Cloud based Scaling of Grid Resources through Grid Middleware

Mary Sumi Kurian
Post Graduate student, Department of Computer Science and Engineering, Karunya University, India

S.P.Jeno Lovesum
Assistant professor (SG), Department of Computer Science and Engineering, Karunya University, India

ABSTRACT
In Grid computing, a common task is performed by combining the resources from different locations or domains. It will use only the fixed number of resources available in different locations. It doesn't scale the resources according to the users' demand. On the other hand cloud computing provides resources to users according to their demand. Grid computing uses the autoscaling capacity of cloud in resource scaling. That means, cloud provides resources to grid and thereby achieving resource autoscaling in grid computing. In this paper, grid resource scaling using cloud is addressed. In this paper the combination of grid and cloud is achieved thorough a middleware called as DIET.

General Terms
Cloud, Integration, Middleware.

Keywords
Cloud computing, Grid computing, IaaS, Scaling, DIET, Virtualization.

1. INTRODUCTION
Using cloud computing anyone can access very large pool of resources in a cost effective way [1]. By using the virtualization concept, cloud computing can support heterogeneous resources and maintains flexibility. Another important advantage of cloud computing is its scalability [2]. All these factors have contributed to making cloud computing popular in the 'computer world'. IaaS cloud is the basic and most popular cloud type. IaaS cloud is the delivery of large computing resources like networks, processors, storage etc. IaaS is mainly accessed by the network and system administrators. Most popular IaaS providers are, Amazon, GoGrid etc.

In order to perform complex computational tasks, different resources are combined that are spread across different geographical locations. This is achieved through grid computing. The main advantage of grid computing is that it can solve large and complex tasks with shorter time. These large tasks have to be scheduled across different computers across the network. Several scheduling and load balancing approaches are already available in the field of grid computing. Grid computing is mainly used for scientific applications.

These individual mechanisms have different advantages. If these two mechanisms are combined, it will yield great benefits. These can be integrated for different purpose. As cloud can provide very large pool of resources, grid can use cloud for the purpose of resource provisioning. Also, grid can achieve greater flexibility through the heterogeneous resources provided by cloud. In these paper, an integrated grid cloud architecture is explained.

The paper is organized as follows: Section 1 provides and introduction and background to cloud and grid computing. Section 2 explains the related works present in the field of grid-cloud integration. Section 3 explains the overall architecture and working of the system. Section 4 contains the details of the experiments conducted and their analysis results. Finally section 6 discusses the conclusion and the future enhancement of the work presented in this paper.

2. RELATED WORKS
Different Grid-Cloud integration is proposed in this section. I.M. Llorente *et al.* [3] proposed a system in which grid directly uses cloud as their resource provider. In this paper cloud is integrated within virtualization. Grid computing is established on top of the virtualization layer. The virtualization layer is the combination of a virtual infrastructure manager and a cloud provider. Through this approach resource management is totally separated from infrastructure management. The architecture uses OpenNebula [4] as the virtual machine manager and Amazon EC2 as the cloud provider. The Function of the virtual machine manager is to deploy, monitor and control VMs. They have also considered virtualization overhead and communication latency during the performance evaluation.

Almost same approach is proposed by Rafael Moreno-Vozmediano *et al.* [5]. Cluster computing services are deployed on top of virtualization infrastructure layer. Along with that it integrates virtualization in the local site to improve the flexibility. The main advantages of this setup are separation of infrastructure management form resource management, partitioning of physical infrastructure from other services and heterogeneous configuration that support a number of services.

Eddy Caron *et al.* [6] use cloud system as a on-demand resource for a grid middleware. They used DIET (Distributed Interactive Engineering Toolbox) [7] as grid middleware and EUCALYPTUS (Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems) [8] as the cloud resource provider. DIET is having a hierarchical structure. The DIET components are Master Agent (MA), Agent, Local Agent (LA) and Server Daemon (SeD). Users submit their task request to MA. MA will forward the request through to corresponding SeD through the hierarchy. SeD will connect with the cloud provider to get the resources. SeD will map the requested services to virtual machines in order to achieve a greater flexibility and scalability.

Claudia [9] is service abstraction layer implemented on top of different cloud environments. It will provide a unique interface to different cloud vendors. Generally, Service Provider (SP) will serve the requests from the user. It will get the resources from the Cloud Provider (CP). Each CP will be having different mechanism to access resources. The burden on SP will increase due to this. Claudia alleviates the problem by providing a

unique interface. This helps the service provider to handle the requests efficiently. In this SGE is deployed on top of Claudia. The main advantages of the system are automatic scaling, smart scaling, appropriate service abstraction and avoidance of cloud vendor lock in problem. In order to handle federated cloud environments another model named InterCloud [10] is developed.

There is no mechanism to control the level of detail in Grid. Shantenu Jha *et al.* [11] use cloud to control the detail in grid. The paper addresses the issue of interoperability between grid and cloud. Different usage modes are introduced in the paper. Cloud provides a wide range of applications and services to grid. High level interfaces help the users to access the cloud resources in a more efficient way. Using cloud affinity, internal components are not visible in the external user environment. It also helps in the management of heterogeneous clouds.

GridWay Metascheduler [12] is a single point access provided for different complex infrastructures. Interoperability between grid and cloud is mainly addressed in this paper. Workload is scheduled across many cloud providers based on the heuristics developed. The heuristics will be based on economic criteria that will depend on both cost and time. Service Manager monitors the metascheduler to find when to scale the resources. The set of heuristics are specified by the Service Manger component.

RightScale [13] works as an intermediary between users and cloud providers by providing unified interfaces. Users can interact with multiple cloud providers on one screen. Well designed user interface and highly customized OS enable users to deploy and manage their cloud applications quickly and conveniently. RightScale allows full customization with abstraction. Grid computing can be integrated with the RightScale so that it can access cloud providers.

Luis Rodero-Merino *et al*. [14] proposed an efficient approach. In this paper, grid uses cloud as the resource provider and an economic mechanism is incorporated with this paper.. Grid middleware DIET is integrated with cloud. TAM (Task Allocation Module) is established between grid and cloud. User request will reach TAM through the grid hierarchy and it will compute different allocation offers according to the capacity available with the cloud provider. All the allocation offers will be sent to the user and the user will choose the best allocation offer that is very much suitable to his requirement. Market based approach is adopted in this paper. By this, each resource is assigned with a price and each user is assigned with a budget. The parameters considered for allocating resources are deadline and cost. Through economic mechanism it also achieves fairness. Fairness means no user will block the execution of other user by holding too many resources. It also incorporated the mechanism of assigning priority to task. The results show that the system yields greater performance while considering the tasks based on risk than the tasks based on their importance.

3. ARCHITECTURE AND WORKING

This experiment is based on the hybrid grid-cloud architecture. That means, a grid middleware is built on the top of a cloud infrastructure. Here the grid middleware used is the DIET. Eddy Caron *et al.* used cloud system as aon-demand resource for a grid middleware. In that paper, DIET architecture is combined with the cloud provider EUCALYPTUS.

3.1 Diet Middleware

DIET is based on the gridRPC message passing mechanism. The basic DIET component is the agent that is responsible for scheduling and data management capabilities. The DIET components are Master Agent (MA), Agent, Local Agent (LA) and Server Daemon (SeD). Each DIET grid has one Master Agent (MA), which is the root of the hierarchy. Users will contact directly with the MA. Each request will be forwarded through the DIET hierarchy until it reaches Server Daemons (SeD) that is responsible for service execution. Each Agent knows the services that can be executed by the corresponding SeDs at the bottom of each one of its children agents. If the SeD cannot execute the particular task, the request will not be forwarded to the corresponding Agent. Each SeD is connected with the DIET hierarchy through the Local Agent (LA).

When each request reaches the SeD, the corresponding SeD will be building a reply according to their state. All the replies will be ordered using some objective function to make the best SeD to come in the first list. At last MA will send all the replies to the user. When the replies reach the user, he will select the best SeD according to some conditions or functions. After that the user will directly contacts the particular SeD for the task execution without the intervention of the DIET hierarchy. Figure 1 shows the DIET hierarchy and message passing. It contains one MA and that will interact with the user while requesting the task as well as by providing the results.

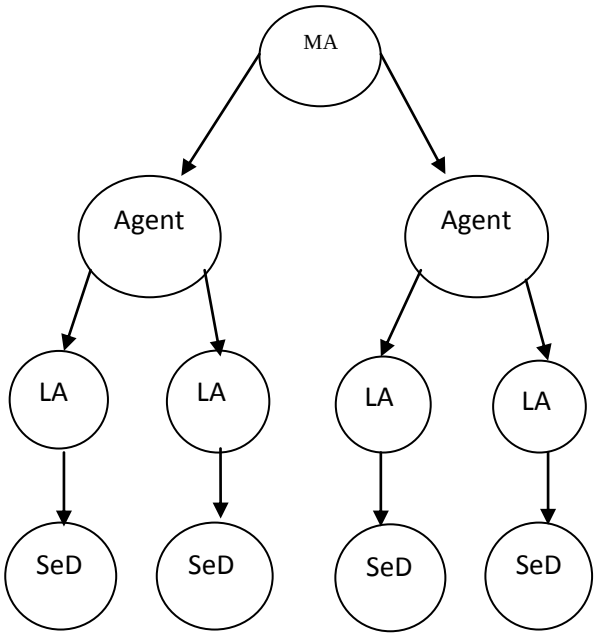

Figure 1. DIET Middelware Architecture

3.2 Integrated Architecture

DIET architecture allows direct connection with IaaS cloud. IaaS cloud will be connected to the SeD nodes, who will decide when to allocate and release resources according to the users' demand. Each service will be run in the VMs hosted in the cloud. Once a VM is created, the SeD node will be connecting to the corresponding VM in order to execute user tasks. The user is totally unaware about the fact that SeDs may run tasks in VMs supplied by IaaS cloud.

Figure 2 shows the integrated grid-cloud architecture. As the DIET architecture will provide a number of offers while

requesting for a task, we built a new module named Offer Computing System (OCS) for that. Using OCS, the DIET middleware can interact with the IaaS cloud providers like Amazon ec2, Eucalyptus etc. Another important function of OCS is that it has to construct all the possible processing offers using the resources available with the cloud providers. After completion of the processing offers computation, it has to be sent back to the SeD. This is done through the OCS.

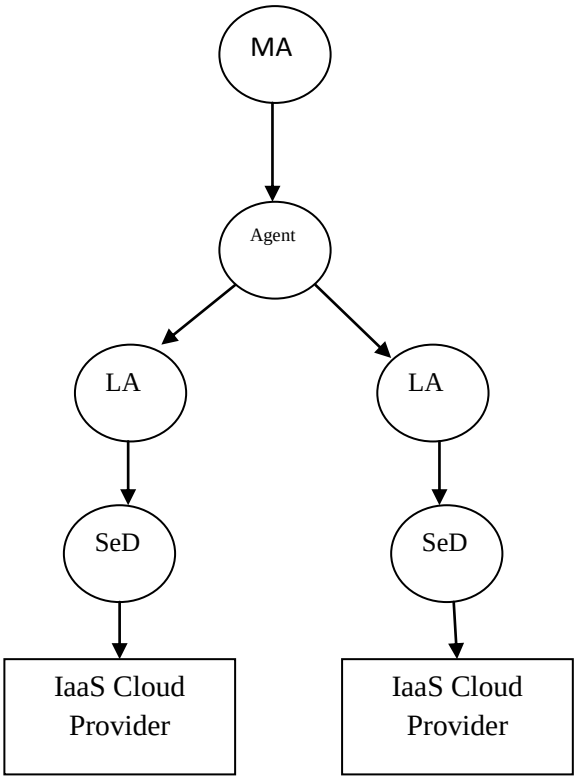

Figure 2. Grid-Cloud Integrated Architecture

3.3 Working

Users will submit the task to the DIET middleware. Users will interact directly with the MA. While receiving a task request from the user, it will travel through the DIET hierarchical architecture and reaches at SeD. When the request reaches SeD, it will be forwarded to the OCS for computing the task offers using the cloud resources.

Before computing the offers, OCS will check some preliminary conditions. That means, it will check whether the received resource request is available with the cloud providers. In our system, it will first check inactive VMs, then active VMs and at last in new VMs. Here the resource considered for the experiment is CPU. The request will contain the amount of CPU needed (expressed in MIPS).

As mentioned earlier, before computing the offers, preconditions have to be satisfied. That means, first OCS has to check whether the requested amount of CPU is available in the active VM, then it will check in the active VM. If these two options are not satisfying, it will go for starting a new VM. In this case, two conditions have to be satisfied. First, it has to check that the VM has the capacity to accomplish the request. Next, it has to check that the new VM can be deployed in a physical server or host. If the conditions are satisfying, the offers have to be calculated. That means, the OCS has to calculate the time required to complete a particular requested task. Let's denote the requested resource using r_i. The processing time taken for the particular task request is calculated using the given formula:

$$P_t = r_i/C_j \quad (1)$$

Where, C_j represents the processing speed of the particular VM. In the case of active VMs, some processes will be running already in that. We have to consider both the currently running task and the task residing in the VM's queue. The processing time in this case is calculated as follows:

$$P_t = (r_i+R_i)/C_j \quad (2)$$

Where, R_i represents the sum of all task execution time and the remaining execution time of the current task. In the last case, we have to find the processing time in a new VM. This is calculated by adding VM starting time with the equation 1.

After completing the processing offers calculation, the offers are sent through the DIET hierarchy upto the MA. When the offer reaches MA, it will be forwarded to the corresponding users. The user will select appropriate offers from the offer list based on their preference. After selecting the appropriate offer, user will contact the corresponding VM. At this stage, the user will contact directly the VM. After the task execution in the corresponding VM, it will send back the result to the corresponding user.

4. EXPERIMENTAL RESULTS

The system is implemented using the popular simulator GridSim [16]. This experiment is conducted with 3 users and 20 tasks. Tasks are randomly assigned to the users. Every task is assigned with a deadline, time needed for the completion of the task. Tasks are failed mainly due to few reasons. One, there is no enough resources available to complete the task. Second one is the processing offers send a processing time that is greater than the deadline of the corresponding tasks.

The experiment is run based on two different algorithms. First, the task requests from users are served based on the First Come First Server (FCFS) basis. Next is based on the Earliest Deadline First (EDF) Basis. Our research results show that the EDF performs better than the FCFS.

4.1 FCFS

In the case of FCFS, the tasks are served in the arrival order. That means, whichever tasks come first will get the first chance. As per this algorithm, the experiment is performed. Out of the 20 tasks, 12 are completed without any problems. 8 tasks are marked as filed. Out of the 8 failed tasks, 6 are failed because the tasks missed their deadline. 2 of the tasks failed due to the lack of resources thereby the OCS couldn't find any processing offers. The analysis is shown in figure 3.

4.2 EDF

Considering, EDF scheduling, the tasks are allocated based on the deadline. If the deadline of the job is near, it will be served first. Second round of experiments are performed absed on this algorithm. Out of the 20 tasks, 15 are completed without any problems. 8 tasks are marked as filed. Out of the 5 failed tasks, 3 are failed because the tasks missed their deadline. 2 of the tasks failed due to the lack of resources thereby the OCS couldn't find any processing offers. EDF algorithm analysis is shown in figure 4.

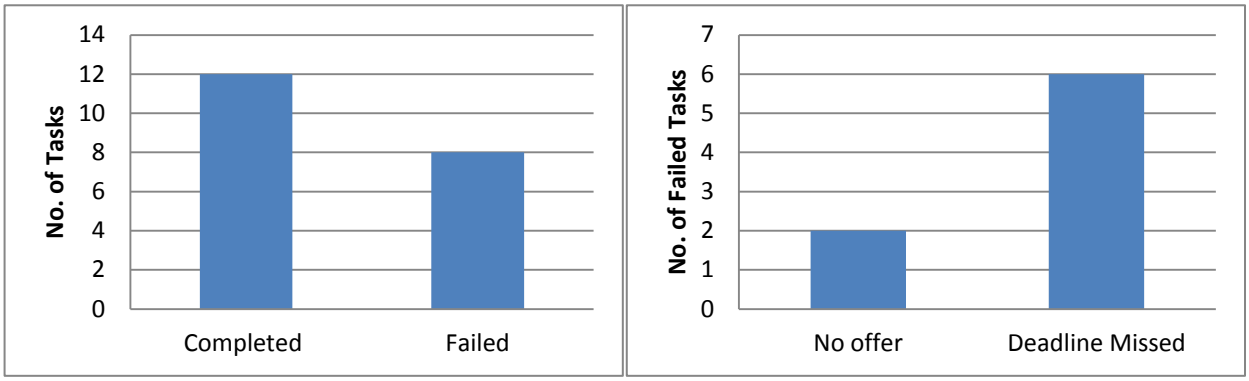

Figure 3. Total and Failed Task Analysis based on FCFS

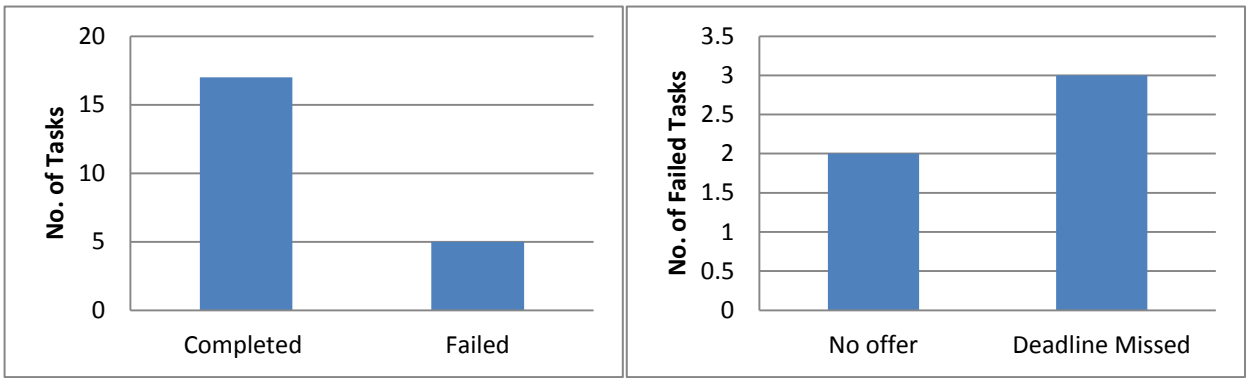

Figure 4. Total and Failed Task Analysis based on EDF

After the execution of both the algorithms, the results are analyzed. Percentages of failed and completed tasks are calculated. Using the FCFS algorithm, the percentage of completed tasks is 60 and the percentage of failed tasks is 40. In the case of EDF algorithm, completed jobs percentage is increased to 75 and the failed tasks percentage is 25. According to the results, majority of the tasks are failed due to the missing their deadlines. There are many situations where a task can be failed due to missing deadline in this architecture. They are,

a. While the task in the request queue itself the deadline has reached.
b. As this is an integrated architecture, we have to consider the overhead of both the environments. DIET is a hierarchical middleware system and the request passing through it will take a longer time. Then both the environments have to be connected with OCS. Finally, it has to interact with the cloud environment. Because of these reasons of time delay, the deadline may be missed. The processing offers computation will take some time as it is checking many preconditions and considering many options like inactive VM, active VM and new VM. During this time the deadline may be reached.
c. Next, the offers have to reach the user through the DIET hierarchy and again the system may fail to reach the tasks deadline during this period.
d. During the processing of offers, it may miss the deadline.
e. Finally, while submitting the tasks to the corresponding VM for the execution, it may fail to meet the deadline.

Because of all these reason, the probability of task failure is high due to missing deadline than other reasons. This system has many advantages. That is, it will give the ability to the user to select any offers according to their preference. It also provides a number of options for task execution. It will collect the processing offers altogether and sending to the user. This will helps the user to
choose the best offer at the same time because the user won't wait for one offer after another.

5. CONCLUSION AND FUTURE WOK

The system presented in this paper combines grid with cloud for resource provisioning. Cloud will provide resources to grid for task execution. By this grid can also achieve autoscaling mechanism. Because of the overhead of the two architectures, the system is having some delay in processing the requests. This is causing some of the tasks to miss their deadlines. Regarding the future work, some mechanisms have to develop to handle the deadline sensitive jobs.

6. ACKNOWLEDGMENTS

The Authors would like to thank all who have provided support and guidelines throughout this work.

7. REFERENCES

[1] Luis M. Vaquero, Luis Rodero-Merino, Juan Cáceres, Maik Lindner, "A break in the clouds: towards a cloud definition", *ACM SIGCOMM Computer Communication Review*, vol.39, issue.1, pp. 50–55, January 2009.

[2] Luis M. Vaquero, Luis Rodero-Merino, Rajkumar Buyya, "Dynamically scaling applications in the cloud", *ACM SIGCOMM Computer Communication Review* vol.41, issue.1, pp. 45–52, January 2011.

[3] I.M. Llorente, R. Moreno-Vozmediano, R.S. Montero, "Cloud computing for on demand grid resource provisioning", *High Speed and Large Scale Scientific*

Computing, in: Advances in Parallel Computing, vol. 18, pp. 177–191, 2009.

[4] J. Fontan, T. Vazquez, L. Gonzalez, R. Montero, and I. Llorente. OpenNEbula: The Open Source Virtual Machine Manager for Cluster Computing. In *Proceedings of the Open Source Grid and Cluster Software Conference*, 2008.

[5] Rafael Moreno-Vozmediano, Rubén S. Montero, Ignacio M. Llorente, "Elastic management of cluster-based services in the cloud", *1st Workshop on Automated Control for Datacenters and Clouds, ICAC'09, ACM*, Barcelona, Spain, pp. 19–24, June 2009.

[6] Eddy Caron, Frédéric Desprez, David Loureiro, Adrian Muresan, "Cloud computing resource management through a grid middleware: a case study with DIET and Eucalyptus", *2nd IEEE International Conference on Cloud Computing, Cloud'09*, pp. 151–154, September 2009.

[7] Eddy Caron, Frédéric Desprez, "DIET: a scalable toolbox to build network enabled servers on the grid, High Performance Computing Applications", *International Journal of High Performance Computing Applications*, vol. 20, issue. 3, pp. 335–352, 2006.

[8] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, Dmitrii Zagorodnov, "The Eucalyptus open-source cloud-computing system", *9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID'09*,pp. 124–131, 2009.

[9] Luis Rodero-Merino, Luis M. Vaquero, Víctor Gil, Javier Fontán, Fermín Galán, Rubén S. Montero, Ignacio M. Llorente, "From infrastructure delivery to service management in clouds", *Future Generation Computer Systems*, vol. 26, issue. 8, pp. 1226–1240, October 2010.

[10] Rajkumar Buyya, Rajiv Ranjan, Rodrigo N. Calheiros, "InterCloud: utilityoriented federation of cloud computing environments for scaling of application services", *10th International Conference on Algorithms and Architectures for Parallel Processing, ICA3PP 2010, in: Lecture Notes in Computer Science*, vol. 6081, pp. 13–31, 2010.

[11] Shantenu Jha, Andre Merzky, Geoffrey Fox, "Using clouds to provide grids with higher levels of abstraction and explicit support for usage modes", *Concurrency and Computation: Practice and Experience*, vol. 21, issue. 8 21, pp.1087–1108., June 2009.

[12] Constantino Vázquez, Eduardo Huedoa, Rubén S. Montero, Ignacio M. Llorente, "On the use of clouds for grid resource provisioning", *Future Generation Computer Systems*, vol. 27, issue. 5, pp. 600–605, May 2011.

[13] Brian Adler, "RightScale Grid: Grid Computing Applications in the Cloud", *A Technical White Paper.*

[14] Luis Rodero-Merino, Eddy Caron, Adrian Muresan, Frédéric Desprez, "Using clouds to scale grid resources: An economic model", *Future Generation Computer Systems*, vol. 28, issue 4, pp. 633-646, April 2012.

[15] Rajkumar Buyya, Manzor Murshed, "GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing", *Concurrency and Computation: Practice and Experience* 14, pp. 1175–1220, 2002.