Agent-based Managing for Grid Cloud System — Design and Prototypal Implementation

Osama H. Younis, Fathy E. Eassa, Fadi F. Fouz, Amin Y. Noaman
Department of Computer Science, Software Engineering and Distributed Systems Research Group, King Abdulaziz University
Jeddah, Saudi Arabia

Ayman I. Madbouly
Department of Research and Consultancy, Deanship of Admission and Registration, King Abdulaziz University
Jeddah, Saudi Arabia

Leon J. Osterweil
Department of Computer Science, University of Massachusetts, USA
Boston, USA

***Abstract*—Here, we present the design and architecture of an Agent-based Manager for Grid Cloud Systems (AMGCS) using software agents to ensure independency and scalability when the number of resources and jobs increase. AMGCS handles IaaS resources (Infrastructure-as-a-Service — compute, storage and physical resources), and schedules compute-intensive jobs for execution over available resources based on QoS criteria, with optimized task-execution and high resource-utilization, through the capabilities of grid clouds. This prototypal design and implementation has been tested and shown a proven ability to increase the reliability and performance of cloud application by distributing its tasks to more than one cloud system, hence increase the reliability of user jobs and complex tasks submitted from regular machines.**

Keywords—Cloud Computing; Grid; Management; Distributed Systems; Architecture

I. Introduction

The growing need for computational resources to solve large-scale problems leads to the cloud computing approach. Before that, Grid computing implemented a paradigm of high-throughput computing with the aim of maximal resource utilization to run multiple jobs or to solve a very big problem by parts [1]. Cloud computing now can fulfill high computations that cannot be done by supercomputers; moreover, its performance can be improved by utilizing all the available resources in a group (grid) of clouds to make sure that most resources are involved according to the required criteria. The proposed Grid Cloud computing infrastructure is considered as an emerging computing paradigm to solve complex applications in science and engineering, as it involves the combined effective utilization of cloud resources to achieve a high-performance computing, and allows the inclusion of a variety of resources like supercomputers, storage systems, and computational kernels. These resources are coupled to be available as a single integrated resource.

This infrastructure can benefit many applications, including distributed supercomputing, high-throughput computing, and data exploration. There is an increasing number of cloud providers varying in the quality of service, and the complex tasks are being increased in the fields of science and engineering. The motivation of this work is to take advantage of these clouds' services and resources to be utilized properly to execute required according to the required QoS criteria. Combining services from multiple providers give new computational capabilities, so for example instead of using costly supercomputers or HPCs, we can group Compute and Storage services (Infrastructure-as-a-Service 'IaaS') together by creating the Grid Cloud. Here, an Agent-based Manager for Grid Cloud System is presented to manage IaaS resources of grid clouds by providing an efficient way of processing high computing requests, based on software agents, for high scalability, robustness, and provider-independency. We have purchased IaaS services from two clouds to be used to execute jobs. Google Compute Engine and Windows Azure Compute have a variety of scalable services to select from, so we have integrated their APIs programmatically with our manager to be able to interact with the clouds and to execute tasks with high scalability according to the QoS. Our manager prototype has been implemented, tested and evaluated with real-world high-computing jobs

II. Background: Grid, Cloud and Agents

Cloud computing uses remote servers and the Internet to maintain applications and data; it allows users to use applications without installation and access their data at any computer through Internet access [2], allowing for efficient computing by centralizing memory, storage, processing, and bandwidth. Cloud computing emerges from Grid computing and provides on-demand resource provisioning. A grid is usually built to utilize the idle resources in an efficient way, but the fact is that if one piece of the software on a node fails, other pieces on other nodes may fail if that component does not have a failover component on another node. Grid and Cloud computing are scalable, network bandwidth and CPU is allocated/de-allocated on demand, storage capacity is increased/decreased depending on the number of instances, users and the amount of transferred data at a given time [3-6].

The autonomous component, Software Agent, has the ability to interact with other agents and its environment on behalf of a user, capable of autonomous actions like figuring out and deciding what needs to be done to satisfy its objectives

Deanship of Scientific Research, King Abdulaziz University, grant No. (24/34 Research Group).

[7]. It is a kind of software abstraction as it provides a powerful and convenient way to describe a complex software entity, defined in terms of its behavior rather than attributes/methods in other programming languages. A multi-agent system consists of a number of interacting agents cooperating, coordinating and negotiating with one another [7]. As known, one of the cloud computing essentials is resource sharing and pooling, so in agent-based cloud computing the coordination and cooperation protocols is adopted to automate the process of resource sharing and pooling in the clouds [8].

III. RELATED WORK

There are a number of management systems for cloud services, some of them can be found as a locally installed management application with a GUI, command-line tools, extensions of a web browser or as online tools. They provide their own management interfaces, designed to specific needs without the ability to interact with other cloud deployments of the same system, particularized to work only with a specific cloud technology and not compatible with others. Some IaaS systems are replicating the same capabilities offered by public providers like Amazon AWS; examples include Nimbus, Eucalyptus, OpenStack and OpenNebula [9, 10].

Others may only suitable for services of one cloud like [11], or for multiple services from multiple providers, like 'Karlsruhe Open Application for cLoud Administration' [12, 13]. There is also an open source, cross-platform, cloud management system called Scalr; provides server management and auto-scaling disaster recovery, where the manager is able to scale a virtual infrastructure according to the load based on RAM, disk, CPU, network or date [14]. Furthermore, there are open source initiatives like deltacloud [15], jcloud [16] and Libcloud [17], but in addition to their limitation to a specific interface, they mainly concerned with the management of public IaaS providers with basic support for private IaaS systems, while they manage virtual instances, they do not concern about the underlying physical infrastructure.

There are other related works that have involved the use of software agents in the management of the clouds, like [18]; a simulated proposed framework to manage resources for service workflows, with a hierarchical architecture for separating decisions of resource management on service, workflow and cloud levels. Another prototypal implementation found for an interface that is compliant with Open Cloud Computing Interface [19] for IaaS resources management resources negotiation, developed as an entryway to a standard FIPA multi-agent system [20]. The number of works that used software agents to manage clouds are limited. Most of them are either for resource negotiation / brokering or a simulated idea for resource allocation without implementation on real clouds [21, 22, 23]. We studied their management functions and how they were designed to understand the architectures in the computational cloud systems at the IaaS level.

The proposed concept *Grid Clouds* is similar to an existing concept called *Cloud Federation*, where services comprised from different clouds are aggregated together. In other words, the physical cloud resources are themselves being considered as a service, and cloud providers are offering their resources for other providers to expand the global cloud coverage offered to their customers without needing physical resources in every geographic locale [4]. Consequently, the cloud becomes a federation of providers/clouds that interoperate together, i.e. exchanging computing resources and data through a defined interface. There are two types of the federation, Horizontal federation and Vertical federation [24]. Horizontal federation expands the capacity of a cloud by integrating a new site and it takes place on one level of the Cloud Stack e.g., infrastructure level. Vertical federation allows the integration of new infrastructures to provide new capabilities by spanning multiple levels [24]. Presently it is almost still a theoretical concept, as there is no common standard for clouds interoperability. As an initiative for developing a common standard, the Open Cloud Computing Interface is trying to standardize an API among different clouds. This enables interoperability between providers, new business models/platforms, specialization of single clouds as well as a broader choice for users [3].

The important point here is that Cloud Federation requires one provider to rent/sell computing resources to another provider, which becomes a permanent or temporary extension of the buyer's cloud environment. Therefore, an agreement must be initiated between providers to make this federation valid. The idea of managing grid cloud services emerges new way of computing technology through grid cloud system. Related work that aggregate clouds are only simulated works; no real clouds involved in their experiments. In contrast, our manager here is using real clouds with no need for an agreement between providers, as the manager is a composite of APIs of these Clouds together to manage resources and tasks.

IV. AGENT-BASED MANAGER FOR GRID CLOUD SYSTEM (AMGCS)

One way of classifying a manager is by its operation scope. A centralized manager schedules and manages all jobs submitted to the grid cloud, whereas a decentralized manager manages jobs submitted to a particular manager in the grid cloud. A centralized manager has a full knowledge and control of the resources and jobs so it can perform good scheduling, but easily become a single point of failure and a performance bottleneck. In contrast, decentralized manager architecture scales well but with low optimal scheduling performance due to the multiplicity of managers.

Scheduling policies classified into two major categories: user-oriented and system-oriented scheduling. The first is trying to optimize the performance for an individual user by minimizing the response time for each job submitted by a user, whereas system-oriented scheduling optimizes the system overall throughput and average response time [26]. A decentralized manager uses a user-oriented policy, whereas a centralized manager performs system–oriented scheduling.

The grid cloud scheduler does not own the physical resources and therefore does not have control over them [27], hence the scheduler must make best effort decision and submit the job to the resources selected. In general, the scheduler function is to map jobs to the suitable resources in the grid cloud. The scheduler involves three phases: Resources Discovery, Resource Selection and Job Execution (Fig. 1). Grid Cloud scheduling maintains a list of available resources and selects the best set of resources depending on users requirement and load balancing strategies. Then the scheduler dispatches jobs to selected virtual machines to be executed and collects the results.

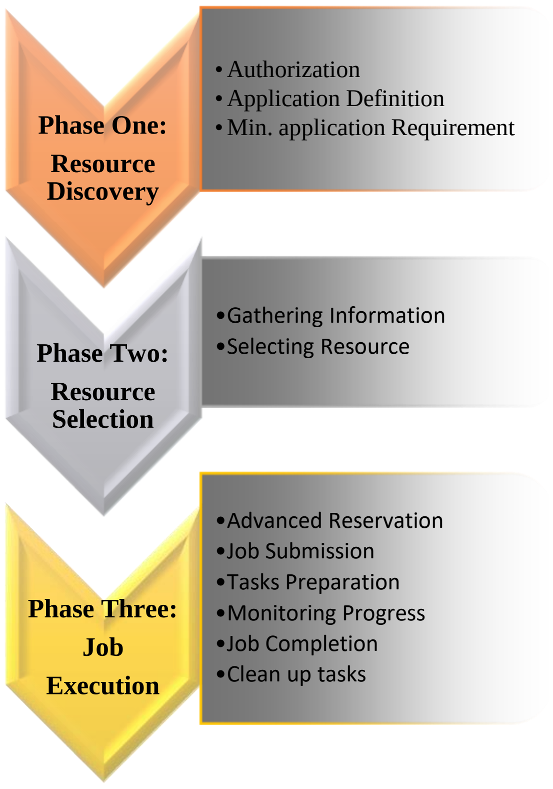


Fig. 1. Scheduling Phases

Grid Cloud resource management focuses on the virtualization and the coordinated use of heterogeneous and distributed resources. The current trend in Cloud systems is the adoption of the software agents for Grid Cloud architecture as the agents' characteristics are compatible with cloud environments [28-30]. This compatibility with Grid Cloud architecture allows us to design a manager for such architectures based on software agents to ensure platform independency and increase manager's scalability and flexibility with high cloud provider independency. Different resources in a grid cloud are varying in operating systems, CPUs, VM images, memory… etc. this difference leads to complex management for these resources. The software agent is well suited to address issues that arise from such a heterogeneous remotely controlled globally shared system. The manager here intend to group multiple clouds together (Fig. 2) and run complex jobs that need high CPU by using the CPUs of the virtual machines in the grid clouds.

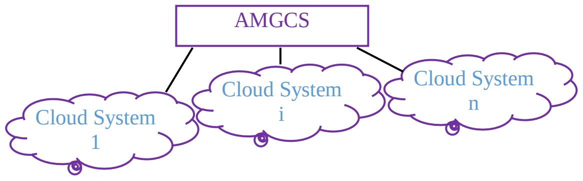


Fig. 2. Managing integrated cloud systems

The Grid cloud concept is built based on the concept of Cloud Federation, as mentioned in section 3, so a grid cloud is a way in which services characterized by interoperability features are aggregated from different clouds in one grid. It addresses the problems of vendor lock-in and provider integration, in addition to increase the performance and disaster-recovery process through techniques like co-location/geographic distribution. It also enables further reduction of costs due to partial outsourcing to more cost-efficient regions. This concept satisfies some security requirements, on un-trusted providers, by using the fragmentation technique to execute part of the job on one cloud and the other part on another cloud, then combining results without allowing each cloud to know the actual job context. Applying this concept in our manager adds benefits like resource redundancy, resource relocation and the combination of complementary services by combining different types to combined services [25]. Here we focus on the horizontal federation as it decreases provider dependency and increases availability (across multiple geographic regions). Therefore, if, for example, the QoS of executing jobs/tasks specifies a low cost, it can be executed on a cloud with the lowest cost or any other QoS requirement. Unlike Cloud Federation, AMGCS does not require an agreement between providers to integrate their services and resources, the manager itself combines the APIs of all grid clouds. The general structure of the designed Grid Cloud Manager is illustrated in Fig. 3, it consists of different agents, each of which has its own task and they are all cooperating together to achieve the manager's roles.

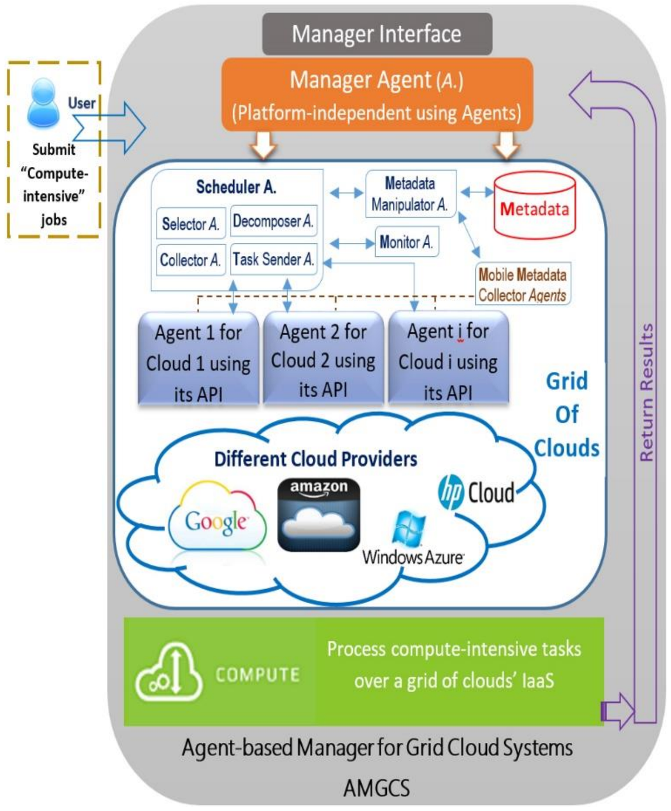

Manager Interface
Manager Agent (A.)
(Platform-independent using Agents)
User
Submit "Compute-intensive" jobs
Scheduler A.
Selector A.
Decomposer A.
Collector A.
Task Sender A.
Metadata Manipulator A.
Monitor A.
Metadata
Mobile Metadata Collector Agents
Agent 1 for Cloud 1 using its API
Agent 2 for Cloud 2 using its API
Agent i for Cloud i using its API
Grid Of Clouds
Return Results
Different Cloud Providers
Google
amazon
Cloud
Windows Azure
COMPUTE
Process compute-intensive tasks over a grid of clouds' IaaS
Agent-based Manager for Grid Cloud Systems
AMGCS

Fig. 3. AMGCS structure

As shown in the figure, the manager's modules are: Scheduler, Monitor, Selector, Decomposer, Collector, Metadata, Mobile metadata collector and Metadata manipulator. All of these modules are agents communicating together with a specific role for each one; the Selector is selecting resources from Metadata to assist the Scheduler in managing resources available in the grid clouds and allocating proper resources to the jobs according to the specified Quality of Service (QoS) and user desires. The monitor is monitoring jobs' executions and resources reserved on the clouds for these jobs. Task sender is the agent that invokes the API calls on the selected cloud, and hence reporting the job's and VM's status to the scheduler, as shown in Fig.4.

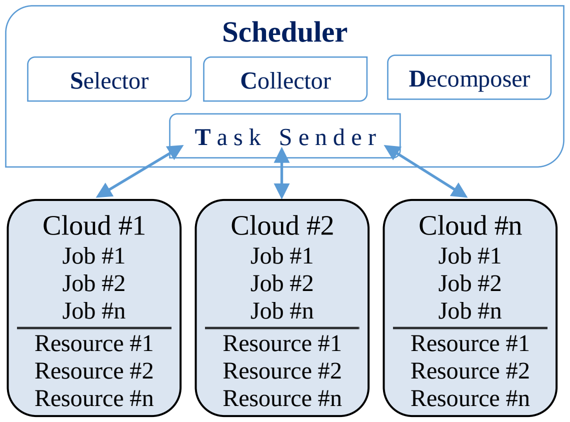


Fig. 4. Deployment of AMGCS Modules

Job decomposition helps in decomposing the job into tasks to assign each task to a proper resource according to the metadata information collected about all resources available in the grid clouds. Decomposition is done based on the job structure/nature; object-oriented system decomposed into packages, web-service based application decomposed to web services and so on. In this prototypal implementation, the decomposition is done programmatically, predefined in the application itself. The Monitor module is tracking the execution of the jobs and the associated resources, Mobile Metadata Collector agents collect and update the Metadata with available resources in the grid clouds.

AMGCS manages the virtual machines in the system and responsible for grouping multiple clouds together in the system; it schedules jobs according to the metadata information then sends the job to the selected cloud to execute it on the associated virtual machine and returns the results, finally terminates VMs after finishing their work. Therefore, the manager consists of a number of services each of which is implemented on an agent and performs a specific function of the manager's tasks, by cooperating with each other to achieve the manager's responsibilities and goals.

In order to achieve a high performance and throughput, the manager is applying best-fit and first-fit mechanisms in its selector algorithm. Best-fit shortlists the best options available for a task, this selection mechanism is slower than First-fit, which selects the first of the best. These two mechanisms are used for the purpose of ranking metadata resources and selecting the proper one that fits the task. Fig. 5 shows the algorithm used by the manager to select and manage grid cloud resources.

```
Input: Job J, User desires U, Different Resources R
Begin
Retrieve Metadata
Run Selector agent
If J is decomposable, Decompose J into tasks JTs
Create shortlist L of R from grid clouds
Sort L according to U, then apply best-fit mechanism
Create list best-fit resource BFR
Loop: select best options from L and insert into BFR
Apply first-fit mechanism on BFR
Initiate connection to the selected virtual machine VM
Task Sender agent sends J or JTs to VM
Notify Monitor agent to monitor JTs, R, VM
Execute J or JTs on selected VM
Monitor agent notifies Collector agent on completion
Collector agent returns results
Terminate connection to VM
End
```

Fig. 5. AMGCS Algorithm

V. Implementation of AMGCS

The manager has been built using Java to implement the agents that compose the manager itself. Each agent is implemented to perform specific tasks, managing resources on different clouds and updating the metadata that contains the information about the available resources in all grid clouds. The manager is integrated with multiple clouds APIs, these clouds are Google Compute Engine, Google Cloud Storage, and Windows Azure Compute. Resources on these clouds are managed/controlled through API functions of each cloud. Authorization and authentication processes must be initiated once before the actual invoking of functions and making requests.

A. Integration with Google Compute Engine (GCE)

The API of GCE has been integrated with our manager to directly call requests to manage available resources. GCE first needs to authenticate the machine before accepting any request; this is done through the OAuth 2.0 protocol. This authorization framework provides clients or third-party applications a method to access resources (HTTP services) either by allowing them to obtain access on its own behalf or on behalf of a resource owner by coordinating an approval interaction between the resource owner and the HTTP service.

Fig. 6 shows the architecture of GCE and the different ways of accessing the cloud, Command Line Interface (CLI), User Interface (UI) and API code library.

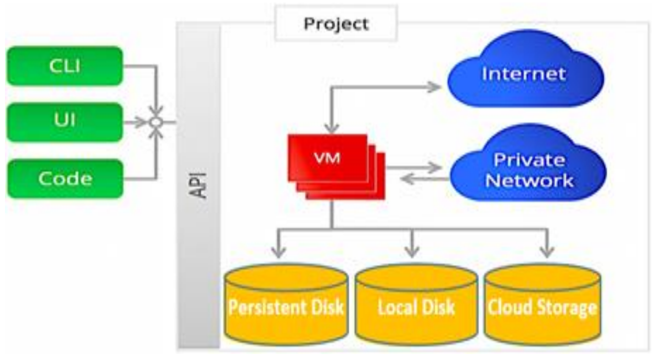


Fig. 6. Google Compute Engine Architecture

To use this API within our manager, a key is required from Google Compute Engine that must be associated with the manager in order to be authorized to perform any requests, this key is called "client_secrets" and can be downloaded in a JSON format from Google Developers Console, example client_secrets.json file:

```
{
 "installed": {
 "client_id": "837647042410-75ifg...usercontent.com",
 "client_secret":"asdlkfjaskd",
 "redirect_uris": ["http://localhost", "urn:ietf:wg:oauth:2.0:oob"],
 "auth_uri": "https://accounts.google.com/o/oauth2/auth",
 "token_uri": "https://accounts.google.com/o/oauth2/token"
  }
}
```

The manager will use OAuth 2.0 in order to authenticate the RESTful API. This API will be used to create and delete disks, virtual machine instances, and other resources, and to integrate with other Google Cloud services, i.e. Google Cloud Storage which is used here to store the metadata for the manager itself. Fig. 7 shows the flow of authenticating APIs calls.

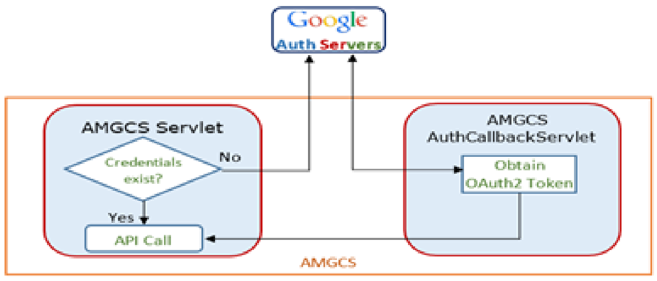


Fig. 7. Authenticated API calls sample flow

Servlet is a Java class to extend the server capabilities to respond to any requests types and to extend the applications hosted by web servers. Several machine types are available from GCE; micro, standard, high CPU and high memory machine types, shown in Table 1. AMGCS select high CPU types for tasks require more virtual cores relative to memory. GCE uses GCEU (Google Compute Engine Unit) as a unit of CPU capacity describing compute power. Minimum power of one logical core on the Sandy Bridge platform is 2.75 GCEUs.

TABLE I. SELECTED LIST OF MACHINE TYPES ON GCE

Configuration	Virtual Cores	Memory
Micro - Small	Shared core	0.60 – 1.7 GB
Standard	1	3.75 GB
	2	7.50 – 13 GB
High Memory,	4	15 – 26 GB
High CPU	8	30 – 52 GB
	16	60 – 104 GB

These machine types, in addition to many others, are included in our metadata database, so the scheduler will choose the proper one to execute the job, according to the required QoS and whether cost or response time is the most critical factor for user desires. After selecting the machine type, API request will be sent, after being authenticated, to initiate a new instance with the specified properties/configurations. Afterward, the manager calls an API function to start running the specified instance (VM) on that Infrastructure using the instances().insert function. These instances can run Linux server from many available images; provided by Google or customized images of other systems, as needed. Finally, jobs will be executed on this instance and others on other instances, results returned to the manager then to the user. The integration of GCE API with our manager is illustrated in Fig. 8.

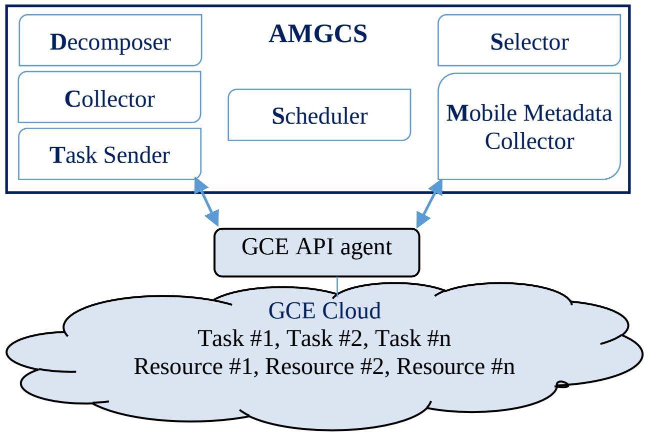

AMGCS
Decomposer
Selector
Collector
Scheduler
Mobile Metadata Collector
Task Sender
GCE API agent
GCE Cloud
Task #1, Task #2, Task #n
Resource #1, Resource #2, Resource #n

Fig. 8. GCE integration with AMGCS

B. *Integration with Windows Azure Compute (WAC)*

Windows Azure is supporting Microsoft operating systems and non-Microsoft operating systems. Its VM image gallery includes latest releases of Windows Server, SharePoint, SQL Server, BizTalk Server, and many non-Microsoft workloads like Ubuntu, SUSE Linux, openSUSE, OpenLogic, etc. Integrating WAC with AMGCS gives the power of handling yet more requests for high computing by calling the associated API call to create new instances, with built-in capability of Load Balancing, monitoring and restarting VMs.

Windows Azure Compute has many machine types ranging from extra small to extra-large machines, also AMGCS will select from this wide range of machine types the proper machine type with proper configurations that suit the job requirements, Table 2 shows some of these machine types, and Fig.9 shows the integration of GCE API with manager's agents.

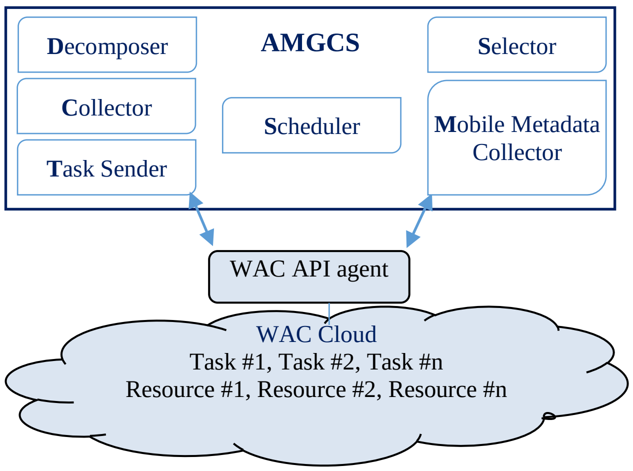

Decomposer
AMGCS
Selector
Collector
Scheduler
Mobile Metadata Collector
Task Sender
WAC API agent
WAC Cloud
Task #1, Task #2, Task #n
Resource #1, Resource #2, Resource #n

Fig. 9. Windows Azure Compute integration with AMGCS

TABLE II. SELECTED LIST OF MACHINE TYPES ON AZURE COMPUTE

Configuration	Virtual Cores	Memory
Extra small	Shared core	768 MB
Small	1	1.75 GB
Medium	2	3.5 – 14 GB
Large	4	7 – 28 GB
Extra large	8	14 – 56 GB

In order to use WAC API, the following is required:

1) A subscription Id: which uniquely identifies our subscription; this id is obtained from the Windows Azure portal.

2) A management Certificate: that must be associated with our subscription to authenticate the API calls.

As the endpoints are accessible over HTTP, we have to create, in our code, an endpoint specific to a particular kind of operation we want to perform, then create HTTP request for that endpoint and the management certificate is attached with that request to be authenticated.

C. AMGCS Metadata

The manager schedules tasks according to this updated metadata information. A cloud database is used here to store the manager's metadata; this cloud database is Google Cloud Storage, to guarantee the compatibility and independency of any platform. Provided by Google, with features like object versioning, parallel uploads and CRC-based integrity checking to maintain the robustness of our sophisticated manager. We can access its API using XML, JSON or using the libraries for several popular programming languages including Java. This storage service is used here to guarantee platform independency and the proper integration with mobile metadata collector agents. The metadata includes the following info of each cloud system in the grid clouds:

Name: Name of the resource or virtual machine.

Description: Description about the resource.

ID: The unique ID of the resource.

CPUs: Number of CPUs in the virtual machine

ImageSpace: The size of the server image in Gigabyte.

Kind: The category of the virtual machine, e.g. high memory, high CPU, or standard.

Disks: The maximum number of disks can be associated to a specific virtual machine.

DisksSize: The size of the disks associated to a specific virtual machine.

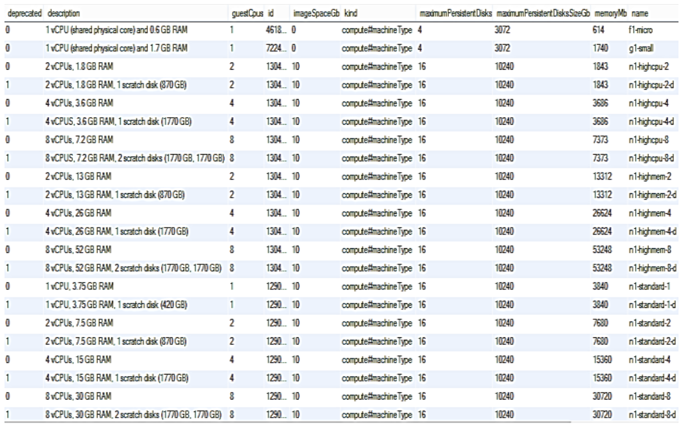
Memory: The size of memory in Megabyte.

Location: The location of the server, e.g. Central US, West Europe, East Asia… etc.

ServerType: The type of the server, e.g. Windows, Linux, SQL, Oracle…etc.

ServerImage: The image of the server, which contains the boot loader, an operating system and a root file system that is necessary for starting an instance, e.g. debian-7, centos-6, rhel-6, sles-11, Windows Server 2012 R2 Data-center, SQL Server 2012 SP1 Enterprise, OpenSUSE 13.1, Ubuntu Server 14.04 LTS… etc.

Fig. 10 is a snapshot of the current metadata collected about available resources



deprecated	description	guestCpus	id	imageSpaceGb	kind	maximumPersistentDisks	maximumPersistentDisksSizeGb	memoryMb	name
0	1 vCPU (shared physical core) and 0.6 GB RAM	1	4618...	0	compute#machineType	4	3072	614	f1-micro
0	1 vCPU (shared physical core) and 1.7 GB RAM	1	7224...	0	compute#machineType	4	3072	1740	g1-small
0	2 vCPUs, 1.8 GB RAM	2	1304...	10	compute#machineType	16	10240	1843	n1-highcpu-2
1	2 vCPUs, 1.8 GB RAM, 1 scratch disk (870 GB)	2	1304...	10	compute#machineType	16	10240	1843	n1-highcpu-2-d
0	4 vCPUs, 3.6 GB RAM	4	1304...	10	compute#machineType	16	10240	3686	n1-highcpu-4
1	4 vCPUS, 3.6 GB RAM, 1 scratch disk (1770 GB)	4	1304...	10	compute#machineType	16	10240	3686	n1-highcpu-4-d
0	8 vCPUs, 7.2 GB RAM	8	1304...	10	compute#machineType	16	10240	7373	n1-highcpu-8
1	8 vCPUS, 7.2 GB RAM, 2 scratch disks (1770 GB, 1770 GB)	8	1304...	10	compute#machineType	16	10240	7373	n1-highcpu-8-d
0	2 vCPUs, 13 GB RAM	2	1304...	10	compute#machineType	16	10240	13312	n1-highmem-2
1	2 vCPUs, 13 GB RAM, 1 scratch disk (870 GB)	2	1304...	10	compute#machineType	16	10240	13312	n1-highmem-2-d
0	4 vCPUs, 26 GB RAM	4	1304...	10	compute#machineType	16	10240	26624	n1-highmem-4
1	4 vCPUs, 26 GB RAM, 1 scratch disk (1770 GB)	4	1304...	10	compute#machineType	16	10240	26624	n1-highmem-4-d
0	8 vCPUs, 52 GB RAM	8	1304...	10	compute#machineType	16	10240	53248	n1-highmem-8
1	8 vCPUs, 52 GB RAM, 2 scratch disks (1770 GB, 1770 GB)	8	1304...	10	compute#machineType	16	10240	53248	n1-highmem-8-d
0	1 vCPU, 3.75 GB RAM	1	1290...	10	compute#machineType	16	10240	3840	n1-standard-1
1	1 vCPU, 3.75 GB RAM, 1 scratch disk (420 GB)	1	1290...	10	compute#machineType	16	10240	3840	n1-standard-1-d
0	2 vCPUs, 7.5 GB RAM	2	1290...	10	compute#machineType	16	10240	7680	n1-standard-2
1	2 vCPUs, 7.5 GB RAM, 1 scratch disk (870 GB)	2	1290...	10	compute#machineType	16	10240	7680	n1-standard-2-d
0	4 vCPUs, 15 GB RAM	4	1290...	10	compute#machineType	16	10240	15360	n1-standard-4
1	4 vCPUs, 15 GB RAM, 1 scratch disk (1770 GB)	4	1290...	10	compute#machineType	16	10240	15360	n1-standard-4-d
0	8 vCPUs, 30 GB RAM	8	1290...	10	compute#machineType	16	10240	30720	n1-standard-8
1	8 vCPUs, 30 GB RAM, 2 scratch disks (1770 GB, 1770 GB)	8	1290...	10	compute#machineType	16	10240	30720	n1-standard-8-d

Fig. 10. Manager's metadata

After the job is received by the manager, it will look for the suitable resource available from this metadata, then start a new virtual machine with specific properties on the specific cloud provider, then it will send the task to this particular virtual machine. Another agent of the manager will monitor the execution of these tasks on these resources, and will send periodic notifications to inform when the task or job execution is completed. To minimize costly communication of large amounts of data, some intercloud mapping feature is needed, so most clouds provides a feature called VHD (Virtual Hard Disk), which is a file format used as the hard disk of a virtual machine that may contain any amount of data. It is portable to be attached to any cloud virtual machine from any provider, so the manager can attach such a VHD to any created VM. After finishing the job execution, the manager will make a request to terminate the VM and deals locate the associated resources. The integration of the GCS API with AMGCS agents is illustrated in Fig. 11.

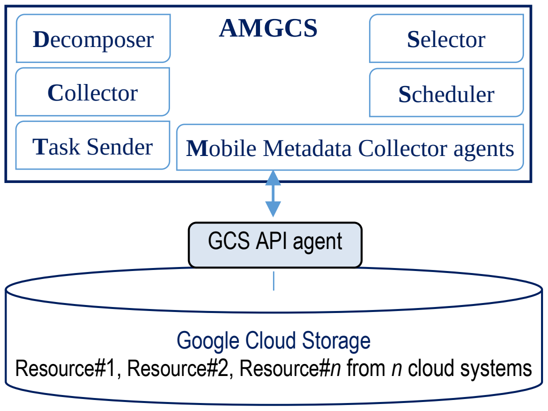

AMGCS
Decomposer
Selector
Collector
Scheduler
Task Sender
Mobile Metadata Collector agents
GCS API agent
Google Cloud Storage
Resource#1, Resource#2, Resource#n from n cloud systems

Fig. 11. Google Cloud Storage integration with AMGCS

VI. Evaluation and Testing

To evaluate AMGCS system, various testbeds have been set up to measure its efficiency, and we have successfully tested the execution of complex jobs that require high computations. If a compute-intensive job is requested to be executed with specified user desires like performance, cost or reliability, the AMGCS manager selects the proper resource from the metadata and sends the job to that resource in order to execute it. This process requires having full information about the resources available in the grid clouds, which is done through our manager by calling the API functions associated with each cloud. There is an updated list of this information in the manager's metadata.

A. *Test cases and Results*

With a centralized manager, the AMGCS has been tested with a compute-intensive job, a big matrices multiplication job. These matrices are huge and need a long time to be manipulated. The size of the first matrix is [4096][2048] and the second matrix size is [2048][2048], this calculation would take a long time to be done, depending on the type of the machine that executes the job, memory, location, server type and so on. The first test case is a single job executed on a single cloud system, with the required user desires: Low cost and minimum execution time. If a regular user wants to perform this job on a cloud system, he will just select any cloud with any properties as he is looking for a low cost, he might manually select the lowest-cost resource to execute this job. In contrast, the AMGCS manager will select the most proper resource that suits the user desires, and achieves this job efficiently.

Consider, as a case, the user has sent the job arbitrary to a lowest-cost resource, which is a virtual machine with a shared core, the job of multiplying these huge matrices took approximately 43.8 minutes to be done. However, AMGCS manager submitted this job to a more proper VM from the list of resources available in the manager's metadata, it is also a shared core but with capabilities that make this VM the best option to select from the available resources in the grid cloud, "Memory and *ServerImage*". Fig. 12 shows the result of this test case.

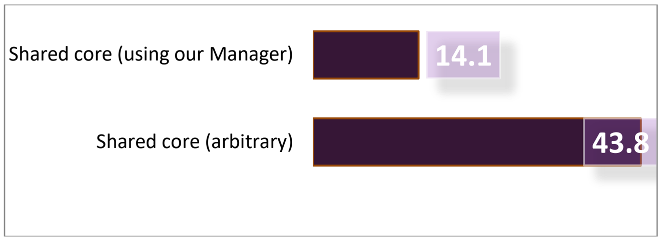


Fig. 12. Comparison of execution time (minutes), shared core

The optimized execution is shown in Fig. 12 is achieved by executing the job on a VM with proper properties (capabilities), and because the manager knows the full details about all resources from the metadata, it chose one cloud of the grid clouds and create a proper virtual machine to execute this job. The configuration of this particular VM customized the memory, and the server image (Debian 7 Wheezy). Here, obviously, one server is better than the other and hence the significant difference in execution time between them. The server type at the bottom is a Windows server, and the type of the upper (optimized) one is a Linux server.

To prove that the enhancement here is achieved by the manager's selection strategy and not by the server type (Linux or Windows), we did a second test to arbitrary execute the same job on a Linux server also with a shared core, but without using our manager. The results proved that the AMGCS scheduling is the reason for that significant enhancement, because of the many options that can be customized to a particular VM to make it the best proper option to execute the required tasks, unlike the regular user's selection that may ignore any consideration to the capabilities or properties of the server's VM. Fig. 13 shows the execution time on the other server type (Linux) without using AMGCS, it is almost near the time taken on the Windows server in Fig. 12, only four minutes less.

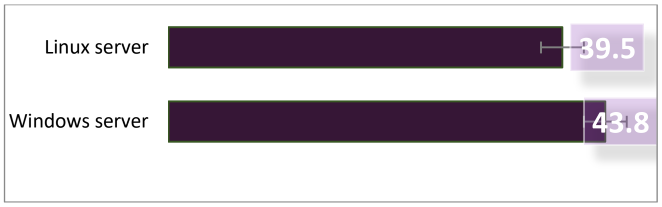


Fig. 13. Execution time (minutes) without AMGCS, shared core

A third test case, the same job but different user desire, which is minimum execution time. Here the user does not care about the cost and the most care about the execution time. It also tested by submitting this job arbitrary to any cloud with any properties and compare this with the selection of our manager which depends on knowledge of the user desires, job structure and nature, resources available and the recommended resources for complex jobs. As we need here the minimum execution time, a high CPU power is required to solve this job as fast as possible. Hence, a virtual machine with eight cores is the proper one. Yet, even with eight cores, the performance could be optimized further by taking into account factors that might degrade the performance or efficiency of execution, like memory and server type. So here the job has been executed on multiple VMs that have the same core number, eight cores. Fig. 14 shows the difference in execution time; the upper one is much faster compared to the arbitrary selected VM at the bottom. This is because the manager has submitted the job to a more suited cloud with better virtual machine capabilities (*serverType* and *memory* space).

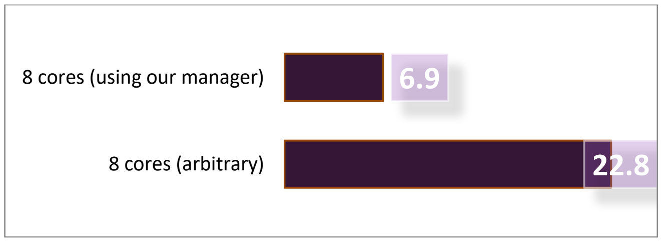


Fig. 14. Comparison of execution time (minutes), 8 cores

All these aforementioned test cases have submitted the job to the clouds' VMs without decomposing it into tasks. Decomposing this big job into tasks to be individually executed on multiple clouds will increase the performance and reliability of the job execution.

The fourth test case is to measure the improvement of enhancement after decomposing a job, the same job has been divided into two tasks (parts) to be executed on the grid cloud system, with one user desire which is minimum execution time. Decomposition here is programmed in the code just to test the prototypal manager, by dividing the first matrix by half and keep the second as it is, to maintain the matrix multiplication rules. Now the manager has many options to execute these tasks; one of these options is to send each of these tasks to a different virtual machine in order to be executed separately and then combine the results together. This is the case here, where the two tasks of the multiplication job are processed on multiple clouds from the grid clouds; hence, the time decreased by half. Fig. 15 shows the significant difference in time compared to the execution on single cloud system.

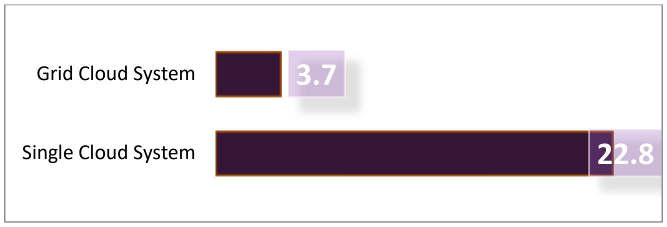


Fig. 15. Comparison of execution time 'minutes', Single vs. Grid Cloud

The fifth test case was conducted to evaluate the improvement of using grid clouds in executing tasks, by sending replications of these tasks (parts of the job) to multiple clouds. Each task has been replicated and processed two times on multiple virtual machines (other than previously used VMs) so if any failure occurs in any VM we still have another copy on another VM. Hence, the reliability of execution is guaranteed for these tasks, despite the cost that might be high because here reliability is the user desire and reliability is always costly. Fig. 16 shows the results of this experiment.

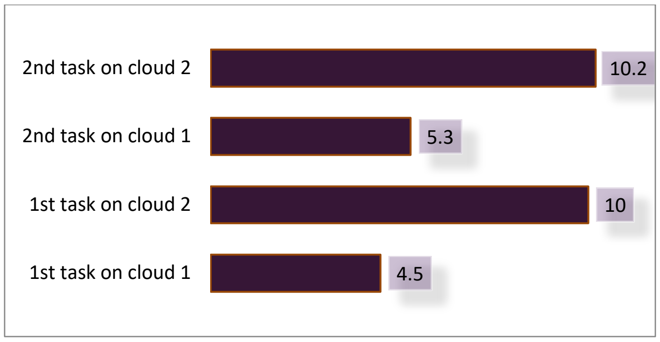


Fig. 16. Execution time (minutes) for tasks of the job, on Grid Cloud

We end up with an enhancement in executing complex tasks on grid cloud resources in an efficiently managed way. Combining comparisons above proves that there is an improvement by 16% - 30% between single and grid cloud system, illustrated in Fig. 17.

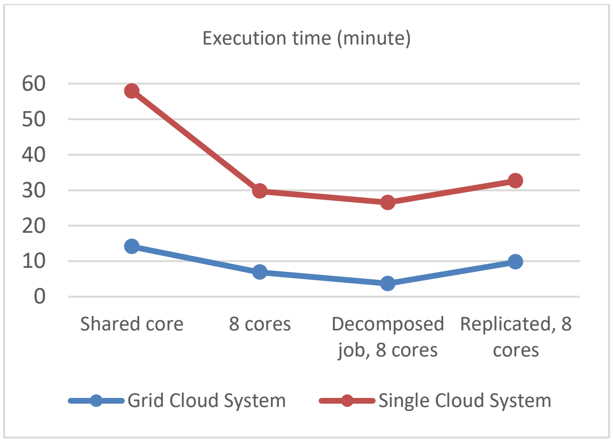


Fig. 17. Overall enhancement, AMGCS vs. Single Cloud

B. Discussion on Experiments Results

Depending on the results that we got from the performed tests and experiments, our finding could be summarized as follows:

- The Manager solves current challenges of executing tasks on the cloud, utilizes grid clouds' resources, solves complex and compute-intensive tasks, and tasks that require high reliability and high performance.
- AMGCS manages jobs and resources and gives good performance in terms of execution time, resource utilization and system throughput compared to a single cloud system.
- Increasing the number of the grid clouds in the system gives more optimize options and high performance compared to using a small number of grid clouds.
- The overall enhancement of Grid Cloud System is about 16% - 32% compared to a single Cloud System.
- The manager does not require any provider-side agreement, only configuring the libraries of the grid clouds.

- Fault tolerance is guaranteed by replication and increased performance through scaling resources to accommodate user's needs, more or less.
- There is a trade-off between high reliability and cost, our manager may replicate tasks on multiple clouds and hence more cost.

VII. Conclusion

In this paper, we introduced an agent-based manager for grid cloud system that has been designed based on software agents to ensure platform independency, heterogeneity handling and flexibility of managing grid clouds. It has been designed, implemented and successfully tested on real clouds. The limitations of the proposed manager could be summarized in its inability to be fully interoperable between different virtualization technologies and recourses compatibilities from different providers. But this interoperability issues could be solved later when cloud standards are clearly defined and followed by all providers to allow such perfect integration between their technologies and resources. The benefits of using AMGCS are shown in increasing and optimizing the available compute power, managing jobs/resources, and utilizing grid clouds' IaaS resources through integration between system's modules and clouds' APIs. This idea can be beneficial to research centers to solve real-world complex problems that need high computing capabilities, such as Bioinformatics applications, engineering simulations, and mathematical analysis.

Acknowledgment

This Project was funded by Deanship of Scientific Research (DSR), King Abdulaziz University, under grant No. (24/34 Research Group). The Authors, therefore, acknowledge technical and financial support of KAU.

References

[1] Vladislav Falfushinsky, Olena Skarlat, Vadim Tulchinsky, "Cloud computing platform within Grid Infrastructure", Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), 2013 IEEE 7th International Conference on (Volume:02), Sept. 2013.

[2] Armbrust M, Fox A, Griffith R, Joseph AD, Katz RH, Konwinski A, Lee G, Patterson DA, Rabkin A, Stoica I, Zaharia M, "Above the Clouds – A Berkeley View of Cloud", Tech-nical report UCB/EECS-2009-28, EECS Department, Uni-versity of Berkeley, California, 10 February 2009.

[3] Stanoevska-Slabeva, Katarina; Wozniak, Thomas; Ristol, Santi, "Grid and cloud computing: a business perspective on technology and applications", Springer, 2010.

[4] Beaty, Donald, "Cloud computing 101", ASHRAE Journal, Volume 55, Issue 10, p. 88. Oct. 2013.

[5] Jha S, Merzky A, Fox G, "Clouds Provide Grids with Higher-Levels of Abstraction and Explicit Support for Usage Modes". Presentation for Open Grid Forum (OGF) 2008.

[6] José C. Cunha and Omer F. Rana, "Grid Computing: Soft-ware Environments and Tools", ISBN: 978-1-84628-339-0, Springer 2006.

[7] M. Wooldridge, "An Introduction to Multiagent Systems", second ed. John Wiley & Sons, 2009.

[8] K. M. Sim, "Agent-Based Cloud Computing", IEEE Transactions On Services Computing, VOL. 5, NO. 4, December 2012.

[9] A. Lonea, D. Popescu, and O. Prostean, "A survey of management interfaces for eucalyptus cloud," in Applied Computational Intelligence and Informatics (SACI), 7th IEEE International Symposium on, pp. 261–266. May 2012.

[10] X. Wen, G. Gu, Q. Li, Y. Gao, and X. Zhang, "Comparison of open-source cloud management platforms: Openstack and opennebula," in Fuzzy Systems and Knowledge Discovery (FSKD), 9th International Conference on, pp. 2457 –2461. May 2012.

[11] L. Xu and J. Yang, "A management platform for eucalyptusbased iaas," in Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on, Sept. 2011, pp. 193 –197.

[12] C. Baun, M. Kunze, and V. Mauch, "The koala cloud man-ager: Cloud service management the easy way," in Cloud Computing (CLOUD), IEEE International Conference on, pp. 744 –745. July 2011.

[13] C. Baun and M. Kunze, "The KOALA cloud management service: a modern approach for cloud infrastructure management," in Proceedings of the First International Workshop on Cloud Computing Platforms, ser. CloudCP '11. New York, NY, USA: ACM, p. 1:1–1:6. 2011.

[14] "Scalr," [online] Available at: http://github.com/Scalr/. [Accessed: 01 January 2017].

[15] "Apache libcloud," [online] Available at: http://libcloud.apache.org/. [Accessed: 01 January 2017].

[16] "jcloud," [online] Available at: http://www.jclouds.org/. [Accessed: 01 January 2017].

[17] "Apache deltacloud," [online] Available at: http://deltacloud.apache.org/. [Accessed: 19 July 2015].

[18] Yi Wei and M. Brian Blake, "Adaptive Service Workflow Con-figuration and Agent-based Virtual Resource Management in the Cloud", Cloud Engineering (IC2E), IEEE International Conference on, March 2013.

[19] Metsch T., Edmonds. A., et al. Open Cloud Computing Interface Core and Models, Standards Track, no. GFD-R in The Open Grid Forum Document Series, Open Cloud Computing Interface (OCCI) Working Group, Muncie (IN) 2011.

[20] Venticinque S., Tasquier L., Di Martino B., "Agents based Cloud Computing Interface for Resource Provisioning and Management", Sixth International Conference on Complex, Intelligent, and Software Intensive Systems, 2012.

[21] Domenico Talia, "Cloud Computing and Software Agents: Towards Cloud Intelligent Services", WOA, volume 741 of CEUR Workshop Proceedings, page 2-6. CEUR-WS.org, 2011.

[22] ZJ Li, Chen C. and Wang K., "Cloud Computing for Agent-Based Urban Transportation Systems", IEEE Computer Society, 2011.

[23] M.V. Haresh, S. Kalady and V.K. Govindan, "Agent based Dynamic Resource Allocation on Federated Clouds," Proc. IEEE Recent Advances in Intelligent Computational Systems (RAICS'11), pp.111 - 114. 2011.

[24] Del Castillo, Lorenzo and others, "OpenStack Federation in Experimentation Multi-cloud Testbeds." HP Laboratories. 2013.

[25] Kurze, Tobias, et al. "Cloud federation." CLOUD COMPUTING, The Second International Conference on Cloud Computing, GRIDs, and Virtualization. 2011.

[26] Rawat, S. and Rajamani, L., "Experiments with CPU Scheduling Algorithm on a Computational Grid ", IEEE International Advance Computing Conference (IACC 2009), PP. 71-75. 2009.

[27] Chunlin, Li, Zhong Jin Xiu, and Li Layuan. "Resource scheduling with conflicting objectives in grid environments: Model and evaluation." Journal of Network and Computer Applications 32, no. 3: 760-769. 2009.

[28] R. Buyya et al., "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility," Future Generation Computer Systems, vol. 25, no. 6, pp. 599- 616, June 2009.

[29] K.M. Sim, "Towards Complex Negotiation for Cloud Economy," Proc. Int'l Conf. Advances in Grid and Pervasive Computing (GPC '10), R.S. Chang et al., eds., pp. 395-406, 2010.

[30] K.M. Sim, "Towards Agent-Based Cloud Markets (Position Paper)," Proc. Int'l Conf. E-CASE, and E-Technology, pp. 2571-2573, Jan. 2010.