ISSN:0975-9646

Amit Batra et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 2 (5) , 2011, 2246 - 2253

Employing Grid Computing Strategies in Cloud Computing

Amit Batra[#], Rajender Kumar[*], Arvind Kumar[#]
[#]*CSE Department,*
H.C.T.M Kaithal India

[*] *H.C.T.M Kaithal India*

***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.**

***Keywords*— Cloud, Grid, *Web Service Resource Framework* (WSRF), Largest Job First (LJF) , Smallest Cumulative Demand First (SCDF)**.

I. Introduction

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 hard-ware. 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. 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.

II Basics of Grid and Cloud Computing

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 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.

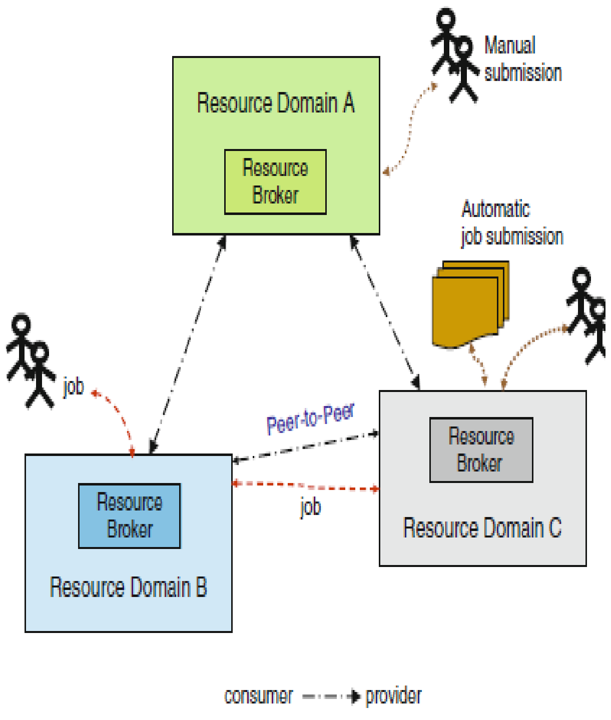

Figure 1. Grid collaborating domains

2. Basics of Cloud Computing

IDC 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 delivery of products, services and solutions over the Internet (i.e., enabling Cloud services)".

In this paper, we will focus the computing infrastructure and platform aspects of the Cloud. Amazon's Elastic Compute Cloud 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 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.

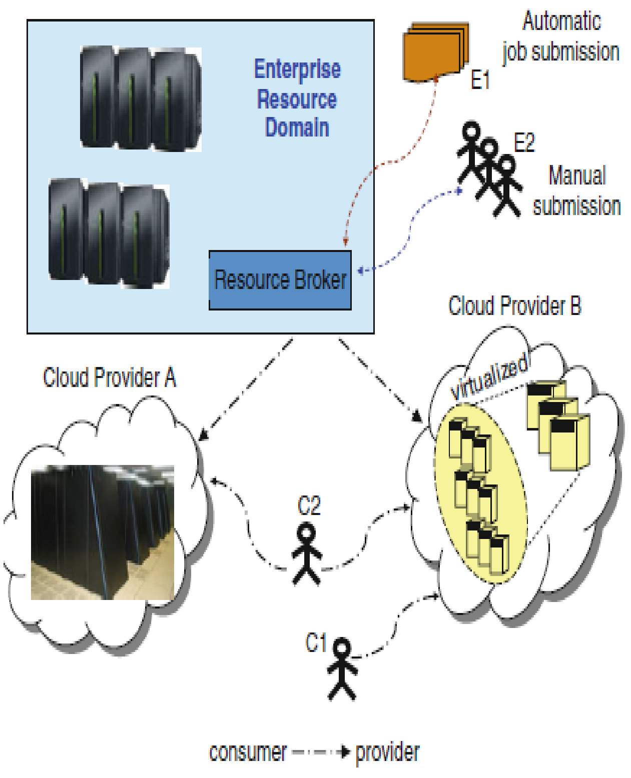

Figure 2 Cloud Usage Models

III. 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 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) 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) is an exemplary proposal in this direction. Modeled after similar formations in the Grid community, OGF officially launched a workgroup, named the Open Cloud Computing Interface Working Group (OCCI-WG) 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.

1. 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.

2. 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. 3.

2.1 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, 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, 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

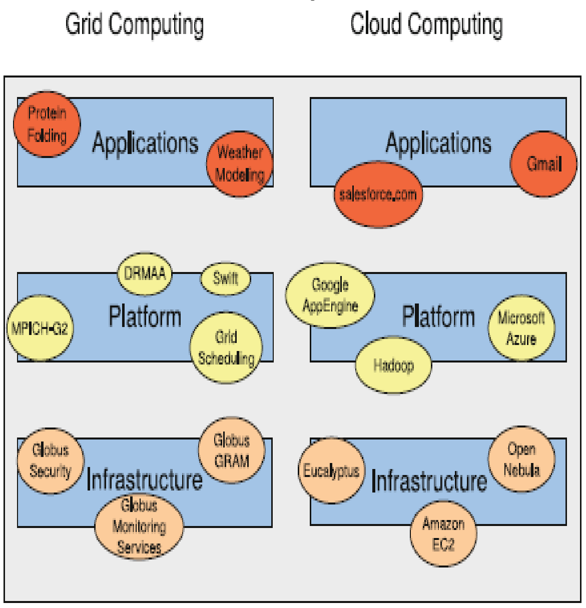

Figure. 3. Grid and Cloud Layers

(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 Cloud Watch.

2*.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, Salesforce's force.com or Microsoft Azure. 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:

2.2.1 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.

2.2.2 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.

IV 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 resourcesand 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.

V 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 , data transfer, resource monitoring or security . 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.

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 uses the WS and WSRF model as an interface for clients to access its Cloud workspaces.

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 involve large amounts of data. 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 lightweight 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, 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.

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 VMmanagement 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. 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 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 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 (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 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.

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 selfhealing measures. They also will anticipate problems based on early reports from sensors and take steps to avoid or mitigate them. Figure 4 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.

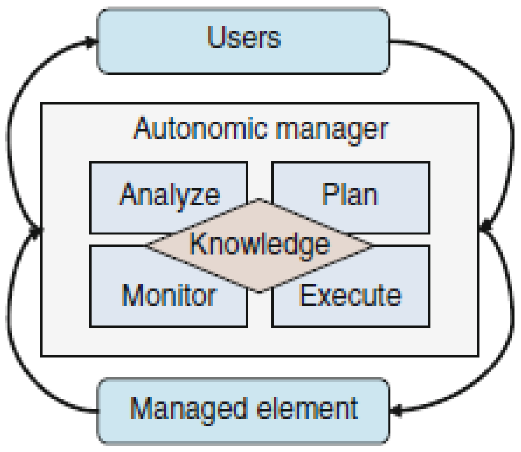

Figure 4. One basic structure of an autonomic element. Elements interact with other elements and external environments through autonomic manager.

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, Feitelson & Ruddph,). Other policies use estimated application information (for example the execution time) which make no assumptions such as Smallest Job First (SJF) (Majumdar, Eager, & Bunt,), Largest Job First (LJF) (Zhu & Ahuja,), Smallest Cumulative Demand First (SCDF) (Leutenegger & Vernon,) or Backfilling (Mu'alem & Feitelson,), 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,). 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).

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,), GridX1 in Canada (Agarwal et al.,), Naregi in Japan (Matsuoka et al.,) and EGEE in Europe (Berlich, Hardt, Kunze, Atkinson, & Fergusson,). 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.,), DEISA (Alessandrini & Niederberger,) and PRACE, 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,), illustrated in Fig. 5.. 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:

• *I*nfrastructure interoperability

***G*ridWay** (Huedo, Montero, & Llorente,), which is mainly based on Globus, supports multiple Grids using Grid gateways (Huedo, Montero, & Llorente,) 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., ; Bobroff et al.,) , 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,) 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,) 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,) 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,) 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.

VI CONCLUSIONS

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 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.

ACKNOWLEDGMENT

We would like to express our gratitude to all those who gave us the possibility to complete this paper. Furthermore we would like to thank International Journal of Computer Science and Information Technology for giving this this esteemed opportunity for publishing this paper. We would also like to thank Director, H.C.T.M, Kaithal, as well as Head of Department(CSE), H.C.T.M., Kaithal for their collaborat-ion and helping us to make resources availbility.

REFERENCES

[1] 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)).
[2] 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.
[3]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.*
[4] 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).
[5] 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.
[6]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.
[7] 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.
[8] Balaton, Z., & Gombas, G. (2003). Resource and job monitoring in the grid. *Euro-Par 2003 Parallel Processing, Volume LNCS 2790, Klagenfurt, Austria*, 404–411.
[9] 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.