-protecting in two senses. They will defend the system as a whole against large-scale, correlated problems arising from malicious attacks or cascading failures that remain uncorrected by self-healing measures. They also will anticipate problems based on early reports from sensors and take steps to avoid or mitigate them.

Figure 8.5 shows one basic structure of an autonomic element as proposed by IBM. It consists of autonomic manager which monitors, analyzes, plans and executes based on collected knowledge, and external environments including human users and managed elements. The managed element could be hardware resources such as CPU, memory and storage, software resources such as a database, a directory service or a system, or an application. The autonomic manager monitors the managed elements and its external environment including changing users requirements, and analyzes them, computes a new plan reflecting changing conditions and executes this plan.

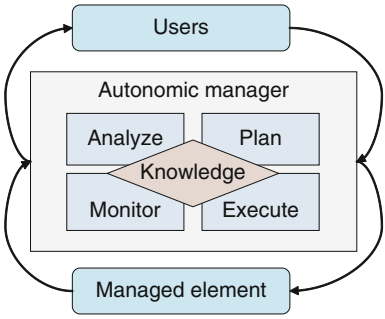

Fig. 8.5 One basic structure of an autonomic element. Elements interact with other elements and external environments through autonomic manager

Autonomic concepts have been effectively applied to distributed computing environments such as Grids (Parashar, Li, & Chandra, 2010), and communication/networking systems (Vasilakos, Parashar, Karnouskos, & Pedrycz, 2010), to monitor resources, changing workloads or models (Quiroz, Gnana-Sambandam, Parashar, & Sharma, 2009), and then adjust resource provisioning to satisfy requirements and constraints (Quiroz, Kim, Parashar, Gnanasambandam, & Sharma, 2009). Such capabilities have been incorporated into Cloud systems. We use CometCloud (Kim, el Khamra, Jha, & Parashar, 2009) as a use case that provides autonomic capabilities at all levels. CometCloud is an autonomic computing engine for Cloud and Grid environments. It is based on decentralized coordination substrate, and supports autonomic applications on highly heterogeneous and dynamic Cloud/Grid infrastructures, as well as integration of public/private Clouds/Grids. For example, it supports autonomic cloudbursts, where the goal is to seamlessly (and securely) bridge private enterprise Clouds and data centers with public utility Clouds or Grids on-demand, to provide an abstraction of resizable computing capacity that is driven by user-defined high-level policies. It enables the dynamic deployment of application components, which typically run on internal organizational compute resources, onto a public Cloud or Grids (i.e., cloudburst) to address dynamic workloads, spikes in demands, economic/budgetary issues, and other extreme requirements. Furthermore, given the increasing application and infrastructure scales, as well as their cooling, operation and management costs, typical over-provisioning strategies are no longer feasible. Autonomic cloudbursts can leverage utility Clouds to provide on-demand scale-out and scale-in capabilities based on a range of metrics.

Other examples of Clouds technologies that are adopting autonomic computing techniques from Grid computing are Aneka (Chu, Nadiminti, Jin, Venugopal, & Buyya, 2007), VioCluster (Ruth, McGachey, & Xu, 2005) and CloudWatch.

8.4.5 Scheduling, Metascheduling, and Resource Provisioning

In the last few decades a lot of effort has been devoted to the research of job scheduling, especially in centers with High Performance Computing (HPC) facilities. The general scheduling problem consists of, given a set of jobs and requirements, a set

of resources, and the system status, deciding which jobs to start executing and in which resources. In the literature there are many job scheduling policies, such as the FCFS approach and its variants (Schwiegelshohn & Yahyapour, 1998a, 1998b; Feitelson & Ruddph, 1995). Other policies use estimated application information (for example the execution time) which make no assumptions such as Smallest Job First (SJF) (Majumdar, Eager, & Bunt, 1988), Largest Job First (LJF) (Zhu & Ahuja, 1993), Smallest Cumulative Demand First (SCDF) (Leutenegger & Vernon, 1990) or Backfilling (Mu'alem & Feitelson, 2001), which is one of the most used in HPC systems.

In Grid computing, scheduling techniques have evolved to incorporate other factors, such as the heterogeneity of resources or geographical distribution. The software component responsible for scheduling tasks in Grids is usually called meta-scheduler or Grid resource broker. The main actions that are performed by a Grid resource broker are: resource discovery and monitoring, resource selection, job execution, handling and monitoring. However, it may be also responsible for other additional tasks such as security mechanisms, accounting, quality of service (QoS) ensuring, advance reservations, negotiation with other scheduling entities, policy enforcement, migration, etc. A taxonomy and survey of Grid brokering systems can be found in (Krauter, Buyya, & Maheswaran, 2002). Some of their most common characteristics are discussed as follows:

- They can involve different scheduling layers through several software components between the Grid resource broker and the resources where the application will run. Thus, the information and control available at the resource broker level is far less than that available at a cluster scheduling level.
- A Grid resource broker usually does not have ownership or control over the resources. Moreover, the cluster scheduling systems may have their own local policies that can conflict with the Grid scheduling strategy.
- There are conflicting performance goals between the users and the resource owners. While the users focus on optimizing the performance of a single application for a specified cost goal, the resource owners aim to obtain the best system throughput or minimize the response time.

While in Grid computing the most important scheduling tasks are optimizing applications response time and resource utilization, in Cloud computing other factors become crucial such as economic considerations and efficient resource provisioning in terms of QoS guarantees, utilization and energy. As virtualized data centers and Clouds provide the abstraction of nearly-unlimited computing resources through the elastic use of consolidated resources pools, the scheduling task shifts to scheduling resources (i.e. provisioning application requests with resources). The provisioning problem in question is how to dynamically allocate resources among VMs with the goal of optimizing a global utility function. Some examples are minimizing resource over-provisioning (waste of resources) and maximizing QoS (in order to prevent falling on under-provisioning that may led to providers revenue loss). Different provisioning techniques for data centers have been proposed such

as those based on gang scheduling (Wiseman & Feitelson, 2003), those based on advance reservations (Sotomayor, Montero, Llorente, & Foster, 2008), those based on energy efficiency (Nathuji & Schwan, 2007; Ranganathan, Leech, Irwin, & Chase, 2006) or those based on multi-tiered resource scheduling approaches (Song, Wang, Li, Feng, & Sun, 2009).

Although Cloud computing scheduling is still challenging, several techniques developed for Grid environments can be taken into account. Existing job scheduling techniques can be also applied in virtualized environments, specially when the application requests rates are those expected in future Clouds. In fact, some approaches have started addressing this issue. Advance reservation developed for Grid scheduling is used in the Haizea lease manager for OpenNebula. Different SLA management policies for Grid computing have been extended for Clouds such as those proposed by Buyya, Yeo, Venugopal, Broberg, & Brasdic, (2009). Market-oriented allocation of resources policies developed for Grids have been realized for Clouds in Aneka (Buyya et al., 2009). Sodan (2009) proposes adaptive scheduling, which can adjust sizes of parallel jobs to consider different load situations and different resource availability through job re-shaping and VM resizing. Moreover, Cloud computing scheduling strategies can leverage Grid multi-layer architectures and strategies such the cross-layer QoS optimization policy proposed by Chunlin and Layuan (2008).

8.4.6 Interoperability in Grids and Clouds

One goal of Grid computing is to provide uniform and consistent access to resources distributed in different data centers and institutions. This is because the majority of Grids are formed based on regional as opposed to local initiatives so interoperation is a key objective. Some examples are TeraGrid in US (Catlett, Beckman, Skow, & Foster, 2006), GridX1 in Canada (Agarwal et al., 2007), Naregi in Japan (Matsuoka et al., 2005) and EGEE in Europe (Berlich, Hardt, Kunze, Atkinson, & Fergusson, 2006). Interoperation is addressed at various architectural points such as the access portal, resource brokering function, and infrastructure standardization.

Some production Grid environments, such as HPC-Europa (Oleksiak et al., 2005), DEISA (Alessandrini & Niederberger, 2004) and PRACE,[19] approach interoperability using a uniform access interface to application users. Software layers beneath the user interface then abstract the complexity of the underlying heterogeneous supercomputing infrastructures.

One tool that takes this approach for Grid interoperation is meta-brokering (Kertesz & Kacsuk, 2008), illustrated in Fig. 8.6. Meta-brokering supports the Grid interoperability from the viewpoint of the resource management and scheduling. Many projects explore this approach with varied emphases. Examples grouped loosely by primary technical foci, are reviewed below.

[19] http://www.prace-project.eu/

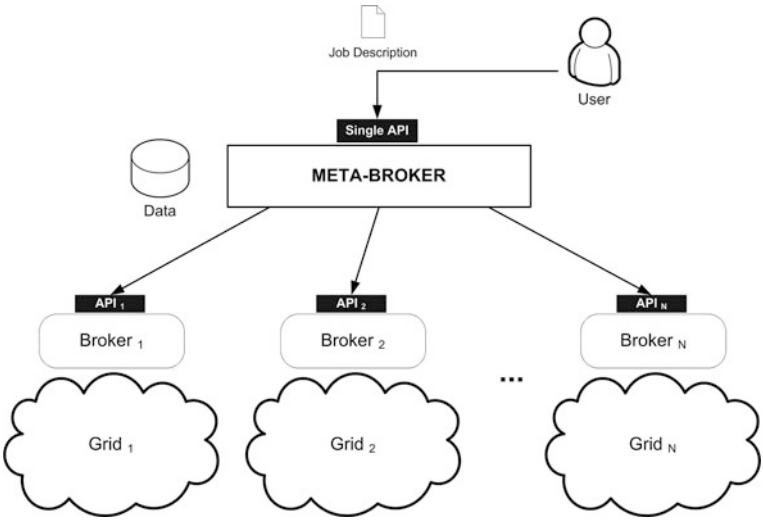

Fig. 8.6 Meta-brokering architecture

- ***I*nfrastructure interoperability**

  ***Grid*Way** (Huedo, Montero, & Llorente, 2004), which is mainly based on Globus, supports multiple Grids using Grid gateways (Huedo, Montero, & Llorente, 2004) to access resources belonging to different domains. **GridWay** forwards local user requests to another domain when the current one is overloaded.

  ***L*atin American Grid Meta-brokering** (Badia et al., 2007; Bobroff et al., 2008), proposed and implemented a common set of protocols to enable interoperability among heterogeneous meta-schedulers organized in a peer-to-peer structure. The resource domain selection is based on an aggregated resource information model (Rodero, Guim, Corbalan, Fong, & Sadjadi, 2010) and jobs from home domain can be routed to peer domains for execution.

- ***R*esource Optimization in interoperated Grids**

  ***K*oala Grid Scheduler** (Mohamed & Epema, 2008) is focused on data and processor co-allocation. To inter-connect different Grid domains as different Koala instances. Their policy is to use resources from a remote domain only if the local one is saturated. They use delegated matchmaking (Iosup, Epema, Tannenbaum, Farelle, & Livny, 2007) to obtain the matched resources from one of the peer Koala instances without routing the jobs to the peer domains.

  ***I*nterGrid** (Assuncao, Buyya, & Venugopal, 2008) promotes interlinking different Grid systems through peering agreements based on economic approaches to enable inter-grid resource sharing. This is an economic-based approach, where business application support is a primal goal, and this also supposed to establish sustainability.

***VIOLA* MetaScheduling Service** (Seidel, Waldrich, Zeigler, Wieder, & Yahyapour, 2007) implements Grid interoperability via SLA mechanisms (WS-Agreement) and provides co-allocation of multiple resources based on reservation.

Other projects explore the interoperability of Grid systems through the use of standard mechanisms, protocols and interfaces. For example, the Grid Interoperability Project (GRIP) (Brooke, Fellows, Garwood, & Goble, 2004), the Open Middleware Infrastructure Institute for Europe (OMII-Europe) project[20] or the work done within the P-GRADE portal (Kacsuk, Kiss, & Sipos, 2008). GRIP was one of the first proposals enabling interoperability between UNICORE and Globus Toolkit. OMII-Europe aimed to influence the adoption and development of open standards that facilitate interoperability between gLite (Lawre et al., 2006) and UNICORE such as OGSA BES (Foster et al., 2008) or JSDL (Anjomshoaa et al., 2005). The P-GRADE portal tries to bridge different Grid infrastructures by providing access to standard-based interoperable middleware. The Grid Interoperation Now Community Group (GIN-CG)[21] and the Production Grid Infrastructure Working Group (PGI-WG)[22] of the OGF also address the problem of Grid interoperability. In the former case, driving and verifying interoperation strategies and, in the latter case, oriented to production Grid infrastructures.

While significant progress on interoperation has been achieved in Grid computing, interoperability among Cloud providers has yet to be explored. While enthusiasm in establishing Cloud interoperability is limited among the for-profit Cloud providers, there are many pursuers in the academic and scientific communities.

The RESERVOIR project[23] addresses Cloud interoperability with a modular, extensible Cloud architecture based on federation of Clouds. In the RESERVOIR model, each infrastructure provider is an autonomous business with its own business goals. A provider federates with other providers based on policies aligned with the site's business goals. In the context of RESERVOIR, Grid interfaces and protocols may enable the required interoperability between the Clouds or infrastructure providers. A similar initiative is the Nuba project[24] whose main aim is the development of a federated IaaS Cloud platform to facilitate the easy and automatic deployment of Internet business services, allowing dynamic scaling based on performance and business goals criteria.

Some Cloud standardization groups have started working on defining common interfaces for interoperation. The Open Grid Forum Open Cloud Computing Interface (OCCI) working group[25] of OGF is working on defining an API

[20] http://www.omii-europe.org
[21] http://forge.gridforum.org/sf/projects/gin
[22] http://forge.ogf.org/sf/projects/pgi-wg
[23] http://www.reservoir-fp7.eu/
[24] http://nuba.morfeo-project.org/
[25] http://forge.ogf.org/sf/projects/occi-wg

specification for remote management of Cloud computing infrastructure, allowing for the development of interoperable tools for common tasks including deployment, autonomic scaling and monitoring. Project OpenNebula and RESERVOIR projects have provided OCCI-compiant implemenations. The Cloud Computing Interoperability Forum (CCIF)[26] is a vendor neutral, not for profit community of technology advocates, and consumers dedicated to driving the rapid adoption of global Cloud computing services.

While encouraging activities in the area of interoperable Clouds are occuring, Grid technologies are more mature. Thus, it is promising to extend these to the Cloud, particularly in the research and evaluation of scheduling and resource selection strategies. While Grid computing focuses on utilization, Cloud computing is more atuned to factors such as QoS, cost, and energy efficiency. Finally, Clouds will want to take advantage of elastic use of their resources in order to optimize both resource usage (and thus, Cloud providers revenue) and QoS given to the users.

8.4.7 Security and User Management

Clouds currently lack many of the mechanisms required for fluid intersite operation of which security is a key enabling factor. Interoperability mandates common security mechanisms that can be translated to the models chosen by local administrators. Additionally, users need to be able to submit requests regardless of the institutions involved in the process of performing the requested task or providing the necessary data. This requires introduction to the Cloud of a mechanism of privilege delegation enabling single sign-on. Finally, collaboration between multiple institutions sharing resources requires development of new methods to manage user privileges.

These challenges have already been addressed in Grid computing where a primary goal is to allow sharing of resources among Virtual Organizations. A VO defines a group of people and resources that can spawn across multiple administrative domains, and allows the definition of fine grained security policies on those resources. The Grid solutions are described in the context of Globus middleware and show how the concepts can be applied to Clouds.

Users in the Grid are granted privileges by site administrators based on their credentials, which are provided by a trusted Certificate Authority. The Grid Security Infrastructure (GSI) (Welch et al., 2003) is the component of the Globus middleware responsible for orchestrating security across different sites. GSI is used by job execution, file transfer, and resource discovery and monitoring protocols to ensure that all operations started by a user are allowed in the target resources.

GSI uses X.509 Public Key Infrastructure (PKI) and SSL/TLS protocols for transport encryption. This allows individuals belonging to organizations to trust foreign credentials issued by a CA without affecting their organization's security measures. However, two additional requirements arise from the dynamic nature of Grid systems.

[26] http://cloudforum.org/

***S*ingle sign-on:** Users in the Grid need to access multiple resources and services with different authentication models. It would be burdensome if each time a user had to perform an action in a remote resource he had to enter a passphrase to use his private key to authenticate himself. Possible solutions such as caching the passphrase could lead to additional security problems.

***P*rivilege delegation:** Due to the dynamic nature of the Grid, users often need to delegate their privileges to other services. This occurs when requests require the orchestration of different resources, or when the user creates a new service to perform a set of capabilities. Following the principle of least privilege, a set of minimum capabilities should be transferred to these services so they can execute.

These requirements are fulfilled by an extension to X.509 certificates called proxy certificates (Welch et al., 2004). Proxy certificates are not issued by a CA, which would be burdensome given their frequency of use and dynamic nature. Instead, the issuer is identified by another public key certificate. This allows temporary certificates to be signed and used for short periods of time without the need to access the user's long time private keys. The relaxed security of proxy certificates suffices as they have a short life cycle.

Proxy certificates are also used to create new certificates with delegated subsets of privileges. The GSI architecture allows different levels of granularity when defining which of the privileges are inherited by the created proxy. Finer levels of granularity can be implemented by using policy languages to express delegation policies. This opportunity is effectively exploited by more advance security services built on the GSI such as the CAS service described below.

An example of the use of proxy certificates would be a computational job that requires access to a storage server to pull datasets to be processed. In this case, a new proxy would be created at the first site by delegating the user's privileges over the network, and in turn, the resource receiving the request would transfer the credentials to the storage server which would perform the operation based on its authorization policies.

One problem with x509 based proxy certificates is the need for users to initiate requests from a machine that has their private keys stored, in addition to the required software to generate a proxy and start a request to the Grid. Often, users access the Grid through web portals, making it difficult to generate their proxy certificates. The MyProxy credential repository was created to solve this issue and permit any user to access Grid resources through a Grid portal using a web browser (Novotny, Tuecke, & Welch, 2001). The MyProxy model adds a repository service where users delegate their credentials and associate them to a user name and password. Subsequently, users can log in to a MyProxy enabled web portal and retrieve and use a previously stored Grid certificate. Certificates delegated to MyProxy repositories have longer lifetimes than usual proxies so users just need to generate them occasionally.

The GSI infrastructure allows resource owners to define access policies in an ad-hoc fashion: usually, site administrators are in charge of defining a mapping from Distinguished Names (DNs) to the local security method. This poses a number of

problems, specially when dealing with large VOs that are distributed across different institutions: the first problem is the burden added to administrators to include access policies for all users, specially if there is a need of defining finer grained ones that vary from one resource to another. Second, systems administrators in charge of assigning access policies don't have a big picture of the project's needs in terms of authorization structure.

The Community Authorization Service (CAS) (Pearlman, Welch, Foster, & Kesselman, 2002) is an extension build on the GSI that provides additional mechanisms to address the deficiencies mentioned above. The CAS abstracts the complexity of access policies for a project into a central server that acts as a repository of policies and users, freeing local resource administrators from the task of identifying authorization requirements. The immediate benefit of this separation of concerns is that project administrators can define users and access rules in the CAS server, and even create groups to define fine grained policies. Once users are added to the CAS server, they contact it when access to a resource is needed, and the CAS server confers them a capability that is equivalent to a proxy certificate. Site administrators need only to validate that the intended operation is allowed for the community the user belongs to and that the operation is allowed by the offered capability. This method scales independently from the number of users and resources. It is directly built on the GSI, which allows its deployment with minimal changes to existing technologies.

In the case of Cloud computing, the lack of standardization among vendors results in multiple security models: for example, both Amazon EC2 and Eucalyptus employ pairs of X.509 certificates and private keys for authentication. Google App Engine, an example of PaaS solution, requires users to first log-in via Google Accounts. The variety of methods makes it difficult to create new opportunities for interoperation, and the fragmentation of security models hinders the reuse of newly developed features.

The OGF Open Cloud Computing Interface Working Group (OCCI) has made a step towards proposing a standardized set of operations, and in its specification it suggests that implementations may require authentication using standard HTTP mechanisms and/or encryption via SSL/TLS. The latest versions of OpenNebula support this specification for communicating with their Cloud controller. This definition represents a possibility to create common grounds for IaaS implementations, providing uniform security paradigms among different vendors.

However, there is still much work in order to achieve a good security infrastructure in Clouds. Methods to specify trust among certificate issuers and resource owners have yet to be implemented, especially for scenarios in which different organizations participate in sharing them. Models such as the GSI infrastructure, where different providers trust various Certifying Authorities without compromising the rest of institutions, would allow the scaling of Clouds outside of single institution boundaries. In those cases, additional techniques to manage users and their associated privileges would be necessary to avoid centralization, and new distributed methods for accounting would be required. Clouds can learn from these solutions in order to define new, standardized interfaces that allow secure, inter-organizational communication.

8.4.8 Modeling and Simulation of Clouds and Grids

Since it is difficult or even not feasible to evaluate different usages on real Grid testbeds, different simulators have been developed in order to study complex scenarios. Simulations allow us to research policies for large and complex configurations with numerous jobs and high demand of resources and to easily include modifications and refinements in the policies. There are many rich simulation models developed by the Grid community.

The GridSim (Sulistio, Cibej, Venugopal, Robic, & Buyya, 2008) simulator has been widely used by many researchers to evaluate Grid scheduling strategies. As described by the GridSim project team it provides a comprehensive facility to create different classes of heterogeneous resources that can be aggregated using resource brokers. GangSim (Dumitrescu & Foster, 2005) allows the simulation of complex workloads and system characteristics. It is also capable of supporting studies for controlled resource sharing based on SLAs. The SimGrid toolkit (Legrand, Marchal, & Casanova, 2003) is a non workload based simulator that allows the evaluation of distributed applications in heterogeneous distributed environments. In these last models, almost all of them model how the jobs are scheduled at the multi-site level (by a given broker or meta-scheduler) but not how the jobs are scheduled and allocated once sent to the final computing resources. In a different approach, the Alvio simulator (Guim, Corbalan, & Labarta, 2007) and Teikoku (Grimme et al., 2007) model all the scheduling layers that are involved in Grid architectures, from meta-brokering policies (see Section 8.4.6) to local job scheduling strategies. DGSim (Iosup, Sonmez, & Epema, 2008) is another relevant simulation framework, which also allows Grid simulation with meta-brokering approaches but, as the former approaches, it does not model local scenarios.

Simulation tools are specially important for Cloud computing research due to the fact that many Clouds are also still in development. CloudSim (Buyya, Ranjan, & Calheiros, 2009) models and simulates Cloud computing environments supporting multiple VMs within a data center node. In fact, VM management is the main novelty of this simulator. It also allows simulation of multiple federated data centers to enable studies of VM migration policies for reliability and automatic scaling of applications. However, several aspects of Cloud computing have not been addressed yet such as the simulation of multiple layers simultaneously. Therefore, the lack of simulators for Cloud computing motivates the extension of existing simulators that were developed for Grid systems and have similar requirements. Some of the existing Grid simulators are described below. While many Cloud simulation models are yet to be developed, leveraging some of the simulation models and experiences would likely accelerate the development for Clouds.

Workloads are crucial to evaluate policies using simulation tools. Although different workload models have been proposed, traces from logs of production systems capture better the behavior of realistic scenarios. There are different publicly available workload traces from production system such as those provided by the Grid

Observatory,[27] which collects, publishes, and analyzes data on the behavior of the EGEE Grid.[28] This is currently one of the most complex public Grid traces with higher frequency of application request arrivals than other large Grids such as Grid5000. However, any of them captures the heterogeneous nature of virtualized Cloud infrastructures with multiple geographically distributed entry points and potential high job arrival rates.

Furthermore, since the traces from different systems are in different formats, using standard formats is very important. Within the Parallel Workload Archive,[29] as well as providing detailed workload logs collected from large scale parallel systems in production use such as San Diego Supercomputer Center or Los Alamos National Lab, Feitelson et al. proposes the Standard Workload Format (SWF) (Chapin et al., 1999) that was defined to ease the use of workload logs and models. Iosup et al. extended this idea for Grids with the Grid Workload Archive (Iosup et al., 2008) and with the Failure Trace Archive (Kondo, Javadi, Iosup, & Epema, 2010) to facilitate the design, validation, and comparison of fault-tolerant models and algorithms.

There is a lack of workload traces and standard models for Clouds. This is an important obstacle to model and simulate realistic Cloud computing scenarios due to Cloud workloads may be composed of different application types, including service requests that have different behavior than the modeled in the current public traces. These existing approaches for parallel systems and Grid systems can be extended to Cloud computing with a definition of a standard Cloud workload format. Workload logs collected from production or research Cloud systems should be also made publicly available to facilitate the research of Cloud computing techniques through simulation.

8.5 Concluding Remarks

Grids and Clouds have many similarities in their architectures, technologies and techniques. Nowadays, it seems Cloud computing is taking more significance as a means to offer an elastic platform to access remote processing resources: this is backed up by the blooming market interest on new platforms, the number of new businesses that use and provide Cloud services and the interest of academia in this new paradigm. However, there are still multiple facets of Cloud computing that need to be addressed, such as vendor lock-in, security concerns, better monitoring systems, etc. We believe that the technologies developed in Grid computing can be leverage to accelerate the maturity of the Cloud, and the new opportunities presented by the latter will in term address some of the shortcomings of the Grid.

As this chapter tries to convey, perhaps the area in which Clouds can gain the most from Grid technologies is in multi-site interoperability. This comes naturally

[27] http://www.grid-observatory.org/

[28] http://www.eu-egee.org/

[29] http://www.cs.huji.ac.il/labs/parallel/workload/

from the fact that the main purpose of Grid systems is to enable remote sites under different administration policies to establish efficient and orchestrated collaboration. This is arguably one of the weakest points in Clouds, which usually are services offered by single organizations that enforce their -often proprietary- protocols, leading for examples to the already identified problem of vendor lock-in. On the other hand, Grid computing, through the use of well defined standards, has achieved site interoperability as it can be seen by the multiple computing and data Grids used by projects in fields as particle physics, earth sciences, genetics and economic sciences.

Another path worth diving into is the one exploring how the new paradigm of Cloud computing can benefit existing technologies and solutions proposed by the Grid community: the realization of utility computing, elastic provisioning of resources, or the homogenization of heterogeneous resources (in terms of hardware, operating systems and software libraries) through virtualization bring a new realm of possible uses for vast, underutilized computing resources. New consolidation techniques allow for studies on lower energy usage for data centers and diminished costs for users of computing resources. There is effectively a new range of applications that can be run on Clouds because of the improved isolation provided by virtualization techniques. Thus, existing software that was difficult to run on Grids due to hard dependencies on libraries and/or operating systems can now be executed on many more resources that have been provisioned complying with the required environment.

Finally, there are some outstanding problems that need to be considered which prevent some users from switching to new Cloud technologies. These problems need to be tackled before we can fully take advantage of all the mentioned opportunities. Other authors, such as (Armbrust et al., 2009), have already listed several of such problems. Some examples are:

1. In certain cases, when processes require intense use of I/O, virtualized environments offer lower performance than native resources. There is a range of scientific applications that have a high communication demand, such as those that rely on synchronous message passing models. Those applications do not offer good performance on Cloud systems.
2. Even though Clouds offer the promise of elasticity of computing resources that would appear to users as endless supply, there are scenarios in the scientific world for which the resources offered by a single Cloud would not be enough. Once the demand for processing power reaches the maximum capacity for a provider, there are no additional means to acquire new resources for the users, if need be. Attempting to use different providers as a back up would mean different protocols, security schemas and new APIs to be employed. For example, the Large Hadron Collider (LHC) project requires processing power not available by any single organization and if deployed to the Cloud, there is a need for interoperability among different Cloud vendors.

We hope that the efforts being taken by numerous researchers in this area identify and address these shortcomings and lead to better and more mature technologies that

will improve the current Cloud computing practices. In these efforts, we believe that a good knowledge of existing technologies, techniques and architectures such as those developed in the field of Grid computing will greatly help accelerating the pace of research and development of the Cloud, and will ensure a better transition to this new computing paradigms.

Acknowledgments We would like to thank Hyunjoo Kim, from the NFS Center for Autonomic Computing at Rutgers University, for her useful insights in Section 8.4.4. This work was partially supported by the National Science Foundation under Grant No. OISE-0730065 and by IBM. Any opinions, findings, and conclusions or recommendations expressed in this chapter are those of the authors and do not necessarily reflect the views of the NSF or IBM.

References

Agarwal, A., Ahmed, M., Berman, A., Caron, B. L., Charbonneau, A., Deatrich, D., et al. (2007). GridX1: A Canadian computational grid. *Future Generation Computer Systems, 23*, 680–687.

Alessandrini, V., & Niederberger, R. (2004). The deisa project: Motivations, strategies, technologies. *19th International Supercomputer Conference, Heidelberg, Germany*.

Allcock, B., Bester, J., Bresnahan, J., Chervenak, A. L., Foster, I., Kesselman, C., et al. (2001). Secure, efficient data transport and replica management for high-performance data-intensive computing. *In Processings of IEEE Mass Storage Conference, IEEE Press, San Diego, CA, USA*.

Allcock, W., (2003). Gridftp protocol specification. Global grid forum recommendation GFD.20 (Tech. Rep., Open Grid Forum (OGF)).

Anjomshoaa, A., Drescher, M., et al. (2005). Job submission description language (JSDL) specification version 1.0, GFD-R.056 (Tech. Rep., Open Grid Forum (OGF)).

Antonioletti, M., Krause, A., & Paton, N. W. (2005). An outline of the global grid forum data access and integration service specifications. *Data Management in Grids LNCS, 3836*, 71–84.

Appleby, K., Fakhouri, S., Fong, L., Goldszmidt, G., Kalandar, M., Krishnakumar, S., et al. (2001). Oceano-sla based management of a computing utility. *Proceeding of the 7th IFIP/IEEE International Symposium on Integrated Network Management (IM 2001), Seattle, WA*.

Armbrust, M., Fox, A., & Griffith, R., et al. (2009). Above the clouds: A berkeley view of cloud computing (CoRR UCB/EECS-2009-28, EECS Department, University of California, Berkeley).

Assuncao, M. D., Buyya, R., & Venugopal, S. (2008). InterGrid: A case for internetworking Islands of grids. *Concurrency and Computation: Practice and Experience, 20*, 997–1024.

Badia, R., et al. (2007). High performance computing and grids in action, Chap. *Innovative Grid Technologies Applied to Bioinformatics and Hurricane Mitigation, IOS Press, Amsterdam*, 436–462.

Badia, R. M., Labarta, J., Sirvent, R., Pérez, J. M., Cela, J. M., & Grima, R. (2003). Programming grid applications with grid superscalar. *Journal of Grid Computing, 1*, 2003.

Balaton, Z., & Gombas, G. (2003). Resource and job monitoring in the grid. *Euro-Par 2003 Parallel Processing, Volume LNCS 2790, Klagenfurt, Austria*, 404–411.

Balis, B., Bubak, M., Funika, W., Wismüller, R., Radecki, M., Szepieniec, T., et al. (2004). Performance evaluation and monitoring of interactive grid applications. *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Volume LNCS 3241 345–352.

Berlich, R., Hardt, M., Kunze, M., Atkinson, M., & Fergusson, D. (2006). Egee: building a pan-european grid training organisation. *ACSW Frontiers '06: Proceedings of the 2006 Australasian Workshops on Grid Computing and e-Research, Darlinghurst, Australia,* 105–111.

Bobroff, N., Fong, L., Liu, Y., Martinez, J., Rodero, I., Sadjadi, S., et al. (2008). Enabling interoperability among meta-schedulers. *IEEE International Symposium on Cluster Computing and the Grid (CCGrid), Lyon, France*, 306–315.

Brooke, J., Fellows, D., Garwood, K., & Goble, C., et al. (2004). Semantic matching of grid resource descriptions. European Acrossgrids Conference, Volume LNCS 3165 Nicosia, Greece, 240–249.

Buyya, R., Ranjan, R., & Calheiros, R. N. (2009). Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. *7th High Performance Computing and Simulation Conference (HPCS 2009), Leipzig, Germany*.

Buyya, R., Yeo, C. S., Srikumar Venugopal, J. B., & Brandic, I. (2009). Cloud computing and emerging it platforms: Vision, hype and reality for delivery computing as the 5th utility. *Future Generation Computer Systems, 25*, 599–616.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems, 25*(6), 599–616.

Catlett, C., Beckman, P., Skow, D., & Foster, I. (2006). Creating and operating national-scale cyberinfrastructure services. *Cyberinfrastructure Technology Watch Quarterly, 2*, 2–10.

Chapin, S. J., Cirne, W., Feitelson, D. G., Jones, J. P., Leutenegger, S. T., Schwiegelshohn, U., et al. (1999). Benchmarks and standards for the evaluation of parallel job schedulers. *Job Scheduling Strategies for Parallel Processing (JSSPP), Volume LNCS 1659, Klagenfurt, Austria*, 66–89.

Chervenak, A., Deelman, E., Foster, I., Guy, L., Hoschek, W., Iamnitchi, A., et al. (2002). Giggle: A framework for constructing scalable replica location services. *Conference on High Performance Networking and Computing, IEEE Computer Society Press, San Jose, CA* 1–17.

Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., & Tuecke, S. (2001). The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications, 23*, 187–200.

Chu, X., Nadiminti, K., Jin, C., Venugopal, S., & Buyya, R. (2007). Aneka: Next-generation enterprise grid platform for e-science and e-business applications. *e-Science and Grid Computing, IEEE International Conference*, 151–159. DOI 10.1109/E-SCIENCE.2007.12.

Chunlin, L., & Layuan, L. (2008). Cross-layer optimization policy for qos scheduling in computational grid. *Journal of Network and Computer Applications, 31*(3), 1084–8055.

Czajkowski, K., Fitzgerald, S., Foster, I., & Kesselman, C. (2001). Grid information services for distributed resource sharing. *IEEE International Symposium on High-Performance Distributed Computing (HPDC), Paris, France* 181.

Dumitrescu, C. L., & Foster, I. (2005). Gangsim: a simulator for grid scheduling studies. *Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05), Washington, DC, USA*, 1151–1158.

Ezenwoye, O., & Sadjadi, S. M. (2010). *Developing effective service oriented architectures: concepts and applications in service level agreements, quality of service and reliability*, chap Applying concept reuse for adaptive service composition. IGI Global (2010).

Ezenwoye, O., Sadjadi, S. M., Carey, A., & Robinson, M. (2007). Grid service composition in BPEL for scientific applications. *In Proceedings of the International Conference on Grid computing, high-performAnce and Distributed Applications (GADA'07), Vilamoura, Algarve, Portugal*, 1304–1312.

Feitelson, D., & Rudolph, L. (1995). Parallel job scheduling: Issues and approaches. *Job Scheduling Strategies for Parallel Processing (JSSPP), Vol. LNCS 949, Santa Barbara, CA, USA*, 1–18.

Foster, I., Kesselman, C., & Tuecke, S. (2001). The anatomy of the grid – enabling scalable virtual organizations. *International Journal of Supercomputer Applications, 15*, 200–222.

Foster, I., Tuecke, S., & Unger, J. (2003). Osga data services. *Global Grid Forum, 9*.

Foster, I., et al. (2008). *OGSA basic execution service version 1.0, GFD-R.108* (Tech. Rep., Open Grid Forum (OGF)).

Foster, I., Zhao, Y., Raicu, I., Lu, S. (2008). Cloud computing and grid computing 360-degree compared. *IEEE Grid Computing Environments Workshop*, 1–10.

Gerndl, M., Wismuller, R., Balaton, Z., Gombas, G., Kacsuk, P., Nemeth, Z., et al. (2004). Performance tools for the grid: State of the art and future. White paper, Shaker Verlag.

Goodale, T., Jha, S., Kaiser, H., Kielmann, T., Kleijer, P., Laszewski, G. V., Lee, C., Merzky, A., Rajic, H., & Shalf, J. (2006). SAGA: A Simple API for Grid Applications. High-level application programming on the Grid. *Computational Methods in Science and Technology*.

Grimme, C., Lepping, J., Papaspyrou, A., Wieder, P., Yahyapour, R., Oleksiak, A., et al. (2007). Towards a standards-based grid scheduling architecture (Tech. Rep. TR-0123, CoreGRID Institute on Resource Management and Scheduling).

Guim, F., Corbalan, J., & Labarta, J. (2007). Modeling the impact of resource sharing in backfilling policies using the alvio simulator. *Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Istambul*.

Guim, F., Rodero, I., Corbalan, J., Labarta, J., Oleksiak, A., Kuczynski, T., et al. (2007). Uniform job monitoring in the hpc-europa project: Data model, api and services. *International Journal of Web and Grid Services, 3*(3), 333–353.

Guim, F., Rodero, I., Tomas, M., Corbalan, J., & Labarta, J. (2006). The palantir grid meta-information system. *7th IEEE/ACM International Conference on Grid Computing (Grid), Washington, DC, USA* 329–330.

Huedo, E., Montero, R., & Llorente, I. (2004). A framework for adaptive execution in grids. *Software—Practice & Experience*, 34, 631–651.

Huedo, E., Montero, R., & Llorente, I. (2009). A recursive architecture for hierarchical grid resource management. *Future Generation Computer Systems, 25*, 401–405.

Iosup, A., Epema, D., Tannenbaum, T., Farrelle, M., & Livny, M. (2007). Inter-operable grids through delegated matchMaking. *International Conference for High Performance Computing, Networking, Storage and Analysis (SC07), Reno, Nevada*.

Iosup, A., Li, H., Jan, M., Anoep, S., Dumitrescu, C., Wolters, L., et al. (2008). The grid workloads archive. *Future Generation Computer Systems, 24*, 672–686.

Iosup, A., Sonmez, O., & Epema, D. (2008) Dgsim: Comparing grid resource management architectures through trace-based simulation. *Euro-Par '08: Proceedings of the 14th international Euro-Par conference on Parallel Processing, Las Palmas de Gran Canaria, Spain*, 13–25.

Kephart, J. O., & Chess, D. M. (2001). *The vision of autonomic computing.* http://www.research.ibm.com/autonomic/manifesto/. Accessed on July 22, 2010.

Kacsuk, P., Kiss, T., & Sipos, G. (2008). Solving the grid interoperability problem by P-GRADE portal at workflow level. *Future Generation Computer Systems, 24*, 744–751.

Karlsson, M., & Mahalingam, M. (2002). Do we need replica placement algorithms in content delivery networks. *Proceedings of the International Workshop on Web Content Caching and Distribution (WCW), Boulder, CA, USA*, 117–128.

Karonis, N. T., Toonen, B., & Foster, I. (2003). Mpich-g2: a grid-enabled implementation of the message passing interface. *Journal of Parallel and Distributed Computing, 63*(5), 551–563. DOI http://dx.doi.org/10.1016/S0743-7315(03)00002-9.

Kertesz, A., & Kacsuk, P. (2008). Grid meta-broker architecture: Towards an interoperable grid resource brokering service. *CoreGRID Workshop on Grid Middleware in Conjunction with Euro-Par, LNCS 4375, Desden, Germany*, 112–116.

Kim, H., el Khamra, Y., Jha, S., & Parashar, M. (2009). An autonomic approach to integrated hpc grid and cloud usage. *e-Science, 2009. e-Science '09. Fifth IEEE International Conference*, 366–373. DOI 10.1109/e-Science.2009.58.

Kondo, D., Javadi, B., Iosup, A., & Epema, D. (2010). The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems. *10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Melbourne, Australia*.

Krauter, K., Buyya, R., & Maheswaran, M. (2002). A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience (SPE), 32*, 135–164.

Lamehamedi, H., Szymanski, B., Shentu, Z., & Deelman, E. (2002). Data replication strategies in grid environments. *Proceedings of the 5th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'02), IEEE Press, Beijing, China*, 378–383.

Laszewski, G. V., Foster, I. T., Gawor, J., & Lane, P. (2001). A Java commodity grid kit. *Concurrency and Computation: Practice and Experience, 13*(8–9), 645–662.

Laure, E., Fisher, S. M., Frohner, A., Grandi, C., Kunszt, P., Krenek, A., et al. (2006). Programming the grid with glite. *Computational Methods in Science and Technology, 12*, 33–45.

Legrand, A., Marchal, L., & Casanova, H. (2003). Scheduling distributed applications: the simgrid simulation framework. 3rd *International Symposium on Cluster Computing and the Grid (CCGrid'03), Washington, DC, USA*, 138.

Leutenegger, S., & Vernon, M. (1990). The performance of multiprogrammed multiprocessor scheduling algorithms. *ACM SIGMETRICS Performance Evaluation Review, 18*, 226–236.

Majumdar, S., Eager, D., & Bunt, R. (1988). Scheduling in multiprogrammed parallel systems. *ACM SIGMETRICS Performance Evaluation Review, 16*, 104–113.

Massie, M. L., Chun, B. N., & Culler, D. E. (2004). The ganglia distributed monitoring system: Design, implementation and experience. *Parallel Computing, 30*(5–6), 817–840.

Matsuoka, S., Shinjo, S., Aoyagi, M., Sekiguchi, S., Usami, H., & Miura, K. (2005). Japanese computational grid research project: Naregi. *Proceedings of the IEEE, 93*(3), 522–533.

Mohamed, H., & Epema, D. (2008). KOALA: a Co-allocating grid scheduler. *Concurrency and Computation: Practice & Experience, 20*, 1851–1876.

Moore, R., Prince, T. A., & Ellisman, M. (1998). Data-intensive computing and digital libraries. *Communications of the ACM, 41*, 56–62.

Mu'alem, A., & Feitelson, D. (2001). Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems, 12, 529–543*.

Nathuji, R., & Schwan, K. (2007). Virtualpower: Coordinated power management in virtualized enterprise systems. *ACM SIGOPS Symposium on Operating Systems Principles, Stevenson, Washington, USA*.

Newman, H., Legrand, I., Galvez, P., Voicu, R., & Cirstoiu, C. (2003). Monalisa: a distributed monitoring service architecture. *Computing in High Energy and Nuclear Physics (CHEP03), La Jolla, CA*.

Novotny, J., Tuecke, S., & Welch, V. (2001). An online credential repository for the grid: Myproxy. *Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE, San Francisco, CA, USA*, 104–111.

Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., et al. (2009). The eucalyptus open-source cloud-computing system. *Cluster Computing and the Grid, IEEE International Symposium, 0,* 124–131. DOI http://doi.ieeecomputersociety.org/10.1109/CCGRID.2009.93.

Oleksiak, A., Tullo, A., Graham, P., Kuczynski, T., Nabrzyski, J., Szejnfeld, D., et al. (2005). HPC-Europa: Towards uniform access to European HPC infrastructures. *IEEE/ACM International Workshop on Grid Computing, Seattle, WA, USA*, 308–311.

Parashar, M., Li, X., & Chandra, S. (2010). *Advanced computational infrastructures for parallel and distributed applications*. New York, NY: Wiley.

Pearlman, L., Welch, V., Foster, I., & Kesselman, C. (2002). A community authorization service for group collaboration. *IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, George Mason University, USA*, 50–59.

Podhorszki, N., & Kacsuk, P. (2001). Semi-on-line monitoring of p-grade applications. *Parallel and Distributed Computing Practices, 4*(4), 43–60.

Quiroz, A., Gnanasambandam, N., Parashar, M., & Sharma, N. (2009). Robust clustering analysis for the management of self-monitoring distributed systems. *Cluster Computing: The Journal of Networks, Software Tools, and Applications*, *12*(1), 73–85.

Quiroz, A., Kim, H., Parashar, M., Gnanasambandam, N., & Sharma, N. (2009). Towards autonomic workload provisioning for enterprise grids and clouds. *10th IEEE/ACM International Conference on Grid Computing (Grid 2009), Banff, Alberta, Canada*, 50–57.

Ranganathan, P., Leech, P., Irwin, D., & Chase, J. (2006). Ensemble-level power management for dense blade servers. *Annual International Symposium on Computer Architecture, New York, NY, USA*, 66–77.

Rodero, I., Guim, F., Corbalan, J., Fong, L., & Sadjadi, S. (2010). Broker selection strategies in interoperable grid systems. *Future Generation Computer Systems, 26*(1), 72–86.

Ruth, P., McGachey, P., & Xu, D. (2005). Viocluster: Virtualization for dynamic computational domains. *Cluster Computing, 2005, IEEE International,* DOI 1–10. 10.1109/CLUSTR.2005.347064.

Samar, A., & Stockinger, H. (2001). Grid data management pilot (gdmp): A tool for wide area replication. *IASTED International Conference on Applied Informatics (AI2001), ACTA Press, Calgary, Canada*.

Schwiegelshohn, U., & Yahyapour, R. (1998). Analysis of first-come-first-serve parallel job scheduling. *9th annual ACM-SIAM Symposium on Discrete Algorithms, Vol. 38, San Francisco, CA*, 629–638.

Schwiegelshohn, U., & Yahyapour, R. (1998). Improving first-come-first-serve job scheduling by gang scheduling. *Job Scheduling Strategies for Parallel Processing (JSSPP), Vol. LNCS 1459, Orlando, FL*, 180–198.

Seidel, J., Waldrich, O., Ziegler, W., Wieder, P., & Yahyapour, R. (2007). Using SLA for resource management and scheduling – a Survey, TR-0096 (Tech. Rep., CoreGRID Institute on Resource Management and Scheduling).

Sodan, A. (2009). Adaptive scheduling for qos virtual machines under different resource availability first experiences. *14th Workshop on Job Scheduling Strategies for Parallel Processing, Vol. LNCS 5798, Rome, Italy*, 259–279.

Song, Y., Wang, H., Li, Y., Feng, B., & Sun, Y. (2009). Multi-tiered on-demand resource scheduling for vm-based data center. *IEEE International Symposium on Cluster Computing and the Grid (CCGrid), Los Alamitos, CA, USA*, 148–155.

Sotomayor, B., Keahey, K., & Foster, I. (2008). Combining batch execution and leasing using virtual machines. *HPDC '08: Proceedings of the 17th International Symposium on High Performance Distributed Computing, ACM, New York, NY, USA*, 87–96. DOI http://doi.acm.org/10.1145/1383422.1383434.

Sotomayor, B., Montero, R., Llorente, I., & Foster, I. (2008). Capacity leasing in cloud systems using the opennebula engine. *Workshop on Cloud Computing and its Applications (CCA08), Chicago, IL, USA*.

Stockinger, H., Samar, A., Allcock, B., Foster, I., Holtman, K., & Tierney, B. (2001). File and object replication in data grids.

Sulistio, A., Cibej, U., Venugopal, S., Robic, B., & Buyya, R. (2008). A toolkit for modelling and simulating data grids: An extension to gridsim. *Concurrency and Computation: Practice and Experience (CCPE), 20*(13), 1591–1609.

Tierney, B., Aydt, R., Gunter, D., Smith, W., Swany, M., Taylor, V., et al. (2002). A grid monitoring architecture.

Vasilakos, A., Parashar, M., Karnouskos, S., & Pedrycz, W. (2010). *Autonomic communication* (XVIII, pp. 374). ISBN: 978-0-387-09752-7. New York, NY: Springer.

Vecchiola, C., Chu, X., Buyya, R. (2009). Aneka: A Software Platform for .NET-based Cloud Computing, Technical Report, GRIDS-TR-2009-4, *Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia*.

Venugopal, S., Buyya, R., & Ramamohanarao, K. (2006). A taxonomy of data grids for distributed data sharing, management and processing. *ACM Computing Surveys (CSUR), 38*(1), 2006.

Welch, V., Foster, I., Kesselman, C., Mulmo, O., Pearlman, L., Gawor, J., et al. (2004). X.509 proxy certificates for dynamic delegation. *Proceedings of the 3rd Annual PKI R&D Workshop, Gaithersburg, MD, USA*.

Welch, V., Siebenlist, F., Foster, I., Bresnahan, J., Czajkowski, K., Gawor, J., et al. (2003). Security for grid services. *High-Performance Distributed Computing, International Symposium, 0*, 48. DOI http://doi.ieeecomputersociety.org/10.1109/HPDC.2003.1210015.

Wiseman, Y., & Feitelson, D. (2003). Paired gang scheduling. *IEEE Transactions on Parallel and Distributed Systems, 14*(6), 581–592.

Wolski, R., Spring, N. T., Hayes, J. (1999). The network weather service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems, 15*(5–6), 757–768.

Zanikolas, S., & Sakellariou, R. (2005). A taxonomy of grid monitoring systems. *Future Generation Computer Systems, 21*(1), 163–188.

Zhao, Y., Hategan, M., Clifford, B., Foster, I., von Laszewski, G., Nefedova, V., et al. (2007). Swift: Fast, reliable, loosely coupled parallel computation. *Services, IEEE Congress on 0*, 199–206. DOI http://doi.ieeecomputersociety.org/10.1109/SERVICES.2007.63.

Zhu, Y., & Ahuja, M. (1993). On job scheduling on a hypercube. *IEEE Transactions on Parallel and Distributed Systems, 4*, 62–69.