Grids vs. Clouds

Michael Brock and Andrzej Goscinski
School of Information Technology
Deakin University
Pigdons Road, Waurn Ponds, Australia
{mrab, ang}@deakin.edu.au

Abstract — **When it comes to grid and cloud computing, there is a lot of debate over their relations to each other. A common feature is that grids and clouds are attempts at utility computing. However, how they realize utility computing is different. The purpose of this paper is to characterize and present a side by side comparison of grid and cloud computing and present what open areas of research exist.**

Keywords – grid computing; cloud computing; grid definition; cloud definition; comparison; utility computing

I. INTRODUCTION

When it comes to comparing grid and cloud computing, there is a lot of debate on how the two are related to each other. We claim here that the two are different and only share features of distributed systems and the idea of utility computing.

Utility computing is the idea coined in 1961 where computational resources (CPU cycle, etc) are accessible in the same manner as the [then] telephone network [40]. Instead of paying the installation and on-going costs of IT infrastructures, utility computing makes it possible to rent them. For decades though, utility computing remained an idea due to limitations in computer hardware and networks and has been recently revived in the early 2000s with the emergence of cloud computing.

As a background of this paper the descriptions of these two technologies are as follows. Grid computing is the ability to process information by utilizing a collection of networked heterogeneous information-processing components (hardware and software), all of which are provisioned from various geographical locations and across organizational boundaries.

Cloud computing is the idea where IT resources are accessed via scalable services in an on-demand manner over the Internet. Cloud computing makes use of servers, physical and virtual, when the need arises. This differs from grids as resources are requested and reserved before they are used. The aim of cloud computing is to apply huge computational power and storage capacity to solve problems such as analyzing investment decisions and risk in financial portfolios, delivering personalized medical information, and powering computer games [21, 33].

For continued research in grid and cloud computing, a clear understanding on their differences is needed. We aim to present a comparison of it against grids and summarize with the identification of possible research areas both areas.

The rest of this paper is structured as follows. Section II provides the most basic elements of distributed computing, specifically middleware and the service oriented architecture. Section III introduces the state of the art in grid and cloud computing and concludes with a side by side comparison. Section IV presents our vision of future cloud computing systems using our innovative RVWS framework. Section V presents possible research areas and elaborates on the future of grid and cloud computing.

II. BASICS OF DISTRIBUTES SYSTEMS AND COMPUTING

The purpose of section is to give a simple summary of distributed computing basics, specifically middleware and SOA. The reason for focusing on middleware and SOA is they are the most commonly used and current approaches to ease the creation of distributed systems.

As the size and complexity of software systems rise, it is no longer viable to run entire software systems on a standalone computer system. The solution to this is to employ distributed systems and distribute computation load across individual systems. The challenge of using distributed systems is ensuring that software systems do not need to know how the distributed system itself is constructed. If this transparency requirement is not satisfied, software system can become bound to the current state of the distributed system.

To best address these needs, the concept of middleware is coupled with the service oriented architecture (SOA) [28]. In general, middleware provides an abstraction of the underlying distributed system and SOA provides an easily reusable model for offering and making use of computing resources.

Middleware is a layer of software that exists between the distributed system and the process that run on it. All the resources are homogenized into a single accessible system image [26]. SOA provides a consistent, flexible, reusable architecture for accessing resources and constructing complex and large distributed systems. In general all resources are accessed via a service that has an exposed and self describing interface. As all communication is performed via the interface, the resource can be accessed in an abstract manner.

III. CURRENT GRID AND CLOUD COMPUTING SYSTEMS

The purpose of this section is to present current state of grid and cloud computing systems and give a side by side comparison. Both are expressions of utility computing, but realized utility computing differently..

A. Grid Computing

The vision of grid computing was to allow access to computer based resources (from CPU cycles to data servers) in the same manner as real world utilities [10, 11, 18]. This gave rise to the idea of Virtual Organizations (VOs). Through the creation of VOs, it was possible to access all resources as though all resources were owned by a single organization.

Two key outcomes exist in grids: the Open Grid Service Architecture (OGSA) [19] and the Globus Toolkit [20]. The OGSA presented an open architecture on how grids are created and maintained. A full explanation of OGSA is beyond this paper thus only the core elements are presented. According to the OGSA all resources within a grid are exposed via grid services with reusable interfaces and presented a general set of services that manage (to name a few) the execution of applications, resource security, monitoring, data access and grid service interoperability across domains.

The Globus Toolkit is a software middleware package that aides in the creation and management of a grid and is the leading implementation of the OGSA. All that is required is to install and configure Globus and then create all required resources and services.

The National Research Grid Initiative (NAREGI) [29] is a grid project that focuses on the research and development of grid middleware. The motivating factor for NAREGI was to allow the creation of a computing infrastructure that allowed international businesses, researchers and academics to share resources. A outcome of the NAREGI project is the creation of a grid testbed to test the viability of NAREGI middleware. The testbed contains almost 3000 CPUs and is capable of 17 teraflops of processing power, offered from various research institutions throughout Japan.

A similar project to NAREGI is the Enabling Grids for E-sciencE (EGEE) Project [27]. While EGEE puts some attention to grid middleware, its primary focus is to aide in the creation of a scientific grid to support large scale research projects such as high energy physics. One outcome of the EGEE project is the gLite middleware which contains components from various grid middleware solutions, Globus being one such contribution. The development of gLite was motivated by most grid middleware solutions being only a proof of concept.

Besides the emergence of gLite, the size of EGEE itself is worth noting. According to statistics [17], the grid offered by EGEE spans 55 countries, supports almost 200 virtual organizations and houses 41 petabytes of data. While the same Web page lists multiple application domains, an examination of the domains support show that the EGEE grid is mostly focused on physics and chemistry applications.

Grid'5000 project [14] focused on the creation of a grid testbed where grid environments could be tested and evaluated with actual and simulated application workloads. This is a significant contribution in the form of concept proofs (which most grids are) are only as good as their last successful tests. Additionally, the Grid'5000 project gets its name from linking 5000 CPUs all over France.

Another research grid project is the Open Science Grid (OSG) [34]: a grid project that sought to create a computing infrastructure where OSG participants can share their computational and storage resources. At the time of writing, the OSG has provided support to very large projects, like Large Haddon Collider (LHC) experiments. In terms of size, it can only be told that the OSG a US based grid: all information provided shows various sites in North America but they are not tallied nor are any sites other countries shown.

B. Cloud Computing

Cloud computing does not have a concrete and commonly accepted definition: it depends on which expert is asked [22, 39]. In this paper, we clouds as large scale, virtualized distributed systems with resources that are accessible over the Internet (currently via Web services). Cloud computing is achieved through a combination of virtualization, SOA, and Web services [12, 16]. This is the first distinction clouds have over grids: grids are currently governed by the OGSA which serves as a reference for grid middleware implementations.

Furthermore, while there are numerous cloud offerings [1, 24, 36, 37], all cloud offerings can be placed in one of three broad categories: *Infrastructure as a Service – IaaS*, *Platform as a Service – PaaS* and *Software as a Service – SaaS*.

IaaS clouds offer (in general) server virtualization and data storage. Clients create virtual servers and then install complete software stacks starting with or from the operating system. As all virtual servers have network access, the services inside the virtual servers are still accessible to any client over the Internet. Requiring clients to build complete software stacks makes IaaS clouds difficult to use but offer the greatest flexibility.

PaaS cloud offer clients complete platforms: (virtual) servers with pre-installed and pre-configured software stacks. The immediate advantage of PaaS clouds they are easier to use and clients can spend more time focusing on the creation and maintenance of their own services. The maintenance of the (virtual) hardware and software is taken care of by the cloud provider. The only drawback is client services need to be compatible with the cloud.

Finally, SaaS clouds are like PaaS clouds with additional virtualization and offer the best ease of use. Providers of SaaS clouds manage the entire (virtual) platform and the applications on top of this. As all maintenance is done by the provider, clients only need to focus on the evaluation of possible software services and use them. As all maintenance is the onus of providers, updates to software are non-negotiable. Thus, if an update makes a software service unusable to a given client, the client has to update to be compliant to the cloud.

To ease reading, the order of clouds presented in this section match the order of the cloud categories. Thus, we start with IaaS clouds and then work towards SaaS clouds.

The first cloud examined is Amazon's Elastic Compute Cloud (EC2) [1]: an IaaS cloud. Clients to EC2 are able to create virtual servers and install all required software into the virtual servers. To provide some ease to clients, EC2 also provides a catalogue of premade virtual servers thus allowing clients to immediately install required services and libraries.

Despite its name, EC2 is not automatically elastic. After creating and starting a virtual server, EC2 does nothing to distribute the client demand to the virtual server. If the client (virtual server owner) wishes for EC2 to automatically distribute the load across multiple virtual servers, the client has to make use of the AutoScale [2] and Elastic Load Balancing [3] services separately.

Google App Engine [24] is a PaaS cloud that provides a complete Web service environment. Thus, clients only have to focus on the installation or creation of their own services which run on Google's servers. The trade off with App Engine restricts how services are developed. At the time of writing, App Engine only supports the Java and Python programming languages. This can complicate acceptance if the client is not familiar with either of the languages.

Another PaaS cloud is Microsoft's Azure [31], which has similar features to that of EC2 and App Engine. Clients to Azure are able to create their own services and then upload the completed services to Azure for use by other clients. If demand for a given service rises, the provider of the service is able to create multiple instances of the service and Azure automatically distributes the load across the instances. Azure also has a built in discovery service. Called the .NET Service Bus [38], when a service is created/started, it publishes itself to the Bus using a URI [30] and then awaits requests from clients.

However, Azure requires its services to be written in .NET thus existing applications either have to be modified or submitted with .NET wrappers to make them compatible with Azure. Furthermore, the underlying operating system platform is a 64bit version of Windows Server. It is unknown at the time of writing if the underlying operating system can be changed.

Another PaaS cloud examined is the Force.com platform [37]. Force.com is the most flexible as it supports over 16 programming languages for Web services alone. Furthermore, Force.com offers a Web site called Appexchange [32] where clients can easily discover required services based on descriptions. Overall, Force.com is a PaaS cloud to developers and a SaaS cloud to clients.

The final cloud examined is Salesforce [36], a SaaS cloud that offers customer relations management (CRM) software as a service. Instead of maintaining hardware and software licenses, users use the software hosted on Salesforce servers for a minimal fee is offered. While the most easily to use, clients cannot create and host them.

C. Grids and Clouds: Side by Side

In general: (i) neither grids nor clouds have a commonly accepted definition; (ii) grids are publicly funded and operated, whereas clouds are privately funded and operated, (iii) grids and clouds are instantiations of distributed systems, which is a common feature of them, (iv) grids evolve slowly and clouds evolve fast, and (v) the level of expertise to use a cloud is significantly lower than that of a grids.

To ease the comparison, this section first examines how grids and clouds were created and the rest of the section is structured around the lifecycle of a computing resource (utility). First, the resource has to be made available. This requires the creation of a service and its publication. Second, the resource (and service) has to be protected: the ownership rights of providers have to be honored. Thus, a security mechanism has to exist to allow the creation of policies enforce client (user) authentication and authorization.

Third, clients have to be able to discover and select required resources and services. If clients cannot find and select resource they need, they cannot make use of the grid or cloud. Finally, the system itself has to be easy to use. It does not matter how effective a grid or cloud is, if the effort needed to use the distributed systems far exceeds the benefits returned, the grid or cloud is a lost to the clients and providers.

Idealization: From an ideas perspective, grids and clouds appear the same: both have resources that are accessed via services, both are accessible over networks, both offer the resources in the form of a utility and both depend on centralized facilities operated by third-party providers. It is not until they are examined closely that their differences become obvious. Both, grids and clouds offer resource sharing. However, grids usually modestly share local resources, clouds make huge systems available. Grids do not have any central management entity, which makes them similar to the Internet, whereas clouds are managed by single entities.

Invention Origins: The genesis of grid and cloud computing are different. The former was coined in academia and a need for easy and wide provision of High Performance Computing (HPC) services to enhance science. However, no viable commercial grid computing provider has emerged. The concept of a service was added to the definition of a grid when the problem of interoperability became the main obstacle.

Cloud computing has been introduced by computing vendors influenced by business requirements and environment to compute enterprise applications and support information processing and data mining. Recently, there is a growing interest in and move of clouds into science.

Grids have been developed based on some standards, such as the OGSA, whereas clouds have been built on open Web service standards and technologies. Despite being promised at the early stages, grids do not offer interoperability. Clouds on the other hand are Web services-based thus provide operability.

The place of invention has also impacted the availability of grid and cloud middleware. As grids were created in academia, there is a strong possibility the middleware is available to other clients (mostly researchers). Clouds were invented in commerce thus their software is proprietary.

This has recently changed though with the invention of Eucalyptus [41] and OpenNebula [42]. Both are open source solutions that allow the creation of clouds and act as research tools to help evolve cloud computing. However, both middleware solutions only support the creation of IaaS clouds. In general, both are open source clones of EC2: OpenNebual is even able to interface with EC2.

Resource/Service Creation: As grid computing is governed by the OGSA, all resources in grid computing are encapsulated and accessed via services with reusable interfaces. For a service that exposes a resource, a strict set of interfaces have to be implemented. Failure to implement the interfaces means the resulting service may not interoperate with other grid services.

By contrast, the creation of resources and services for clouds depends on the category of the cloud being used. For example, if an IaaS cloud is used, the provider of the resource is able to completely control how the service that exposes the resource is developed. If the provider goes from IaaS to PaaS and finally SaaS, the range of choice declines until all that is left is for the provider to choose an existing software service.

In summary cloud computing is more viable as providers can choose a cloud category that supports their development choices. Grids only offer a complete service based model. The development choices are restricted by the grid middleware.

Resource/Service Publication: It does not matter how rich resources are in a grid or how flexible a cloud is. If clients cannot find required resources and services already inside, they will either recreate existing services or opt for another distributed system that does offer discovery. In grids, the publication process depends on the type of grid middleware used to construct the grid.

The leading grid discovery service is the Monitoring and Discovery Service (MDS) found in the Globus Toolkit [23]. Overall, the role of MDS is to allow grid services to register themselves, supply notifications on when they change (such as their current location and state), and allow for the discovery of registered services and resource. Furthermore, MDS is able to be structured in a hierarchical manner to make it saleable.

The only issue with MDS (and OGSA itself) is the lack of a commonly accepted data type for storing information about services. MDS, by implementation, keeps information in XML form while other implementations may opt for SQL databases instead. This can complicate discovery if discovery services have to constantly convert results from their peers.

In clouds, discovery is almost nonexistent. Only Azure and Appexchange were capable of service discovery. Neither EC2 nor App Engine provided a discovery service; hence we can only assume that providers have to source discovery services outside of the clouds, such as UDDI [13]. Our examination of Eucalypus and OpenNebula also shows that discovery is only present to support resource allocation to virtual server. There is no support for services within the virtual servers.

In Azure, clients (service providers) are offered discovery in Azure's underlying .NET Service Bus [38]. After creating a service, the service developer is able to register a URI [30]. When clients attempt to discover a required service, the URI is specified and the client requests are transparently redirected. The only drawback with Azure is it is unclear is information other than the URI can be published.

Finally, Appexchange is the only other discovery service found in the current cloud offerings. While detailed, the information is mostly static and only focuses on elements such as the cost and the functional descriptions of services. The current activity is not publishable.

In summary, grid computing has mature service discovery. What information about services is completely up to their providers and they are able to publish as much or as little as they choose. Furthermore, publication in grid computing supports the idea of service states thus it is possible to publish the current activity of services.

Security: How security is offered in a grid is determined by the construction of the grid itself. While grids have the OGSA, it is primarily focused on how grids services are created, maintained, used and destroyed. How they are secured depending on what grid framework is used.

The most common grid security approach is the Grid Security Infrastructure (GSI) which has been implemented in the Globus Toolkit. While the goal of GSI was to abstract security approaches between organizations, its implementation acts otherwise. The problem with GSI is every client, physical grid node and service has to have a certificate and all certificates have to map down to a known UNIX user account. This conflicts the vision of utility computing as clients who do not have a certificate or have a certificate that does not map down to a user account cannot access the resources.

In comparison, cloud computing is even worse. As there are categories of clouds, there is no one approach to secure all forms of clouds. For example, IaaS clouds (specifically EC2) offer virtual servers. As the services are inside the virtual servers, which do not belong to the cloud provider, the services cannot be easily secured. This is not a technological problem; it is an ethical/legal/liability issue.

The worst situation from the cloud security point of view is in EC2 and App Engine. The user agreement that accompanies EC2 states that Amazon hold no liability to any virtual server compromised [4]. Finally, App Engine services can only be secured using the Google Accounts service [25]. In Azure, security is done through the use of tokens called claims: requests to services contain tokens from a certifying authority that identify and describe the client making the request. To address the scenario where claims are in a format not understood by Azure, Azure is also capable of converting claims so they care be examined and used in Azure services.

Overall, while overly complex and difficult to share resources, grid computing is far more secure than cloud computing. The only cloud with suitable security is Azure. App Engine only supports security provided by Google and EC2 only protects its own infrastructure.

Discovery and Selection: How the discovery and selection is done in grids depends on the grid middleware being used. Earlier in this section, the MDS service in grids was mentioned. Beside the registration of grid services, MDS is also used to allow grid clients to find required services and resources. Having had experience with the MDS service bundled with Globus, we can say it is anything but usable. It is detailed but impossible to select required services as a single service is capable of generating seven or more pages of XML.

NAREGI takes strong initiatives to address the problem. Specifically, a Web frontend to its information services [32] simplifies how grid service information is queried and displayed. Clients are able to learn of resources from job queues to the current state of services inside a NAREGI grid. Result data is returned either as tables or in summarized trees that can be expanded at the client's request. However, there appears to be no string based approach to querying information. Clients are able to specify a category from a list but that is all.

In contrast to cloud computing, most clouds do not offer a discovery service. The only clouds we assessed that had a discovery service were Azure and Appexchange. While there is discovery in Azure, it was found that only a URI of a service can be specified thus is the only element that can be queried. Even though Appexchange keeps static descriptions, clients are still able to specify query strings to find required services.

Like with security, grids are more mature. The only risk is the choice of grid middleware can impact how the information is presented and how it is accessible.

Ease of Use: Ease of use would have to be the most vital issue. It does not matter how viable a distributed system is, if the clients cannot easily use the system, they will reject the system and push for an alternative. Ease of use is hard to gauge has it is a broad term that covers many areas. Rather than going into fine detail, this ease of use is examined in general terms.

In grids, as long GSI is still present, we cannot say that grids will ever be easy to use. Certificates in concept, alone, are not easy to comprehend and are even harder to store and use. In clouds, ease of use depends on the type of cloud used. The easiest to use are SaaS clouds. As clouds make use of Web services, all cloud services are easily accessible and described. It does not matter how a cloud service is developed, all cloud services are described using Web service standards: specifically the Web Service Description Language (WSDL).

IV. Toward "newer" Clouds with the RVWS Framework

As Section III shows, there are significant differences between grid and cloud computing. For example, grids are very restrictive in how services are created while clouds come in three categories. Another example shows that grids are more secure than clouds (which offer little to no security!).

In this section, we present our own innovative Resources Via Web Services (RVWS) framework. The RVWS framework was originally created in response to lack of stateful Web service information being readily available [8, 9] and has since evolved to allow for effective Web service discovery [5] and has even succeeding in simplifying the exposure, discovery, selection and use of a computational cluster [6, 7]. A full explanation of the framework is beyond this paper thus only a summary of its features and accomplishments is given.

The original motivation of the RVWS framework was to make the state of resources behind Web services accessible via WSDL documents. While stateful Web service frameworks such as WSRF [15] existed, the state itself was hidden. Furthermore there was no other Web service discovery standard beside UDDI [13] and it was primitive and did not support the publication of state. The exposure of state was done using data elements called dynamic attributes. As the dynamic attributes changed over time, the state shown in the WSDL document was current to the resources behind the Web service. This was a significant advance the current state of resources could be learned from the Web services directly. Beside the state, it was possible to expose resource characteristics such as the resource being 32bit or 64bit in design. The exposed state and characteristics are applicable to both grids and clouds. In grids, there is no common approach on how grid service state is structured thus there is not even a commonly accepted data type to specify requirements.

Another feature of the RVWS framework is the offering of a Dynamic Broker that supports the publication, discovery and selection of required Web services and resources based on the information exposed by WSDL documents. Like MDS, our Dynamic Broker is distributed and is able to communicate with other Dynamic Brokers in a peer-to-peer manner. This made it possible for clients to learn of required services and resources regardless of where the services and resources were located.

The most recent achievement of the RVWS framework is the publication, discovery, selection and use of a cluster. The outcome itself was a new form of Web service technology called Cluster as a Service (CaaS). Clusters were chosen as the specific test case because clusters are very dynamic and are the most complex resource to offer to clients. Furthermore, clusters are common elements to grid and clouds.

Overall, our RVWS framework has significant advances in addressing the issues raised in Section III.C. It makes it easy to publish resource state and characteristics (publication), the Dynamic Broker is able to provide matching resources and service based on their current stateful WSDL documents (discovery and selection) and we have even created technology that simplified a cluster (ease of use).

The only issue that has not yet been addressed is security. This is a top issue for both clouds and the RVWS framework. At the time of writing, the Dynamic Broker allows *any* client to find *any* resource and service. There is no option for service providers to limit the discoverability of resources and services to clients. As the main use case of the RVWS framework is clouds, we are now looking into security in clouds.

V. Conclusion and Research Directions

When it comes to grid and cloud computing, the two are often seen as the same computing paradigm under different names. In this paper, we sought to separate grids from clouds and provide a side by side comparison in how they are assembled and what services are offered. On comparison, grids are very detailed in how resources are published and discovered and are very secure. However, grids are mostly focused on specialized applications (such as high energy physics) and are not easy to use even when additional middleware is used. Thus, we see that the future of cloud computing would be to take the some ideas of grids and realize them in cloud environments. By improvement, we mean that clouds are given additional services and middleware, we do not mean that grid services are used within cloud to enrich them.

Our RVWS framework is a strong candidate for improving the current state of clouds. Currently, the framework is very effective in how resources are published, discovered, selected and used. Thus we consider that some of the benefits of grids have already made possible in clouds through our framework. Currently, we are now working on security. Once we learn of the best approaches on how to secure resources and services in clouds, we will adapt our experiences into our RVWS framework.

References

1. Amazon (2007) Amazon Elastic Compute Cloud. Accessed 1 August 2009, http://aws.amazon.com/ec2/.

2. Amazon (2009) Auto Scaling. Accessed 28 July 2009, http://aws.amazon.com/autoscaling/.

3. Amazon (2009) Elastic Load Balancing. Accessed 28 July 2009, http://aws.amazon.com/elasticloadbalancing/.

4. Amazon (2009) Security. Updated 30 October 2009, Accessed 20 November 2009, http://aws.amazon.com/agreement/#7.

5. M. Brock and A. Goscinski (2009) Attributed Publication and Selection for Web Service-based Distributed Systems. *Proceedings of the 3rd International Workshop on Service Intelligence and Computing (SIC 2009)*. Los Angeles, CA, USA, 732-739.

6. M. Brock and A. Goscinski (2010) A Technology to Expose a Cluster as a Service in a Cloud. *8th Australasian Symposium on Parallel and Distributed Computing (AusPDC 2010)*. Brisbane, Australia, (to appear)

7. M. Brock and A. Goscinski (2009) Offering Clusters from Clouds using WSDL and Stateful Web Services. *Proceedings of the IEEE Asia-Pacific Services Computing Conference*. Biopolis, Singapore, (in press).

8. M. Brock and A. Goscinski (2008) Publishing Dynamic State Changes of Resources Through State Aware WSDL. *International Conference on Web Services (ICWS) 2008*. Beijing, 449 - 456. 10.1109/ICWS.2008.73.

9. M. Brock and A. Goscinski (2008) State Aware WSDL. *Sixth Australasian Symposium on Grid Computing and e-Research (AusGrid 2008)*. Wollongong, Australia, 82, 35-44, ACM.

10. M. L. Bote-Lorenzo, Y. A. Dimitriadis and E. Gómez-Sánchez (2004) Grid Characteristics and Uses: A Grid Definition. *Grid Computing*. (Ed.). Pgs. 291-298. Springer Berlin / Heidelberg.

11. I. Foster (2003) The Grid: A New Infrastructure for 21st Century Science. *Grid Computing*. G. F. Fran Berman, Tony Hey (Ed.). Pgs. 51-63.

12. D. Booth, et al. (2004) Web Services Architecture. Accessed 18 Feburary 2009, http://www.w3.org/TR/ws-arch/.

13. D. Bryan, et al. (2002) Universal Discovery, Description, Integration. Updated 19 July 2002, Accessed 1 Feburary 2008, http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm.

14. F. Cappello, et al. (2005) Grid'5000: A Large Scale and Highly Reconfigurable Grid Experimental Testbed. *Proc. of the 6th IEEE/ACM Int. Workshop on Grid Computing.* IEEE Computer Society.

15. K. Czajkowski, et al. (2004) The WS-Resource Framework. 5 March 2004. http://www.globus.org/wsrf/specs/ws-wsrf.pdf.

16. F. Curbera, et. al. (2002) Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing,* V. 6, I. 2.

17. EGEE Project (2009) EGEE in Numbers. Updated October 2009, Accessed 24 November 2009, http://project.eu-egee.org/index.php?id=417.

18. I. Foster, C. Kesselman and S. Tuecke (2001) The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International J. Supercomputer Applications*, V. 15, I. 3.

19. I. Foster, et al. (2005) The Open Grid Services Architecture, Version 1.0. January 29, 2005, Informational Document.

20. I. Foster (2005) Globus Toolkit Version 4: Software for Service-Oriented Systems. *FIP International Conference on Network and Parallel Computing.* Springer-Verlag LNCS 3779, 2-13.

21. Gaikai (2009) Accessed 27 November 2009. http://www.gaikai.com/

22. J. Geelan (2009) Twenty-One Experts Define Cloud Computing. Updated 24 January 2009, Accessed 17 July 2009, http://virtualization.sys-con.com/node/612375.

23. Globus (2007) GT Information Services: Monitoring & Discovery System (MDS). Accessed 4 November 2008, http://www.globus.org/toolkit/mds/.

24. Google (2009) App Engine. Accessed 17 February 2009, http://code.google.com/appengine/.

25. Google (2009) The Deployment Description: web.xml. Accessed 20 November 2009, http://code.google.com/appengine/docs/java/config/webxml.html.

26. A. Goscinski, M. Hobbs and J. Silcock (2000) A Cluster Operating System Supporting Parallel Computing. *Cluster Computing,* V. 4, I., pp. 145-156.

27. B. Jones (2005) An Overview of the EGEE Project. *Peer-to-Peer, Grid, and Service-Orientation in Digital Library Architectures.* Pgs. 1-8.

28. C. M. MacKenzie, et. al. (2006) Reference Model for Service Oriented Architecture 1.0. Updated 12 October 2006, Accessed November 2006, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm

29. S. Matsuoka, S. Shinjo, M. Aoyagi, S. Sekiguchi, H. Usami and K. Miura (2005) Japanese computational grid research project: NAREGI. *Proceedings of the IEEE,* V. 93, I. 3, pp. 522-533.

30. M. Mealling and R. Denenberg (2002) Uniform Resource Identifiers (URIs), URLs, and Uniform Resource Names (URNs): Clarifications and Recommendations. Accessed 28 June 2009, http://tools.ietf.org/html/rfc3305.

31. Microsoft (2009) Azure. Accessed 5 May 2009, http://www.microsoft.com/azure/default.mspx.

32. National Institute of Informatics (2009) User Guide: NAREGI Distributed Information Service. October 2008. http://middleware.naregi.org/Download/Docs/UG-NAREGI-IS-e.pdf.

33. OnLive (2009) OnLive: The Future of Video Games. Accessed 27 November 2009, http://www.onlive.com

34. Open Science Grid (2009) Home Page, Access 5 March 2010, http://www.opensciencegrid.org/

35. Salesforce (2009) Appexchange. Accessed 25 July 2009, http://sites.force.com/appexchange/.

36. Salesforce (2009) CRM - salesforce.com. Accessed, http://www.salesforce.com/.

37. Salesforce (2009) Force.com Platform. Accessed 25 July 2009, http://www.salesforce.com/platform/.

38. A. Skonnard (2009) An Introduction to Microsoft .NET Services for Developers. May 2009, White Paper, Pluralsight. http://go.microsoft.com/fwlink/?LinkID=150833.

39. WebWire (2009) Gartner Highlights Five Attributes of Cloud Computing. Updated 23 June 2009, Accessed 17 July 2009, http://www.webwire.com/ViewPressRel.asp?aId=97756.

40. Wikipedia (2009) John MaCarthy (computer Scientist). Updated 22 November 2009, Accessed 23 November 2009, http://en.wikipedia.org/wiki/John_McCarthy_(computer_scientist).

41. D. Nurmi, et al. (2009) The Eucalyptus Open-Source Cloud-Computing System. *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. Pgs 124-131.

42. B. Sotomayor, et. al. (2008) Capacity Leasing in Cloud Systems using the OpenNebula Engine. Cloud Computing and Its Applications (CCA-08). http://www.cca08.org/index.php