Author/s:
Somasundaram, TS;Govindarajan, K;Kiruthika, U;Buyya, R

Title:
Semantic-enabled CARE Resource Broker (SeCRB) for managing grid and cloud environment

Date:
2014-01-01

Citation:
Somasundaram, T. S., Govindarajan, K., Kiruthika, U. & Buyya, R. (2014). Semantic-enabled CARE Resource Broker (SeCRB) for managing grid and cloud environment. Journal of Supercomputing, 68 (2), pp.509-556. https://doi.org/10.1007/s11227-013-1047-z.

Persistent Link:
https://hdl.handle.net/11343/282724

Semantic-enabled CARE Resource Broker (SeCRB) for Managing Grid and Cloud Environment

Thamarai Selvi Somasundaram[1], **Anna University, Chennai, India** stselvi@annauniv.edu

Kannan Govindarajan[1], **Anna University, Chennai, India** kannan.gridlab@gmail.com

Usha Kiruthika[1], **Anna University, Chennai, India** usha.kiruthika@gmail.com

Rajkumar Buyya
Cloud Computing and Distributed Systems Lab,
The University of Melbourne,
Parkville, Australia
raj@csse.unimelb.edu.au

Abstract
Grid Computing is mainly helpful for executing High-Performance Computing (HPC) applications. However, conventional grid resources sometimes fail to offer a dynamic application execution environment and this increases the rate at which the job requests of users are rejected. Integrating emerging virtualization technologies in grid and cloud computing facilitates the provision of dynamic virtual resources in the required execution environment. Resource brokers play a significant role in managing grid and cloud resources as well as identifying potential resources that satisfy users' application requests. This research paper proposes a semantic-enabled CARE Resource Broker (SeCRB) that provides a common framework to describe grid and cloud resources, and to discover them in an intelligent manner by considering software, hardware and Quality of Service (QoS) requirements. The proposed semantic resource discovery mechanism classifies the resources into three categories viz., exact, high-similarity subsume and high-similarity plug-in regions. To achieve the necessary user QoS requirements, we have included a Service Level Agreement (SLA) negotiation mechanism that pairs users' QoS requirements with matching resources to guarantee the execution of applications, and to achieve the desired QoS of users. Finally, we have implemented the Quality of Service (QoS)-Based Resource Scheduling (QBRS) mechanism that selects the resources from the SLA negotiation accepted list in an optimal manner. The proposed work is simulated and evaluated by submitting real-world bio-informatics application for various test cases. The result of the experiment shows that, for jobs submitted to the resource broker, job rejection rate is reduced while job success and scheduling rates are increased, thus making the resource management system more efficient.

Keywords: Cloud Computing, Grid Computing, High-Performance Computing (HPC), Resource Broker, Semantic Description, Semantic Discovery, Service Level Agreement (SLA).

1. INTRODUCTION

Grid Computing [1] is the federation of computer resources from multiple administrative domains to solve scientific applications involving High Performance Computing (HPC). It consists of a heterogeneous, large-scale and dynamic environment, and it aims to exploit the usage of underutilized resources and the parallel processing capability of HPC applications. However, the major drawback of grid is its inability to provide the required execution environment due to its rigid nature. Thus, it leaves certain user application requirements unsatisfied and this result in the failure or rejection of some jobs. This drawback has been overcome through integrating virtualization technology with grid and cloud computing, and provisioning ~~of~~ resources in an on-demand manner. This flexibility in resource provisioning, in addition to the conventional features of grid, enables improved throughput and success rate of jobs. NIST [2] characterizes cloud computing in terms of on-demand self-service, broad network access, resource pooling, rapid elasticity and measured service. The cloud service models are categorized into "Software as a Service" (SaaS), "Platform as a Service" (PaaS), and "Infrastructure as a Service" (IaaS). In SaaS, the consumer gets the provider's applications that are hosted and run in cloud infrastructure. In PaaS, the consumer gets the platform that is running in the cloud infrastructure for creating or deploying the applications. In IaaS, the consumer gets the virtual resources in which an execution environment can be deployed to run an application. Virtualization concepts [3] create a wide range of opportunities for using virtualized resources. Correspondingly, it provides the multitenant [4] capability of running multiple virtual machines, or resources sharing the same hardware, in an isolated manner for different application purposes.

The management of the grid and cloud resources is a serious and complex issue. The complexity of the resource management is due to the dynamic nature of scaling and shrinking resources in a grid and cloud environment. Czajkowski et al. [5] defined that "Resource Management is related to the operations that are mainly used to control the capabilities provided by grid resources and services that are made available to users,

applications or services". In addition, they explained that, the main objective of resource management is to establish a mutual agreement between a resource provider and a resource consumer. In this paper, we have proposed a semantic based resource management framework to manage both the grid and cloud resources. When the required grid resources are not available, cloud resources are used to provide virtual resources dynamically. Consequently, this work is an extension of the existing grid resource broker to manage both the grid and cloud resources in an interoperable and efficient manner. The main challenge in this proposed work involves co-coordinating the use of the resources available in the grid or cloud environments. Moreover, these resources may reside in different geographical locations, managed by different organizations, and thus, there are no standardized interfaces, protocols or commands. To coordinate the usage of resources, and to solve the interoperability issue between different grid and cloud resources, there is a need to integrate semantic technology in the resource broker [6]. According to Berners Lee et al. [7], a semantic web is a cluster or group of information arranged in hierarchical structure and it is easily understandable by computers. In addition, it is an efficient way of representing data on the World Wide Web (WWW). Ontology [8] is the language mainly used for representing the available information. Semantic Grid [9] is a recent initiative to expose semantically rich information associated with grid resources to build more intelligent grid services. Semantic web and semantic grid use a Resource Description Framework (RDF) [10] to create an ontology knowledge base. It is used as a standard model for interchanging data on the web and grid. It is able to merge data even when the underlying schemas are different in nature. Web Ontology Language (OWL) [11] is one of the popular ontology languages recommended by the World Wide Web Consortium (W3C). It is the combination of the descriptions of classes, properties and their instances. Protégé [12] is an editor used for creating an ontology-based knowledge base.

A CARE Resource Broker (CRB) [13] is a grid metascheduler or resource broker that is deployed over the Globus Toolkit 4 (GT4) [14] Middleware to manage grid, and virtualization enabled grid resources. It addresses several scheduling issues encountered when a virtualization technology is integrated with the grid environment. It has the ability to create and manage virtual resources on a virtualization enabled grid. This feature allows CRB to make several application decisions that conventional grid schedulers may not do. It obtains the user's requirement request for running an application in the grid resources, aggregates the grid resource information and discovers the suitable physical resources that perfectly match the application requirements. The various scheduling scenarios that are handled by the CRB are as follows:

A. Application requests ~~to~~ that are run in the existing physical grid resources.
B. Application requests that are run by creating some virtual resources and attaching these virtual resources to the existing physical grid resources (co-allocation) [15].
C. Application requests that cannot be run in the existing physical Grid resources giving that application execution environment are unavailable, in terms of software and operating systems. A virtual cluster is dynamically created, deployed and configured for the execution of application (virtual cluster).
D. Application requests that are used for leasing virtual resources (virtual machines and virtual cluster) [16].

Our proposed work considers the scheduling scenarios B, C and D to create virtual resources in the grid and cloud resources residing in an organization. The main aim of the proposed work is to extend the conventional CRB to a Semantic-enabled CRB (SeCRB) and to integrate a Service Level Agreement (SLA)-driven resource allocation for managing the grid and cloud resources. We have devised and implemented a common semantic resource management framework for describing and discovering grid and cloud resources based on the user's application requirements. The proposed framework mainly aims to solve the interoperability between grid and cloud resources, and effectively allocate job requests to the resources that satisfy the user's QoS requirements. The comparison between CRB and SeCRB is shown in Table 1.

Table 1: Comparison of CRB and SeCRB

S.No.	CRB	SeCRB
1	Keyword Based Match Making	Semantic Based Match Making
2	First Come First Serve (FCFS) Based Job Prioritization	Deadline Based Job Prioritization
3	Rank-Based Resource Allocation	QBRS Based Resource Allocation
4	Static Image Management System	Automated Image Metadata Management System
5	Managing Grid Resources Only	Managing Grid and Cloud Resources
6	Object-Oriented Implementation	Service-Oriented Implementation
7	Grid Monitoring and Discovery Service	Introduction of Cloud Monitoring and Discovery Service
8	Tested with Image Processing Application	Tested with Image Processing and Bio-Informatics Application

The main contributions of the paper are summarized as follows:

- ✓ A semantic-based resource description mechanism to semantically represent the grid and cloud resources information and the operating system images in the ontology knowledge base.
- ✓ A semantic-based resource discovery mechanism to semantically discover the potential resources for virtual resource creation and application execution.
- ✓ The implementation of an SLA negotiation mechanism and a Quality of Service (QoS)-Based Resource Scheduling (QBRS) mechanism that selects the grid and cloud resources in a near optimal manner.
- ✓ The integration of a Cloud Monitoring and Discovery Service (CMDS) to discover and monitor the cloud resources information.
- ✓ The design and development of an Image Metadata Management System to efficiently manage the available operating system images from the grid and cloud resources, which are required for virtual resource creation.
- ✓ The testing of the proposed work in real grid and cloud environments for executing real-world scientific applications and comparing the total execution time of submitted applications.
- ✓ The comparison of various performance parameters such as job rejection rate, job success rate and scheduling success rate, and the processing of the overhead for the semantic-based resource management.

The rest of the paper is organized as follows:
Section 1 gives the introduction and motivation for the proposed research work. Section 2 discusses some related works that motivated ours. Section 3 describes the high-level architectural structure of our proposed work. Section 4 presents the design and implementation details in a thorough manner. Section 5 covers the working principle of the proposed framework. Section 6 introduces the real-time experimental setup made ~~in~~ on our campus, and discusses operating system images and real-world application. The generation of workloads, performance metrics and simulation results are reviewed in section 7. Finally, section 8 concludes the paper by highlighting the features of the proposed work and exploring ideas for possible future work.

2. RELATED WORKS

A lot of research has been done on semantic grid and cloud, and this section discusses some closely related papers that motivated us. Condor [19] uses ClassAds for matchmaking to solve resource allocation problems in a distributed environment. This framework focuses mainly on the matchmaking process that is based on constraints specified by the resource providers and consumers. Computing Resource Execution and Management (CREAM) [20] has been developed and integrated with gLite Workload Management System (WMS) [21]. Using the CREAM, users can submit jobs that are described using Classified Advertisements (ClassAds) [22] and Job Description Language (JDL) [23]. JDL is a high-level, user-oriented language. It works mainly on the principle of the ClassAds developed as a part of the Condor project. It is used to describe job characteristics and requirements that are submitted to the gLite WMS. ClassAds works on the principle of key and value pair. InteliGrid [24] proposed a three layer semantic architecture consisting of conceptual, software and basic resource layers. The concept layer provides ontology descriptions about knowledge entities. The software layer describes software that use the knowledge entities outlined in the concept layer. The basic resource layer presents infrastructure and fabric resources. The Grid Interoperability Project (GRIP) [25] addressed the problem of resource description in the context of a resource broker. The resource broker is compatible with Grid Middleware such as GT2, GT3 and Unicore. GRIP proposed a semantic-based approach to describe the resource information aggregated from different middleware-enabled grid resources in the broker level. Another project "myGrid" [26] is a multi-organizational project mainly aimed at developing infrastructure middleware. It operates on the existing web services and grid infrastructure, and helps scientists using distributed resources to work with ease. The middleware has been integrated with semantic technology for resource discovery and workflow management. Finally, it provides access to Bioinformatic~~s~~ archives and the tools for analysis through web service technologies. The Semantic Grid [27] project exposes semantically rich information associated with Grid resources to build more intelligent Grid services. It takes the structured semantic descriptions of real commodities that have associated identities and behaviors.

Thamarai Selvi et al. [28] proposed a semantic component that supports context-based Grid resource discovery. It has been integrated with Gridbus Broker to identify potential grid resources in a semantic manner. An ontology-based Matchmaker Service proposed in [29] provides the support for a dynamic resource discovery and resource descriptions. They concluded that users' application requests should be expressed as ontology descriptions. Dynamic Virtual Clustering (DVC) [30] is a system for deploying virtual machines in a cluster or multi-cluster environment. The main goals of DVC are reducing job turnaround time and queue wait time, and increasing cluster utilization and workload throughput. Virtual Machines are useful for giving users a fraction of their system's resources, or providing a different software environment to the host system. However, when applied to HPC applications, virtual machines perform worse on application execution relative to native

execution [31]. Eucalyptus [32] is one of the most popular open source cloud middleware. It is used for managing private cloud infrastructures. It is arranged in a hierarchical manner, and ~~it~~ has the following four main service-based components: Cloud Controller, Cluster Controller, Node Controller and Storage Controller. The Cloud Controller is responsible for managing and coordinating cluster and node controllers in the same availability zone. It acts as an interface for clients' requests. The main functions of a Cluster Controller are application provisioning, monitoring of the state information of the virtual instances that running in physical resources and so on. Node Controllers are responsible for creating, deleting and monitoring virtual instances by interacting with hypervisors running on physical resources. The Storage Controllers are responsible for managing data and storing related concepts. OpenNebula [33] is an open source virtual infrastructure manager, and it is used to manage the life cycle of a virtual machine's creation, monitoring and deletion. It manages the resources with different types of hypervisors such as Xen [34], KVM [35] and VMWare [36]. It also has the provision to interface with Amazon EC2 [37] Cloud.

Yong Beom Ma et al. [38] proposed an ontology based resource management for cloud computing to define concepts and relations. They have implemented an SLA-based resource allocation for the selection of resources. Baomin Xu et al. [39] proposed the idea of building a cloud infrastructure on clusters. The main aim of their proposed work is to offer a virtual resource manager to control the compute nodes in the cluster. They are yet to discuss semantic concepts. Rajkumar Buyya et al. [40] proposed and developed the Cloudbus toolkit that works on the principle of market-based resource management and computational resource risk management to maintain an SLA-oriented resource allocation. Jorge Ejarque et al [41] proposed an approach for managing cloud resources using Semantic Resource Allocation as a multi-agent distributed system. Their proposed approach combines the advantages of semantic web and agent technologies for managing the cloud resources of different resource providers. Balakrishnan and Thamarai Selvi [42] proposed SLA-aware CRB for negotiating job requirements with available grid resource information and monitoring the resource consumption of jobs. Jorge Ejarque et al. [43] extended their previous work of [41] as SLA-Driven Semantically Enhanced Resource Allocation (SERA). The resource allocation process is carried out using the negotiation process with the resource provider's agents. The last two papers mentioned motivated us to extend the SLA aware strategy in our proposed work for managing cloud resources. Karl Czajkowski et al. [44] classified SLA into three categories namely Resource SLAs (RSLA), Task SLAs (TSLA) and Binding SLAs (BSLA). They proposed SNAP protocol, which negotiates Grid resources through the exchange of messages between the three categories. In addition, they have addressed the issue of acquiring multiple resources using SLA negotiation.

Srikumar Venugopal et al. [45] proposed a protocol for negotiation based on the concept of alternating offers. Their proposed work is integrated with advance reservation mechanism that reserves negotiation-accepted resources in advance. Jun Yan et al. [46] proposed a framework for automated negotiation using agents in the context of service composition. The service consumer is represented by a set of agents who negotiate the parameters with individual services in the composition. Seokho Son et al. [47] proposed a negotiation mechanism to negotiate multiple issues like price, timeslot and other QoS parameters in a cloud environment. The StratusLab [48] project is aimed at providing virtual resources in an on-demand manner. They - used KVM hypervisors for creating virtual resources. Recently, they integrated the OpenNebula as a cloud management tool to create virtual resources to dynamically provision resources and enhance scalability. WNoDes [49] project effectively makes use of the largely invested Grid Computing Infrastructure. Its main aim is to create virtual resources, in an on-demand manner, of the jobs that are queued due to the unavailability of software requirements in the existing grid environment. The Grid Lab Uniform Environment (GLUE) Schema v1.3 [50] is currently used in the gLite-based Grid infrastructure to describe resource attributes. Initially, ApplicationSoftwareRunTimeEnvironment was used to describe the list of software packages installed in the grid environment. Now, GLUE has been extended to describe the virtual operating systems bundled with application software libraries. An Open Cloud Computing Interface (OCCI) [51] is used to interface with cloud resources. Similar to their work, we have extended the GLUE schema in the broker level to represent the cloud and grid resource information and the same has been represented in the Ontology knowledge base. However, the StratusLab [48] and WNoDes [49] projects, previously mentioned, created the virtual resources that are specifically due to the unavailability of software execution environments. Marcos Dias de Assunção et. al [52] proposed an approach to acquire cloud resources in addition to their local cluster resources, so as to decrease the response time of user job requests. Their proposed work is evaluated with seven scheduling strategies. Finally, they have compared the various performance metrics of job slowdown, number of deadline violations, average weighted response time, number of rejected jobs and cost of improving performance. Their experiment showed that naïve scheduling strategies lead to higher costs for heavy loads. In addition, other scheduling strategies such as request backfilling and selective backfilling have been shown to improve the performance over cloud resources.

Simon Ostermann et al. [53] investigated the benefits of using the cloud compute resources ~~to~~ in an Askalon based grid environment to increase the execution time of scientific workflow applications. To integrate the cloud resources with the grid workflow management, they have introduced three new components: cloud management, software deployment and images repository. Their proposed approach is implemented by a just-in scheduling

mechanism, which creates the cloud resources when the grid resources are not available to run the workflow applications. Alexandru Iosup et al. [54] presented the performance analysis of Cloud computing services for scientific computing applications. The authors examined if the performance of the clouds are enough for Many Task Computing (MTC)-based scientific computing. Finally, their proposed approach is empirically evaluated with four different types of clouds namely Amazon EC2, GoGrid, ElasticHosts and Mosso. They note that the present cloud computing services are not sufficient for scientific computing on a large scale. Ming Mao et al. [55] have made a performance study of virtual machine startup times in the cloud resources, which play a main role in the performance of an application. A long startup time degrades the performance of an application. The startup time of virtual instances has been made in three different types of cloud providers namely Amazon EC2, Windows Azure and Rackspace. They also analyzed factors such as time of the day, OS image size, instance type and data center location, as well as the number of instances acquired at the same time in virtual machine startup time. Their studies are evidence that the startup time of spot instances are longer and have greater variance relative to on-demand instances. Our proposed resource brokering approach identifies the various cases using scheduling. Another major difference between our work and those mentioned above is a proposed semantic component that is capable of handling scheduling scenarios such as Physical, Hybrid, Virtual and Lease. In addition, they have not considered the semantic relationship between different operating system images and the required one. Our proposed work is extensively tested for HPC based scientific applications. It can be easily extended to provide SaaS based solution to different scientific applications. From related works, we have identified various approaches proposed for semantic-based resource management of grid resources. However, the papers mentioned above provide very little knowledge in regards to the implementation of semantic-based resource management of both the grid and cloud resources for HPC applications.

3. PROPOSED LAYERED ARCHITECTURE

The proposed layered high-level architectural view for managing the grid and cloud resources is shown in Figure 1. It is extended for managing the cloud resources in addition to the grid resource management in CRB. Information about resources and operating system images are discovered and described under Knowledge Base.

3.1 Application Layer

The application layer enables the use of the resources in a grid and cloud environment through various collaboration and resource access protocols. The Graphical User Interface (GUI) and Command Line Interface (CLI) presented in this layer enable the service providers and users to register their services and identities into resource broker. In addition, it enables the user to submit a query and semantically retrieve information from the ontology using the proposed matchmaking algorithm.

3.2 Broker Layer

SeCRB resides in the broker layer. It interacts and aggregates the information from the Grid and Cloud the grid and cloud resources information and represents them as concepts and properties in the ontology knowledge base. The ontology knowledge base is created using the Resource Description Language (RDL) of Web Ontology Language (OWL). Protégé is used for creating the ontology knowledge base. It is implemented with an intelligent matchmaking algorithm for identifying the various scheduling scenarios such as physical, co-allocation, virtual cluster and lease. The matchmaking algorithm identifies the potential resources the grid and cloud environments. The resources may fall into exact, high-similarity plugin and high-similarity subsume regions. The SLA-driven resource allocation mechanism negotiates with the resource provider on behalf of the user. Based on the outcome of SLA negotiation, the QBRS algorithm selects the resources in a near optimal manner that satisfies the user desired QoS.

3.3 Middleware Layer

In middleware layer, Globus Toolkit [14] is used for managing the grid resources and Eucalyptus is used as cloud middleware for managing the cloud resources. Information management is responsible for dynamically monitoring the information from the grid and cloud resources. Security management handles security related issues such as authentication and authorization. Instance management in grid and cloud middleware handles the creation, deletion and monitoring of virtual instances. Image management is useful for managing the operating system images uploaded in the grid and cloud resources. Network Management solves networking related issues such as assigning IP addresses. Execution Management in the grid initiates and monitors the execution of jobs.

3.4 Fabric Layer

The fabric level provides the resources that may belong to the grid or cloud environment. The resources in the fabric level include computational resources, storage resources, network resources and database resources.

4. DESIGN AND IMPLEMENTATION DETAILS

To manage both the grid and cloud resources, the existing CRB has been extended to a Service Oriented resource broker. The four major components that have been integrated in SeCRB at broker level are given below:

1. Semantic Resource Management
2. Job Handler and Resource Selection Management

3. Cloud Resource and Image Information Management
4. Controller and Adapter Management

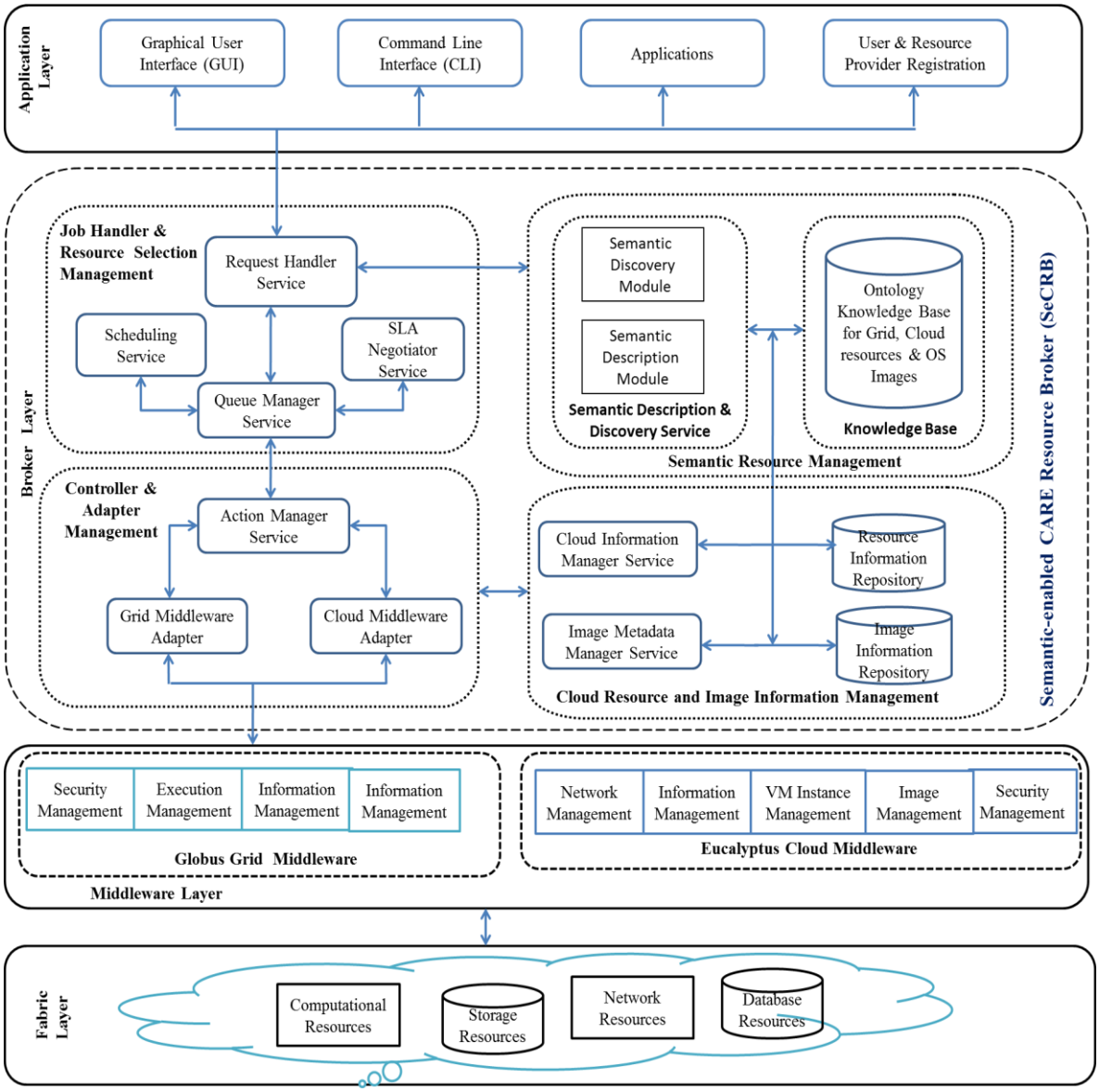

Figure 1: Semantic-enabled CARE Resource Broker (SeCRB)

4.1 Semantic Resource Management

The core part of the semantic component is the grid or cloud Resource Ontology template. An ontology template is a domain specific ontology that provides a hierarchy of concepts and the properties that define their characteristics. The Grid/Cloud Resource Ontology template is structured based on the fact that any resource that can be described using the properties of a specific concept can be modelled as an instance of that concept. Protégé is an ontology editor that creates ontology using a Web Ontology Language. Once the ontology template is created, a knowledge base can be built with the instances and specific property instantiations. The semantic description component shown in Figure 2 is responsible for constructing a knowledge base of Grid/Cloud resources. It obtains the information from three different sources – Monitoring and Discovery Service (MDS), which runs in the Grid resources, Virtual Resource Information Service (VRIS), which deploys and runs in the Grid resources, which supports virtualization, and Cloud Monitoring and Discovery Service (CMDS), which deploys and runs in the Cloud resources. Monitoring and Discovery Service (MDS) in Grid fetch the Grid resources information, once the Grid resources are registered with Grid Middleware. With this information, an instance of the appropriate resource concept in the ontology template is created for every computing resource in the grid. The semantic description module possesses the complete required information with which it can construct a knowledge base of Grid/Cloud resources. Protégé-OWL APIs is used to dynamically create an instance of a particular concept and also to assign values to appropriate properties in the resource ontology template. With these features, the resource information of the entire Grid/Cloud environment

aggregated by the semantic description component can be populated in an ontology template thereby creating a knowledge base. The resource brokering strategy in SeCRB mainly depends on resource ontologies. Ontologies are mainly used to describe the relationship between domains in a machine understandable format. In our proposed work, we have made the following extensions (A) and (B) to semantic based resource management of previous research works of Balachandar et al. [17] and Thamarai Selvi et al. [18] for managing the cloud resources. The semantic-based resource management is implemented using Semantic Description and Discovery Service.

(A) Semantic Description: It describes the grid and cloud resources and operating system images available in the grid and cloud resources.

(B) Semantic Discovery: It develops an intelligent matchmaking system which is capable of handling both the grid and cloud resources for various several scheduling decisions in SeCRB.

4.1.1. Semantic Resource Description

Meaningful and understandable semantic description of resources by the system, is essential for semantic based resource management. The automated tasks of ontology creation and updation are the major tasks in a semantic description module. The resource ontology template is created by the semantic description, and it represents the concepts and properties of the resources. It is a domain-specific ontology that is helpful for representing the concepts and properties that define the characteristics of resources in a hierarchical manner. Ontology is mainly used for understanding the domain information. It describes the concepts in the domain and also relationships between those concepts. OWL is used for creating and representing the ontology template. The knowledge base created is based on the concept of instances and specific property instantiations. The semantic description module as shown in Figure 2 is responsible for constructing a knowledge base, and updating ~~of~~ information about the grid and cloud resources, and the images of operating systems.

The registered grid and cloud resources are monitored by using appropriate grid and cloud middleware adapters. The extended version of the Monitoring and Discovery Service has been developed and deployed in the grid middleware that retrieves information such as architecture, operating system, free ram memory, free hard disk memory, hostname, IP address, processor speed, operating system version, LRMS name, total number of nodes, free nodes, hypervisor type, bandwidth, latency, operating system image details, including the application support, and the hardware details of the compute node. The Cloud Middleware Adapter (CMA) interfaces with the Eucalyptus Cloud middleware to retrieve information such as total memory, free memory, ram memory, hard disk memory, the types of virtual machines and the number of virtual machines that can be created in the physical cloud resources. In a periodic interval, the semantic descriptor module accesses the Grid Resource Repository (GRR), Cloud Resource Repository (CRR) and OS Image Metadata Repository (OSIMR). The information retrieved from the grid and cloud resources is analyzed by the parser module. This parsing feature has been implemented with three types of parsers namely, grid resource information parser to analyze the grid resource information, cloud resource information parser to parse the cloud resource information and image information parser to parse the information of the operating system images and the bundled software libraries. The parsers use the Ontology Creator to update the parsed information to the ontology knowledge base. It is implemented based on Protégé-OWL APIs. Using these APIs, it creates instances and assigns values to the corresponding properties in the resource ontology template in a dynamic manner. For example, an existence of a computer with Linux OS can be represented in the ontology template by creating an instance for the concept “Cluster” and thus, the “hasOS” property of the concept “Cluster” will be assigned the value “Linux”. Similarly, the image information of this resource can be mapped as the appropriate properties described for the concept cluster. With these features, the resource information of the entire grid and the environment aggregated by the semantic description component can be populated in an ontology template thereby creating a knowledge base.

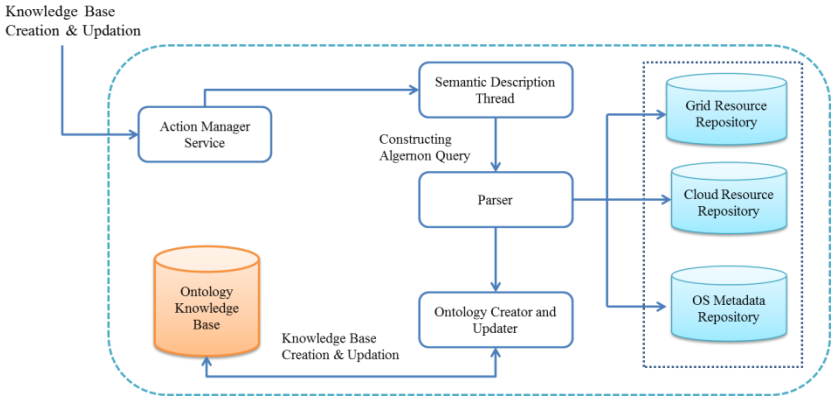

Figure 2: Semantic Description Module

Protégé is used as an ontology editor for creating the ontology template. The created ontology template is shown in Figure 3, which symbolizes the hierarchical representation of the grid and cloud resources. Cluster is a major class and it has a subclass called Head Node. The Head Node has the following subclasses: HypervisorDetails, ImageDetails, NetworkDetails, OSDetails, SoftwareDetails, Compute Node, ApplicationDetails, and HardwareDetails. The HeadNode contains the properties that reflect the details about processor speed (hasProcessorSpeed), free RAM memory (hasFreeRam), free hard disk memory (hasFreeMemory), and so on. The HypervisorDetails has the property of hasHypervisorName to identify the type of hypervisor. The OperatingSystemImageDetails class has the properties to identify the available software "hasAvailableSoftwares" and "hasApplicationSupport" for identifying application support.

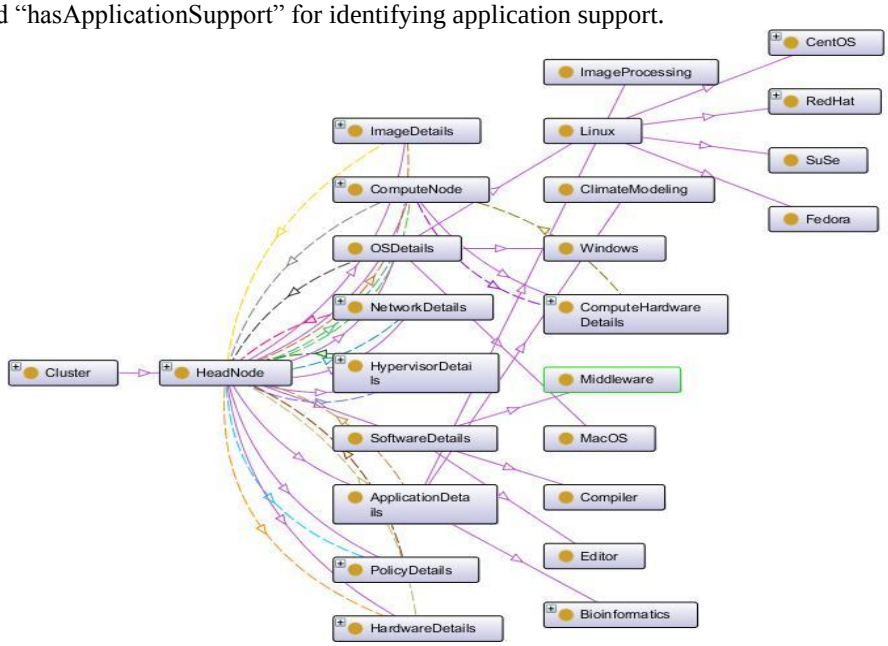

Figure 3: Ontology Template

4.1.2. Semantic Resource Discovery

The main objective of the resource broker is matching the user's application requirements with available resources information. Matchmaking is one of the core components in the resource broker. Based on the output of the matchmaking process the resource broker decides to either consider the job for execution or to reject it because the requested resources are not available. The existing resource brokers, which are available as open-source software, mostly use keyword-based matchmaking for matching the user's application requirements with available resources. However, this process often results in the rejection of application requests because the resource broker exactly matches the keywords given in the user's application requirements with the available resources. For example, if the user requests a Red Hat Enterprise Linux (RHEL) 5.0 for application execution but the available resources have only Cent OS 5.0, the keyword based matchmaking will fail to select the Cent 5.0 resource even though it is a feasible option for running the application. Therefore, there is a strong need for representing the semantic relationship of resource information in the resource broker level.

The semantic discovery component constitutes of the actual resource brokering mechanism, and it implements the matchmaking mechanism. It accepts the user's application requirements expressed in the form of Job Submission Description Language (JSDL). The request expresses hardware, software and Quality of Service (QoS) requirements such as Operating System, Number of Nodes required and Software libraries. The power of the brokering algorithm heavily relies on the ontology reasoner used. In our prototype implementation, we have used an Algernon [56] inference engine for retrieving information from the ontology knowledge base. The proposed brokering algorithm must consider physical and virtual resource information while discovering a suitable resource for a job. Hence, based on the users resource request, the broker constructs an Algernon based query and executes it in knowledge base. The query also determines the semantic relationship between the request and the available options. However, if no such resource is found in the first iteration of a query on the knowledge base, the broker constructs an alternate query to determine the physical resources that match all requirements but possess a lesser number of CPUs than requested. If such resources are available, it will suggest that virtual machines, as many as are still required, should be created in one cluster and connected as computing nodes to another cluster. If it still fails, the broker constructs yet another query to determine the physical resources that exactly matches the hardware requirements but not the software. If such resources are available, the broker will suggest creating virtual clusters corresponding to the user's request. In the last two cases, the broker verifies that the cluster possesses the required virtualization software and can support the creation of

virtual machines. The devised semantic discovery module and the interaction of various components are shown in Figure 4.

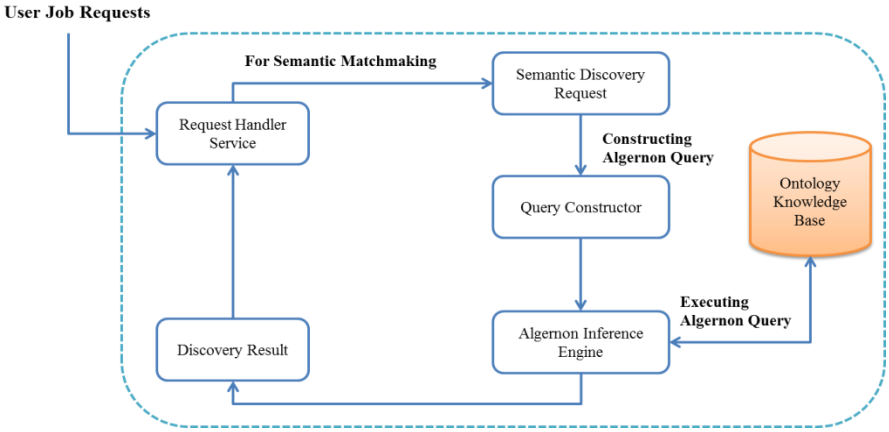

Figure 4: Semantic Discovery Module

The Request Handler Service redirects the job submission requests to Semantic Discovery Service (SDR). It retrieves the application requirements, and the requirements are constructed as Algernon queries by a Query Constructor. The query construction is based on the requirements of the user's hardware, software and applications such as number of nodes, processor speed, ram memory, hard disk memory, bandwidth, etc. The constructed query is sent to the Algernon Inference Engine. It interacts with the ontology knowledge base to find out the potential resource for application execution as well as the virtual resource creation. The matchmaking algorithm implemented in semantic resource discovery module is given in Algorithm 1. It classifies the resources that may fall into three regions namely exact, high-similarity plug-in and high-similarity subsume. The exact region contains resources that perfectly satisfy the hardware and software requirements of the user's job requests. The subsume region contains resources that are overqualified and the plug-in region contains resources that do not meet the demands of the user. Cosine similarity is used to retrieve the resources that are near optimal to the user's request in the subsume and plug-in regions. These are termed high-similarity subsume and high-similarity plug-in regions. The threshold values in the subsume and plug-in regions are fixed between 0 and 1. Let us consider the number of resources in the subsume and plug-in regions to be 'M'. For each resource in the subsume and plug-in regions, cosine similarities are calculated using $\cos\left(J_i, R_j\right)$ where $1 \le j \le M$ & $1 \le i \le N$. A resource R_j from subsume region $(SR)_{J_i}$ will be considered as near the optimal resource included in the set $(hs\_SR)_{J_i}$, if similarity measures $\cos\left(J_i, R_j\right) \ge threshold_{SR}$. Similarly, a resource R_j from the plug-in region $(PR)_{J_i}$ will be considered as near the optimal resource included in the set $(hs\_PR)_{J_i}$, if similarity measures $\cos\left(J_i, R_j\right) \ge threshold_{PR}$. The symbols used in this paper and their descriptions are shown in Table 1.

Algorithm 1: Pseudo Code for Semantic Matchmaking Process

Input: List of Resources R

Output: Three regions, Exact Region $(ER)_{J_i}$, High similarity Subsume Region $(hs\_SR)_{J_i}$, High similarity Plug-in Region $(hs\_PR)_{J_i}$

1. Initialize $(ER)_{J_i} \leftarrow \{\ \}, (SR)_{J_i} \leftarrow \{\ \}, (PR)_{J_i} \leftarrow \{\ \}, Sim\_mat_{1*\left|(SR)_{J_i}\right|} \leftarrow \{\ \}$, $(hs\_SR)_{J_i} \leftarrow \{\ \}$, $Sim\_mat_{1*\left|(PR)_{J_i}\right|} \leftarrow \{\ \}$, $(hs\_PR)_{J_i} \leftarrow \{\ \}$
2. For $\forall\ R_j \in R$ where $1 \le j \le M$
3. If $\mathrm{Re}q\left(J_i\right) \equiv Avail\left(R_j\right)$

4. $(ER)_{J_i} \leftarrow (ER)_{J_i} \cup \{R_j\}$
5. End
6. If $\mathrm{Re}\,q(J_i) < Avail(R_j)$
7. $(SR)_{J_i} \leftarrow (SR)_{J_i} \cup \{R_j\}$
8. End
9. Else
10. $(PR)_{J_i} \leftarrow (PR)_{J_i} \cup \{R_j\}$
11. End
12. End
13. For $\forall R_j \in (SR)_{J_i}$
14. $Sim\_mat_{1,k} \leftarrow d(J_i, R_j) \leftarrow \cos(J_i, R_j) \leftarrow \frac{(J_i . R_j)}{\|J_i\| \|R_j\|}$ Where $1 \le k \le |(SR)_{J_i}|$
15. End
16. If $Sim\_mat_{1,k} \ge threshold_{SR}$ then
17. $(hs\_SR)_{J_i} \leftarrow (hs\_SR)_{J_i} \cup \{R_j\}$ Where $R_j \in (SR)_{J_i}$
18. End
19. For each resource $R_j \in (PR)_{J_i}$
20. $Sim\_mat_{1,k} \leftarrow d(J_i, R_j) \leftarrow \cos(J_i, R_j) \leftarrow \frac{(J_i . R_j)}{\|J_i\| \|R_j\|}$ Where $1 \le k \le |(PR)_{J_i}|$
21. End
22. If $Sim\_mat_{1,k} \ge threshold_{PR}$ then
23. $(hs\_PR)_{J_i} \leftarrow (hs\_PR)_{J_i} \cup \{R_j\}$ Where $R_j \in (PR)_{J_i}$
24. End

The various cases using scheduling, identified by the resource broker for virtual resource creation and application execution, are given below:

Use Case 1: In this use case, the Algernon query will determine if the existing grid has the required application execution environment and operating system to run the application. Then, it will check that the grid resources satisfy the user hardware requirements such as node count, processor speed, ram speed, etc. If the existing physical Grid resource satisfies the job requirements, it is classified into a scheduling category called "Physical".-Here, the job is submitted only to the grid resources without creating any virtual resources, and the jobs may be sequential or parallel in nature. A sample query for identifying the "Physical" scheduling scenario is given below.

Query 1: String Physical Query = "((: instance HeadNode? Head) (HeadNodeApp? Head? App) (hasApplications? App? job.getApplicationName ( )) (HeadNodeOS? Head? HeadOS) (hasOSVersion? HeadOS? OS) (hasOSVersion? HeadOS? job.getOsName ( ) (hasFreeNodes? Head? Nodes) (: TEST (: LISP (>=? Nodes "+job.getNodeCount ( ) + "))))"

Use Case 2: Here, the existing grid resources satisfy the user's application requirements, but the number of required nodes are not available in a single grid resource. For example, if the user's application requires five nodes and the existing cluster (physical grid resource) has 10 nodes, but eight have been already engaged with another job, then there is a shortage of three nodes to run this application request. However, it is possible to create three nodes as virtual machines in the grid or cloud resources. Once a virtual resource is created, it is attached to the existing physical grid cluster using the hostname and IP address assigned to it. Moreover, it is automatically configured with Beowulf Cluster configuration of Domain Name Service (DNS), Network File System (NFS) and Network Information Service (NIS). Once the configuration is completed, it is possible to run

the job in a coordinated manner. This category of scheduling is classified as a “hybrid” scheduling scenario. The job scheduled to the resources, which has both physical nodes, and virtual nodes attached to physical nodes, is shown in Figure 5. The sample query for handling the "hybrid" scheduling scenario is given below. It first identifies the head node of the required operating system image using hasImageName property. Then, it verifies whether the operating system has been bundled with the required application support using hasApplicationSupport property. The images may be available in grid or cloud resources. Once the required images' location is identified the query to find the hardware capability is executed on the ontology knowledge base for virtual resource creation.

Query 2: String Hybrid Query = "((: instance HeadNode? Head1) (HeadNodeImage? Head1? HeadnodeImage) (hasImageName? HeadnodeImage? ImageName) (hasImageLocation? HeadnodeImage? hasImageLocation) (hasApplicationSupport? ImageName? \""+job.getApplicationName ( ) +"\")) ";

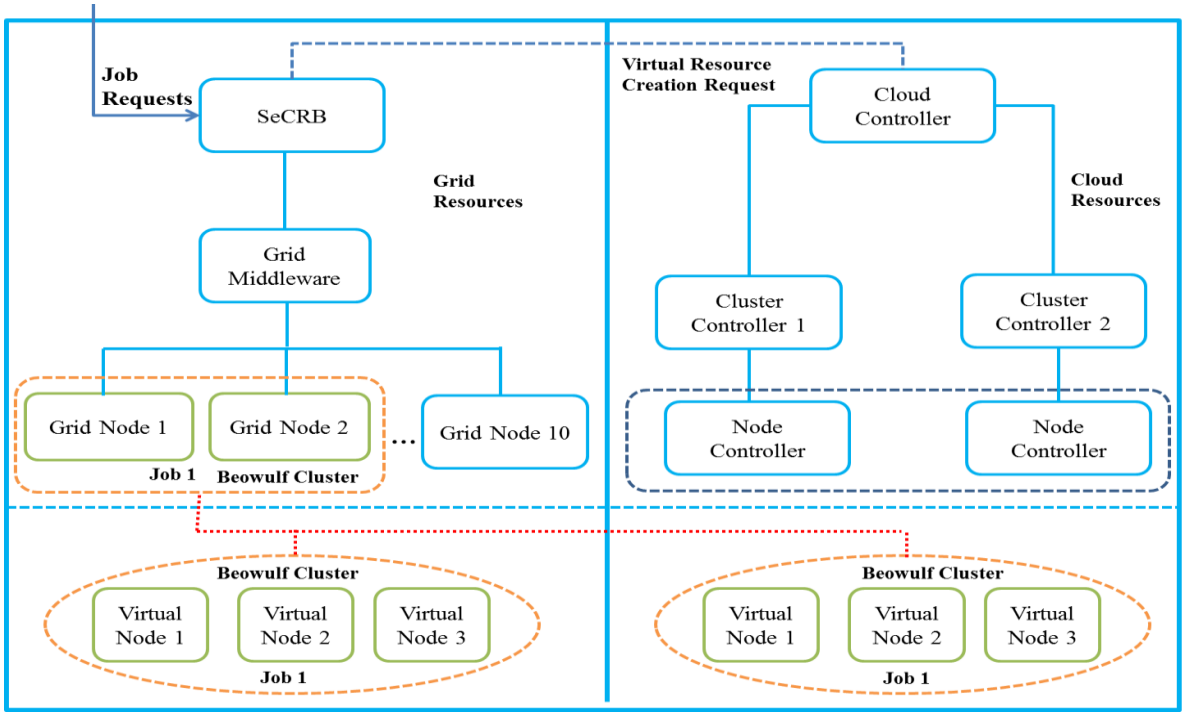

Figure 5: Hybrid Scheduling Scenario

Use Case 3: The existing grid resources do not provide the required user operating system, or the execution environment to support the application. It is classified as “Virtual Cluster” scheduling scenario. It is the creation of a virtual cluster in the grid or cloud resources, in an on-demand manner, and the submission of parallel jobs in the newly created virtual cluster. The query to identify the "virtual cluster" scheduling scenario is given below. It first determines if the head node has the required operating system image and the required application support. The images may be available in the grid or cloud resources. Once the location of the required operating system image is identified, the query to identify the hardware capability is executed on the ontology knowledge base to find the potential resources for virtual resource creation.

Query 3: String Virtual Cluster Query = “((: instance HeadNode? Head) (HeadNodeApp? Head? App) (hasApplicationSupport? App \""+job.getApplicationName ( ))”;

Case 4: Here, the user is interested in leasing resources in the form of virtual machines or virtual clusters without any application requirements. The broker creates the virtual resources and the required operating system, and provides access to the user. The sample query for handling the "lease" scheduling scenario is given below. It identifies the head node of the required operating system image using hasImageName property but does not bother about the application execution environment. Similar to the hybrid and virtual cluster scheduling scenarios, the images may be available in the grid or cloud resources. Once the location of the required operating system images location is identified, the query to identify the hardware capability is executed on the ontology knowledge base to find the potential resources for virtual resource creation.

Query 4: String Lease Query ="((: instance HeadNode? Head) (HeadNodeImage? Head? Imagename))";

4.2. Job Handler and Resource Selection Management

The Job Handler and Resource Selection management is responsible for the following: retrieving job requests from the user, ordering jobs, negotiating with resource providers, selecting the resources for provision and the application execution from the grid or cloud resources. The four main services implemented in the job handler and resource selection management are as follows:

1. Request Handler Service
2. Queue Manager Service
3. SLA Negotiator Service
4. Scheduling Service

4.2.1 Request Handler Service

The Request Handler Service retrieves the user's application requirements as JSDL file. The JSDL file consists of hardware, software and QoS requirements. This service is responsible for parsing the user's application requirements, and the parsed user requirements are updated in the job pool. It maintains a list of dispatched and un-dispatched jobs in job queues. It periodically invokes the Semantic Description and Discovery Service (SDDS) to discover or match the resources for application execution and virtual resource creation. Once the matched resource is retrieved from the Semantic Description and Discovery Service that information is updated in the job pool. It invokes the SDDS every 2 minutes or when the maximum number of jobs waiting in the queue reaches 10. Once the resources are found, it sends the job id and matched resource list to the Queue Manager.

4.2.2. Queue Manager Service

The Queue Manager Service is responsible for retrieving the job requests from the Request Handler Service along with job and scheduling type. The jobs are classified as sequential, parallel or lease, and placed into the queue. The scheduling types may be Physical, Hybrid, Virtual Cluster Creation or Lease. The Queue Manager Service is implemented as a Factory/Instance Pattern. The Queue Manager Service invokes the Scheduler Thread in a periodic interval when scheduling the available jobs in the queue. The Queue Manager Factory Service invokes the Queue Manager Service for each request and creates the instance for each request. Once the job and resources are selected and dispatched, the jobs are removed from the job queue. It invokes the SLA negotiator service to find an acceptable resource list after negotiation. Finally, it calls the scheduling service to invoke the prioritization of job/application requests and the selection of suitable resources.

4.2.3. SLA Negotiator Service

There may be several resources appropriate for a single-user application request. The semantic discovery process returns several resources, which satisfy the user's application requirements. Among these resources, some of the best options need to be selected based on the Quality of Service (QoS) parameters. The broker representing the user match makes and negotiates among all the resources returned- and offers a set of resources that agree with the negotiated QoS. The best QoS value emerges during the course of negotiation and is the value that is mutually agreed on by both the broker and the provider after negotiation. The Service Level Agreement (SLA) template is established with all the resource providers that are returned after negotiation. Instantiation of the template is done after finalizing one provider during scheduling. The QoS parameters to be negotiated are pre-decided. Negotiation is done by giving proposals and counter-proposals. The broker starts by giving the initial proposal of a parameter value to all the resource providers. Each provider replies by generating a counter-proposal. The broker then offers its own counter-proposal to each provider. This process continues for several iterations until an agreement is reached with at least one of the providers. All the providers who reach an agreement with the broker are returned by the algorithm. The generation of counter-proposals is based on the utility values of parameters calculated by each provider. The utility value of a parameter shows the level of satisfaction of the entity calculating the utility on a parameter value. For example, if the deadline for the user's request is 20-Jun-2013 18:00 hours from today, the broker calculates utility for this value. Also, the resource provider calculates the utility for this value. The broker and the provider may come up with different utility values for 20-Jun-2013 18:00 hours that show the different levels of satisfaction of each provider when the deadline is 20-Jun-2013 18:00 hours. This is because the provider aspires to satisfy the request on or after the deadline and the broker aspires to satisfy the request on or before the deadline. A utility function calculates the utility of each parameter for each provider in the negotiation. The utility for a parameter y calculated by provider z is as follows:

$$\left(U_c^y\right)_{R_j} = \left\{ \begin{array}{ll} \dfrac{\left(P_{\max}^y\right)_{R_j} - \mathrm{P}_c^y}{\left(P_{\max}^y\right)_{R_j} - \left(P_{\min}^y\right)_{R_j}} & \textit{if} \quad \text{y is expected to be minimum} \\ \dfrac{\mathrm{P}_c^y - \left(P_{\min}^y\right)_{R_j}}{\left(P_{\max}^y\right)_{R_j} - \left(P_{\min}^y\right)_{R_j}} & \textit{if} \quad \text{y is expected to be maximum} \end{array} \right\} \qquad \text{- (1)}$$

In (1), $\left(P_{\max}^y\right)_{R_j}$ is the maximum value and $\left(P_{\min}^y\right)_{R_j}$ represents the minimum value of parameter y in the acceptance range of provider R_j. It is pre-decided by the provider, and P_c^y is the current value of a parameter. The total utility for each provider is calculated by considering weights w_y for each parameter y ranging from 1 to t. The weights are decided by the providers a priori. The total utility is calculated using Equation (2), and the negotiation algorithm implemented in SeCRB is given below in Algorithm 2.

$$\left(U_c\right)_{\left(R_j\right)} \leftarrow \sum_{y=1}^{t} \left(U_c^y\right)_{R_j} \times w_y \qquad \text{- (2)}$$

Algorithm 2: Negotiation ($\left(ER\right)_{J_i}$ $\left(hs\_SR\right)_{J_i}$, $\left(hs\_PR\right)_{J_i}$, Flag)

Input: $\left(ER\right)_{J_i}$, $\left(hs\_SR\right)_{J_i}$, $\left(hs\_PR\right)_{J_i}$

Output: Set of resource providers from $N_{(ER)_{J_i}}$ or $N_{(SR)_{J_i}}$ and $N_{(PR)_{J_i}}$ with whom negotiation is agreed

1. *CB* broadcasts P_{Init} to $\forall \left(R_j\right) \in \left(ER\right)_{J_i}$ and computes U_{Init}
2. For $\left(R_j\right) \in \left(ER\right)_{J_i}$ sends counter proposal $\left(P_c\right)_{\left(R_j\right)}$
3. $\left(No.proposal\right)_{ER} \leftarrow \left(No.proposal\right)_{ER} + 1$
4. End
5. Initialize $U_{(ER)_{J_i}} \leftarrow \{\ \}$, $N_{(ER)_{J_i}} \leftarrow \{\ \}$
6. For $\forall \left(R_j\right) \in \left(ER\right)_{J_i}$ sends counter proposal
7. $\left(U_c\right)_{\left(R_j\right)} \leftarrow \sum_{y=1}^{t} U_c^y \times w_y$
8. If $\left(U_c\right)_{\left(R_j\right)} > U_{Init}$ then
9. $U_{(ER)_{J_i}} \leftarrow U_{(ER)_{J_i}} \cup \left\{\left(U_c\right)_{\left(R_j\right)}\right\}$
10. $N_{(ER)_{J_i}} \leftarrow N_{(ER)_{J_i}} \cup \left\{\left(R_j\right)\right\}$
11. Flag = 1;
12. return;
13. End
14. End
15. If $U_{(ER)_{J_i}} \equiv \varnothing$ then
16. *CB* Broadcasts P_{Init} to $\forall \left(R_j\right) \in \left(hs\_SR\right)_{J_i}$ and $\forall \left(R_j\right) \in \left(hs\_PR\right)_{J_i}$
17. For $\left(R_j\right) \in \left(hs\_SR\right)_{J_i}$ sends counter proposal $\left(P_c\right)_{\left(R_j\right)}$
18. $\left(No.proposal\right)_{SR} \leftarrow \left(No.proposal\right)_{SR} + 1$
19. End
20. For $\left(R_j\right) \in \left(hs\_PR\right)_{J_i}$ sends counter proposal $\left(P_c\right)_{\left(R_j\right)}$
21. $\left(No.proposal\right)_{PR} \leftarrow \left(No.proposal\right)_{PR} + 1$

22. End
23. Initialize $U_{(SR)_{J_i}} \leftarrow \{\ \}$, $N_{(SR)_{J_i}} \leftarrow \{\ \}$, $U_{(PR)_{J_i}} \leftarrow \{\ \}$, $N_{(PR)_{J_i}} \leftarrow \{\ \}$
24. For $\forall(R_j) \in (hs\_SR)_{J_i}$ send the proposal counter in subsume region,
25. $(U_c)_{(R_j)} \leftarrow \sum_{y=1}^{t} U_c^y \times w_y$
26. If $(U_c)_{(R_j)} > U_{Init}$ *then*
27. $U_{(SR)_{J_i}} \leftarrow U_{(SR)_{J_i}} \cup \{(U_c)_{(R_j)}\}$
28. $N_{(SR)_{J_i}} \leftarrow N_{(SR)_{J_i}} \cup \{(R_j)\}$
29. End
30. End
31. If $U_{(SR)_{J_i}} \equiv \varnothing$
32. $P_{Avg} \leftarrow Avg\left((P_c)_{R_j}\right)$ Where $(R_j) \in (hs\_SR)_{J_i}$
33. For $\forall(R_j) \in (hs\_SR)_{J_i}$
34. If $(P_c)_{(R_j)} \geq P_{Avg}$
35. $U_{(SR)_{J_i}} \leftarrow U_{(SR)_{J_i}} \cup \{(U_c)_{(R_j)}\}$
36. $N_{(SR)_{J_i}} \leftarrow N_{(SR)_{J_i}} \cup \{(R_j)\}$
37. End
38. End
39. End
40. For $\forall(R_j) \in (hs\_PR)_{J_i}$ send the proposal counter in plug-in region,
41. $(U_c)_{(R_j)} \leftarrow \sum_{y=1}^{t} U_c^y \times w_y$
42. If $(U_c)_{(R_j)} > U_{Init}$ *then*
43. $U_{(PR)_{J_i}} \leftarrow U_{(PR)_{J_i}} \cup \{(U_c)_{(R_j)}\}$
44. $N_{(PR)_{J_i}} \leftarrow N_{(PR)_{J_i}} \cup \{(R_j)\}$
45. End
46. End
47. If $U_{(PR)_{J_i}} \equiv \varnothing$
48. $P_{Avg} \leftarrow Avg\left((P_c)_{R_j}\right)$ $(R_j) \in (hs\_PR)_{J_i}$
49. For $\forall(R_j) \in (PR)_{J_i}$
50. If $(P_c)_{(R_j)} \geq P_{Avg}$
51. $U_{(PR)_{J_i}} \leftarrow U_{(PR)_{J_i}} \cup \{(U_c)_{(R_j)}\}$
52. $N_{(PR)_{J_i}} \leftarrow N_{(PR)_{J_i}} \cup \{(R_j)\}$
53. End
54. End

55. End
56. End

4.2.4. Scheduling Service

The Scheduling Service prioritizes the user's application requests based on the jobs that are closer to the deadline. Once the jobs have been prioritized, it invokes the resource scheduler to select the near optimal resource from the negotiation accepted list. The resource scheduler is integrated with the Quality of Service (QoS)-Based Resource Scheduling (QBRS) algorithm. The QBRS algorithm selects the resources from the exact region only if the resources accept the utility values and satisfy the hardware, software and QoS requirements for a job execution. The pseudo code for the selection of resources, which may fall in an exact region, is given below in Algorithm 3.

Algorithm 3: Scheduling_Exact ($(ER)_{J_i}$, $N_{(ER)_{J_i}}$)

Input: $(ER)_{J_i}$, $N_{(ER)_{J_i}}$

Output: Single resource $R_j \in (ER)_{J_i}$ & $N_{(ER)_{J_i}}$

1. For $\forall R_j \in (ER)_{J_i}$
2. If $R_j \in N_{(ER)_{J_i}}$ && $(no\_Nodes_{R_j}) = (no\_Nodes_{J_i})$ then
3. return R_j
4. break;
5. End
6. End

The QBRS will select resources from the high-similarity subsume and high-similarity plug-in regions, if the resources available in the exact region do not accept the negotiation value. This selection will be based on the execution time of the jobs. The QBRS selects a resource from the high-similar subsume region, if the resource has accepted the negotiation that is present in the $N_{(SR)_{J_i}}$. The selection of resources for jobs in the high-similarity plug-in region involves two scenarios. Scenario 1 deals with a set of resources that perfectly match the software requirements of the job but lack the required nodes. The QBRS selects the resources that are present in $N_{(PR)_{J_i}}$. If the physical resources are not present in $N_{(PR)_{J_i}}$ it will select resources for virtual resources creation. In Scenario 2, if the software requirements of a job do not match the resources present in the high similarity plug-in region, a virtual cluster will be created. The scheduler selects the head node that has the maximum hard disk and RAM capacity. The compute nodes are created in other resources depending on its capability.

Algorithm 4: Scheduling_Subsume_Plugin ($(hs\_SR)_{J_i}$, $N_{(SR)_{J_i}}$, $(hs\_PR)_{J_i}$, $N_{(PR)_{J_i}}$, Fp)

Input: $(hs\_SR)_{J_i}$, $N_{(SR)_{J_i}}$ $(hs\_PR)_{J_i}$, $N_{(PR)_{J_i}}$

Output: Single resource $R_j \in (Q\_SR)_{J_i}$ & $N_{(SR)_{J_i}}$ or $\left(R_{(PR)}\right)_{J_i}$ consists of set of resources $\{R_1, ..R_w\}$ where $1 \le w \le \left| N_{(PR)_{J_i}} \right|$

1. $\left(R_{(PR)}\right)_{J_i} = \{\ \}$
2. For $\forall R_j \in (hs\_SR)_{J_i}$
3. If $R_j \in N_{(SR)_{J_i}}$ then
4. Choose R_j
5. break;
6. End
7. End
8. For $\forall R_v \in (hs\_PR)_{J_i}$

9. If $(Sw)_{J_i} \equiv (Sw)_{R_v}$ then
10. If $R_v \in N_{(PR)_{J_i}}$ && $(No.Nodes)_{J_i} \neq 0$ then
11. $(No.Nodes)_{J_i} \leftarrow (No.Nodes)_{J_i} - (No.Nodes)_{R_j}$
12. $\left(R_{(PR)}\right)_{J_i} \leftarrow \left(R_{(PR)}\right)_{J_i} \cup \{R_v\}$
13. Fp = 1;
14. End
15. End
16. End
17. If (Fp == 1)
18. If $(No.Nodes)_{J_i} \neq 0$ && $(No.Nodes)_{\forall R_v \in (hs\_PR)_{J_i}} = 0$ then
19. For $\forall R_v \in (hs\_PR)_{J_i}$
20. If $(No.Nodes)_{J_i} \neq 0$ then
21. $R_{\max} \leftarrow Max(Hd\_Capacity \ \& \ Ram\_Capacity)_{\forall R_v \in (hs\_PR)_{J_i}}$
22. If $(Hd\_Capacity)_{R_{\max}} \geq \left((No.Nodes)_{J_i} * \left(Hd\_needed_{J_i}\right)\right)$ then
23. Create $\left((No.Nodes)_{J_i} - 1\right)$ Virtual machine in $R_{\max}$
24. $(R)_{vm} \leftarrow (R)_{vm} \cup \{R_{\max}\}$
25. If $R_{\max} \notin \left(R_{(PR)}\right)_{J_i}$ then
26. $\left(R_{(PR)}\right)_{J_i} \leftarrow \left(R_{(PR)}\right)_{J_i} \cup \{R_v\}$
27. End
28. $(No.Nodes)_{J_i} \leftarrow (No.Nodes)_{J_i} - \left((No.Nodes)_{J_i} - 1\right)$
29. End
30. End
31. End
32. End
33. End // if Fp == 1
34. Else If (Fp==0)
35. $R_{\max} \leftarrow Max(Hd\_Capacity \& Ram\_Capacity)_{\forall R_v \in (hs\_PR)_{J_i}}$
36. $HeaderNode \leftarrow R_{\max}$
37. For $\forall R_v \in (hs\_PR)_{J_i}$
38. If $R_v \in N_{(PR)_{J_i}}$ && $(No.Nodes)_{J_i} \neq 0$ then
39. If $(Hd\_Capacity)_{R_{\max}} \geq \left((No.Nodes)_{J_i} * \left(Hd\_needed_{J_i}\right)\right)$ then
40. Create $\left((No.Nodes)_{J_i} - 1\right)$ Virtual machine in $R_{\max}$
41. $(R)_{vm} \leftarrow (R)_{vm} \cup \{R_{\max}\}$
42. If $R_{\max} \notin \left(R_{(PR)}\right)_{J_i}$ then
43. $\left(R_{(PR)}\right)_{J_i} \leftarrow \left(R_{(PR)}\right)_{J_i} \cup \{R_v\}$
44. End
45. $(No.Nodes)_{J_i} \leftarrow (No.Nodes)_{J_i} - \left((No.Nodes)_{J_i} - 1\right)$

46. End
47. End
48. End
49. End
50. For $\forall R_v \in \left(R_{(PR)}\right)_{J_i}$ where $1 \le v \le \left|\left(R_{(PR)}\right)_{J_i}\right|$
51. $TET \leftarrow TET + \left(Extime\right)_{R_v}$
52. End
53. If $TET < \left(Extime\right)_{R_j}$ then
54. Call Middleware Adapter ($\left(R\right)_{vm}$);
55. return $\left(R_{(PR)}\right)_{J_i}$
56. End
57. Else
58. return R_j
59. End

The pseudo code for complete resource brokering progression is shown in Algorithm 5. It first invokes the matchmaking algorithm. Next, it invokes the negotiation and the resource selection algorithms, and finally it invokes the appropriate middleware adapters.

Algorithm 5: Resource Brokering Algorithm

Input: Number of Schedule *NS*
Output: Optimal schedule in which jobs are allocated to resources

1. For $\forall$ *Schedule S* where $1 \le S \le NS$
2. For $\forall$ *Job* J_i where $1 \le i \le N$
3. Initialize the values of Flag = 0, Fp = 0;
4. Matchmaking (R)
5. Negotiation ($\left(ER\right)_{J_i}$, $\left(hs\_SR\right)_{J_i}$, $\left(hs\_PR\right)_{J_i}$, Flag)
6. If (Flag == 1) then
7. Scheduling_Exact ($\left(ER\right)_{J_i}$, $\left(N_{(ER)_{J_i}}\right)_{CB}$)
8. End
9. Else
10. Scheduling_Subsume_Plugin ($\left(h\_SR\right)_{J_i}$, $N_{(SR)_{J_i}}$, $\left(hs\_PR\right)_{J_i}$, $N_{(PR)_{J_i}}$,Fp)
11. End
12. End // end of all jobs in a schedule
13. End // end of all schedule

4.3. Cloud Resource and Image Information Management

The description of information about cloud resources, and the details of the available operating system images and software are represented in the semantic knowledge base using ontology. The following two services have been introduced for the purpose of resource and image information management.

1. Cloud Information Manager Service (CIMS)
2. Image Metadata Manager Service (IMMS)

4.3.1. Cloud Information Manager Service (CIMS)

The architecture of CIMS and its components are shown in Figure 6. The Cloud Resource Information Updater (CRIU) component periodically accesses the Cloud Middleware Adapter (CMA) and retrieves the Cloud resource information. The retrieved information is represented in an XML-format in Cloud Resource Repository (CRR). The Cloud Resource Information Parser (CRIP) component parses the cloud resource information available in CRR. The parsed information is updated and represented in the Knowledge Base (KB) using the semantic description module. The sample details of the cloud resource's information represented in CRR is shown in Table 3.

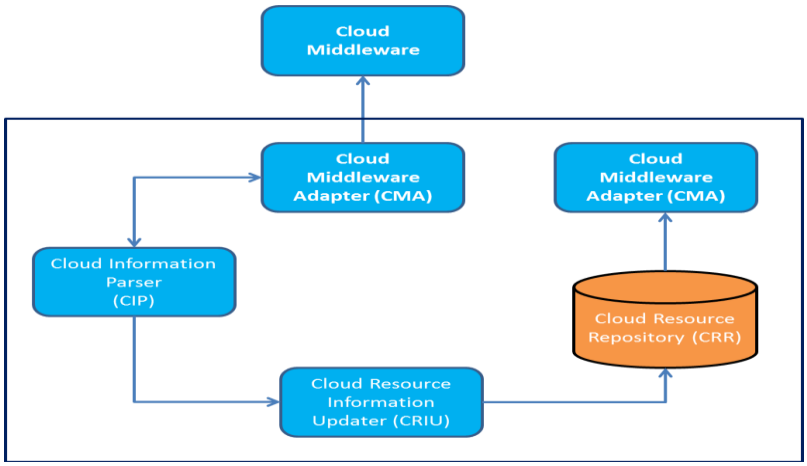

Figure 6: Cloud Information Manager Service

4.3.2. Image Metadata Manager Service (IMMS)

It is essential to know the details of the existing operating system images and the software required to execute an application. Hence, we have introduced Image Metadata Manager Service (IMMS) in SCRB. The architecture of IMMS and its interactions with other components is shown in Figure 7. The main purpose of IMMS is to identify the operating system images bundled with the software required for the execution of an application. In Eucalyptus, a single operating system image is divided into three types of images namely, root, kernel and ramdisk. The images are uploaded to cloud resources through CMA. The images are bundled, encrypted, registered and uploaded in the Cloud resources. The sample metadata that generated the resource with Fedora Core 5.3 as its operating system and bundled the software for running GROningen MAchine for Chemical Simulation (GROMACS) Application, and generated the URI and the cloud controller hostname are represented in Table 4. For example, the image of the Red Hat Enterprise Linux operating system with Fast Fourier Transfer Transform (FFT) [57] and Message Passing Interface (MPI) [58] software libraries are required to run the GROMACS [59] application. IMMS retrieves from the Request Handler Service a request to upload the operating system images and the required software in the cloud infrastructure. The Image Metadata Creation and Parser (IMCP) module creates the metadata for the available OperatingSystemImages and the software libraries as shown in Table 4. The operating system images are uploaded through CMA, and the Eucalyptus cloud middleware generates unique URI's for root, kernel and ramdisk. Once the operating system images are uploaded in the cloud resources, image metadata is updated through the Image Metadata Updater (IMU) and the IMCP module. The image metadata information is updated in the XML-based Operating System (OS) Image Metadata Repository (OSIMR). The generated unique URIs is updated within the ImageURI tag.

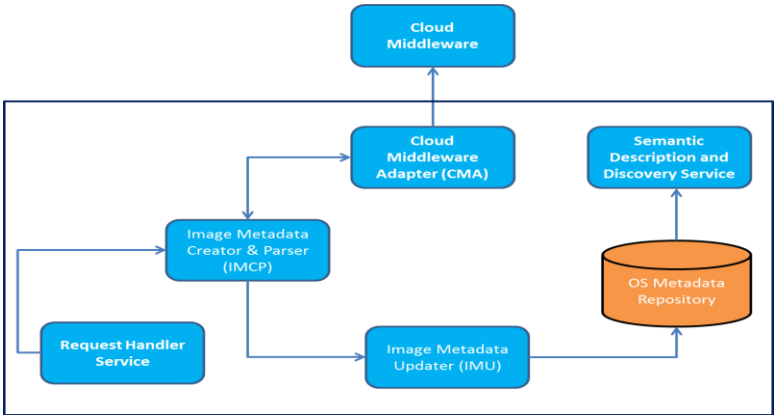

Figure 7: Image Metadata Manager Service

4.4. Controller and Adapter Management

The controller and adapter management is responsible for connecting the grid and cloud resources that are responsible for invoking the appropriate services based on the needs. It consists of the three major components represented in Figure 8.

1. Grid Middleware Adapter
2. Cloud Middleware Adapter

3. Action Manager Service

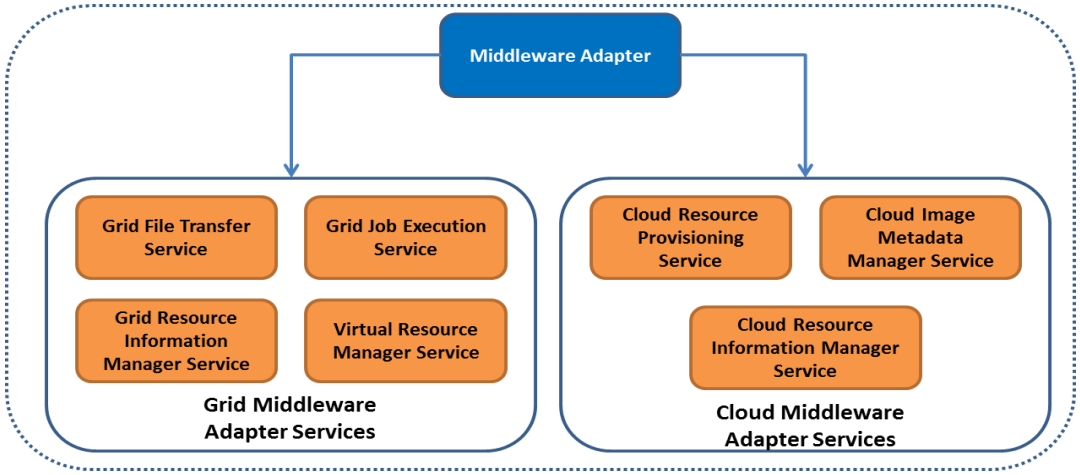

Figure 8: Middleware Adapters

4.4.1. Grid Middleware Adapter (GMA)

The Grid Middleware Adapter (GMA) is responsible for aggregating the grid resources, transferring the executable input and output files, executing and monitoring jobs and creating and deleting virtual resources. It uses the Globus API's for interacting with the grid middleware and performing the various operations ~~as~~ stated above.

A. Grid File Transfer Manager Service (GFTMS) – This service is implemented as a transfer manager factory service and a transfer manager service, and deployed in the resource broker. It uses GridFTP and Reliable File Transfer (RFT) protocol for transferring files. First, creates the remote job directory in the selected resource then it transfers the executable input files to the remote directory. Once the job is successfully completed, the output files are transferred to the resource broker. It acts as a wrapper for the GridFTP and the Reliable File Transfer (RFT) services. Finally, it sends the success/failure message to the Action Manager Component after the file transfer.

B. Grid Job Execution Manager Service (GJEMS) - It uses the Job Managed Factory service in the Globus Toolkit to submit, monitor and retrieve the job status. It is implemented as an execution manager factory service and an execution manager service, and deployed in the resource broker. It provides an abstraction from the resource management layer available in the grid resources. It receives the job id as an input, and accesses the request handler to retrieve the job executables, arguments, input files, selected execution host and LRMS type. Then, it generates the Resource Specification Language (RSL) file for the job request, and the generated RSL is maintained in the job directory of the resource broker node. The RSL file contains the job parameters required to submit the job for GRAM in the grid resources. It invokes the WS-GRAM client to submit the RSL file. The WS-GRAM client interfaces with the Job Managed Factory service of the selected execution host, submits the job for execution and monitors the status of the job execution. Once the job has been executed successfully, it receives the status message as "DONE". Finally, it updates the status message of the Action Manager Service. The WSGRAM client needs an argument such as execution hostname, type of LRMS such as PBS, SGE, Condor, LSF or Fork, RSL file and job id.

C. Grid Resource Information Manager Service (GRIMS) – It is deployed at the broker level. It periodically accesses the Monitoring and Discovery Service (MDS), which is running in grid middleware. It retrieves static as well as dynamic information. Some of the static information retrieved from grid resources is include processor speed, operating system, kernel version, etc. The dynamic information retrieved from grid resources are bandwidth, latency, free ram memory, free hard disk memory, free nodes, etc. The aggregated resource information is parsed and the same is updated in the Grid Resource Repository.

D. Virtual Resource Manager Service – It manages the life cycle of virtual resources including creation, deletion, etc. It interfaces with the Virtual Manager Service, which we developed, and deploys in the grid resources that interact with the hypervisor.

4.4.2. Cloud Middleware Adapter (CMA)

Similar to the grid middleware adapter, the Cloud Middleware Adapter (CMA) is responsible for interacting with the cloud middleware that is deployed in the cloud resources. It aggregates the cloud resource information, provides the virtual resources and manages the uploaded operating system images in the cloud resources.

A. Cloud Resource Information Service – It is deployed at the broker level. Cloud Controller (CLC) is the entry point for the Eucalyptus cloud middleware. This service interfaces CLC to fetch the static and dynamic information about physical cloud resources. The aggregated information is updated in the Cloud Resource Repository. Later on, this information is updated in the ontology knowledge base.

B. Cloud Resource Provisioning Service – This service is deployed at the broker level. It invokes CLC for virtual instance creation, deletion and monitoring.

C. C. **Cloud Image Metadata Manager Service** – Eucalyptus cloud middleware requires three types of images namely, kernel, ramdisk and root for virtual instance creation. It generates unique URIs for each image. It makes use of CLC to upload the operating system images installed and the required software libraries, and retrieve the generated unique URIs from the cloud controller service.

4.5. Action Manager Service – The Action Manager Service acts as a mediator between Queue Manager Service, Grid Middleware Adapter (GMA), and Cloud Middleware Adapter (CMA). If the selection of resource is in the grid environment, the Action Manager Service invokes the GMA to perform the operations shown in Figure 9. Else, if the selection of resources is in the cloud environment, the Action Manager Service invokes the CMA to perform the operations shown in Figure 10.

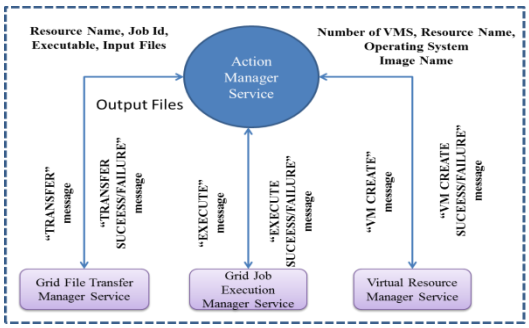

Figure 9: Action Manager with GMA

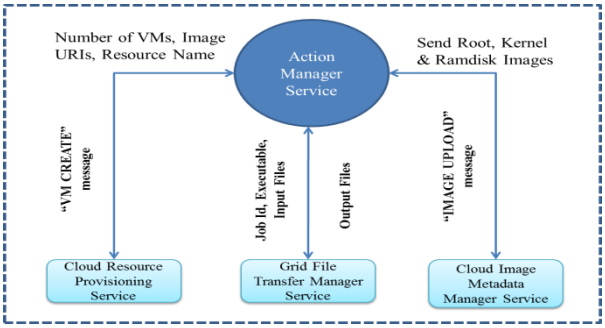

Figure 10: Action Manager with CMA

5. Working Principle of Semantic-enabled CRB (SeCRB)

The working principle of semantic-based resource management for managing both the grid and cloud resources is described below.

1. The user submits an application request through the Graphical User Interface (GUI) or Job Service Description Language (JSDL) to SeCRB.
2. The Request Handler Service retrieves the job request that may consist of hardware, software and Quality of Service (QoS) requirements. It parses the job requirements and the parsed requirements are stored in job object.
3. The Request Handler Service invokes the Semantic Description and Discovery Service to match-make the user's application requests and the available information about the grid and cloud resources.
4. The Semantic Description and Discovery Service finds out suitable and potential grid and cloud resources that may fall into the exact, subsume and plug-in regions for virtual resource creation and application execution.
5. The Queue Manager Service maintains the incoming requests in the job queue and invokes the appropriate services based on the type of job.
6. The scheduling Service is responsible for selecting or prioritizing the user application requests, and selecting the best resource for a virtual resource creation for the hybrid, virtual cluster creation and lease scheduling scenarios. The scheduling service is implemented with resource selection algorithms for selecting the resources from the exact, subsume and plug-in regions in an optimal manner.
7. The Queue Manager Service invokes the SLA Negotiator Service to pair the user application requirements with the matched grid and cloud resources, and retrieves the resources that satisfy the Service Level Objectives (SLOs) of deadline and response time specified by the user.
8. The SLA Negotiator Service first initiates the user application requirements and resources that may fall in the exact region. If the resources that fall into the exact region are not satisfactory, it negotiates the resources to fall in the subsume and plug-in regions.
9. Finally, the Queue Manager Service sends the job object, selected resources and type of resources such as grid or cloud to the Action Manager Service.
10. The Action Manager Service invokes the GMA if the selected resource is in grid else, it invokes CMA if the selected resource is in cloud.

11. The Grid Middleware Adapter (GMA) acts as a port for interfacing the Globus grid middleware to update the grid resource information, transfer files, execute and monitor jobs, and create, delete and update virtual resources.
12. The Cloud Middleware Adapter (CMA) acts as a port for interfacing with the Eucalyptus Cloud Middleware. It is responsible for the following operations that are shown in Figure 11.
    - Virtual Instance Creation
    - Virtual Instance Deletion
    - Cloud Resource Information Updation
    - Image Metadata Updation

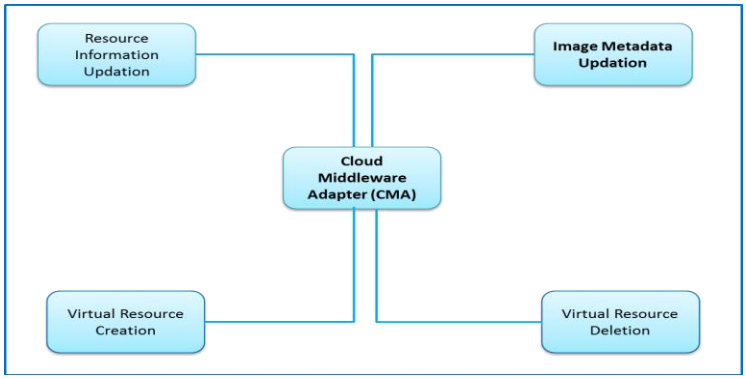

Figure 11: Functionalities of Cloud Middleware Adapter (CMA)

6. Experimental Setup, Operating System Images Preparation and Application Execution

To evaluate the efficiency of our proposed SeCRB – for running High Performance Computing (HPC) applications on the grid and cloud resources, we have processed a real case study for solving the well-known protein analysis problem GROMACS. Generally, the experimental methodology is a two-fold process. First, we have executed the GROMACS application in real grid and cloud resources, which allowed us to gather real traces of job data such as job execution time. Second, we have used the real traces of gathered job data for a simulation using the discrete event simulation method.

6.1 Experimental Setup

To evaluate the performance of the proposed work, we used the sixty resources that are available in the High-Performance Computing (HPC) laboratory and CARE research laboratory in our Anna University Campus. The experimental setup shown in Figure 12 was made on our campus. It consists of three grid resources namely, carecluster.care.mit.in, xencluster.care.mit.in and centcluster.care.mit.in. Each grid resource has one head node and 20, 10, 10 compute nodes respectively. The grid middleware of Globus Toolkit 4.0.7 is used for managing the Grid resources. The Sun Grid Engine (SGE) and the Portable Batch System (PBS) are configured as local resource managers. The grid resources are configured with Beowulf's cluster configuration. We have installed Red Hat Enterprise Linux 5 (RHEL5) and Cent OS 5.5 as operating systems, and Xen-3.2.3 and Kernel Virtual Machine (KVM) (kvm-36) as hypervisors in grid resources. The head node and compute nodes of grid resources has 2 GB RAM, 160 GB hard disk and 3200 MHz processor speed. We have configured two cloud cluster resources namely eucacluster1.care.mit.in and eucacluster2.care.mit.in. The Eucalyptus version 2.0.1 is used as cloud middleware for managing the cloud clusters. Each cloud cluster is configured with 10 node controllers and in total we have configured 20 node controllers for virtual instance creation. The gmetad daemon is configured in the cluster controllers, and gmond daemon is configured in the node controllers. These two daemons are responsible for retrieving the CPU-related information of total memory, free memory, processor speed and etc. The SeCRB is developed as Grid Services, which are deployed in the Globus Toolkit-4.0.7, and installed on a Dell server that consists of 4 CPUs with 2000 GHz per processor (4 CPUs X 4 Cores), 16 GB RAM and RHEL-5.0 as operating system. The Protégé Editor 3.4 is used for creating the ontology knowledge base, and the Algernon 5.0.1 is used as an inference engine. The Graphical User Interface (GUI) is running in a SeCRB server, which was developed using Java Server Faces and NetBeans IDE 6.7. The GUI has the provision for registering the grid and cloud resources as well registering the user details.

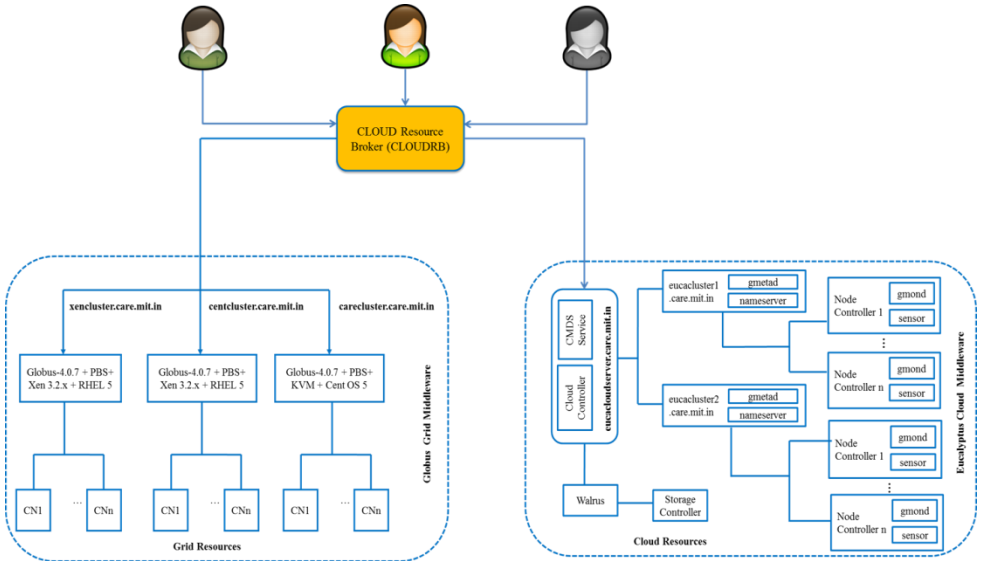

Figure 12: Experimental Setup

6.2 Operating System Images Preparation

We have prepared nine types of Linux operating system images that are listed below. Each Linux operating system image is classified into two types. The first one is grid enabled and is installed with Globus Toolkit using GTAI [59] software and Message Passing Interface (MPI) library. The other one is not grid enabled. Both types of images have been bundled with Fast Fourier Transform (FFT) and MPI library to run the GROMACS application. In addition, images have been configured with Beowulf Cluster configuration of head node, compute node, etc. For Beowulf cluster configuration, we have automated the process of Domain Name Server (DNS), Network File Sharing (NFS) and Network Information Service (NIS). The prepared images have been uploaded in the grid and cloud resources.

1. Red Hat Enterprise Linux Server release 5
2. Red Hat Enterprise Linux Server release 5.2
3. Fedora release 11
4. Fedora Core release 4
5. CentOS release 5.3
6. CentOS release 5.4
7. Scientific Linux CERN Release 3.0.5
8. Scientific Linux CERN SLC release 4.5
9. Scientific Linux SL release 5.0

6.3 GROMACS Application Execution

GROningen MAchine for Chemical Simulation (GROMACS) is a very powerful toolbox in modern molecular modeling. It was developed at the University of Groningen, which was built mainly for testing biochemical molecules like lipids, proteins and nucleic acids, and for understanding the structure, dynamics and motion of individual atoms. The two most commonly used methods are energy minimization and molecular dynamics that optimize the structure and simulate the natural motion of biological macromolecules. It converts molecular coordinates from a Protein Data Bank (PDB) file into its internal format. The GROMACS simulation process first generates the configuration file. The actual simulation process starts once the configuration file is created — producing a trajectory file describing the movements of atoms over time. The generated trajectory file is analyzed and visualized later to understand the structure of the protein. To run the simulation process in grid and cloud resources, we have automated the above seven steps as a shell script. It is defined as an executable in Job Service Description Language (JSDL). The Gromacs parallel MPI application is tested in SeCRB bacteriophage T4 Lysozyme Protein Database (10MB.pdb) [61] structure. The whole simulation process consists of seven steps as given below:

1. Creation of topology files using Protein Data Bank (PDB) file
2. Solvating the protein
3. Energy Minimization
4. Equilibration run
5. Performing MD simulation
6. Viewing Trajectory files
7. Viewing MD simulation results

The execution of an application requires the protein files and the input files as shown in Table 5 when analyzing the structure of the protein. The execution time of the application mainly depends on the number of steps in the input files. We have considered this application for testing our proposed approach by generating the virtual instances in the grid as well as in the cloud resources to complete the application within a minimal time. The SeCRB has been tested with five job requests and each request is varied with the required number of nodes and number of steps. The number of steps in the input files plays a major role in deriving the accurate results about molecule structures. We have pre-configured the VM images with the application required libraries and software environment. These images have been stored in the grid and cloud resources. The metadata information is updated for the uploaded images. The first step in SeCRB is generating job requests in JSDL file and submitting the job requests through Graphical User Interface (GUI). We have conducted this experiment in two iterations to find out the consistency in application execution time. In addition, if we increase the number of steps in the input files, the execution time of applications will obviously increase as well. Hence, we have increased the number of nodes from 10 to 40 as an application requirement that reduces the execution time of applications. We have collected the real traces of the applications' booting time, transfer time and execution time by varying the number of nodes and steps. The execution time of the applications has been reduced significantly using our proposed SeCRB by increasing the number of virtual instances as shown in Figure 13. The application execution is calculated using equation (3). It includes the file transfer time of input files, the booting time of virtual instances and actual running time of jobs. The Grid middleware in CRB directly invokes the Xen libraries to instantiate the virtual machine creation. However, the virtual instances creation request in cloud should go through three levels such as cloud controller, cluster controller and node controller that increase the boot time of the virtual instances. In addition, SeCRB transfer the input files to the newly created cluster head that represents the transfer time of the applications. However, the conventional CRB has the master image for the applications to execute. The conventional CRB bundle the images with required software libraries and transfer the image to the scheduled grid resources. The conventional CRB does not consider the boot time, transfer time for the execution time calculation. In addition to that, SeCRB upload the operating system images with required libraries in the Cloud environment. Moreover in SeCRB, we have not considered the semantic matchmaking time and SLA negotiation time during the execution time calculation. Even though, there is slight variation in the total execution time of applications in SeCRB, it increases the success rate and decreases the job rejection ratio of applications in a linear manner.

$$JCT_{J_i} = TT_{J_i} + BT_{J_i} + ART_{J_i} \quad (3)$$

Table 5: Protein Database, Input File Names and Number of Steps

Protein Database & Input File Names	Number of Steps
10MB.pdb (Bacteriophage T4 Lysozyme Protein Database)	-
em.mdp	400
pr.mdp	25000 - 125000 steps
run.mdp	50000 - 250000 steps

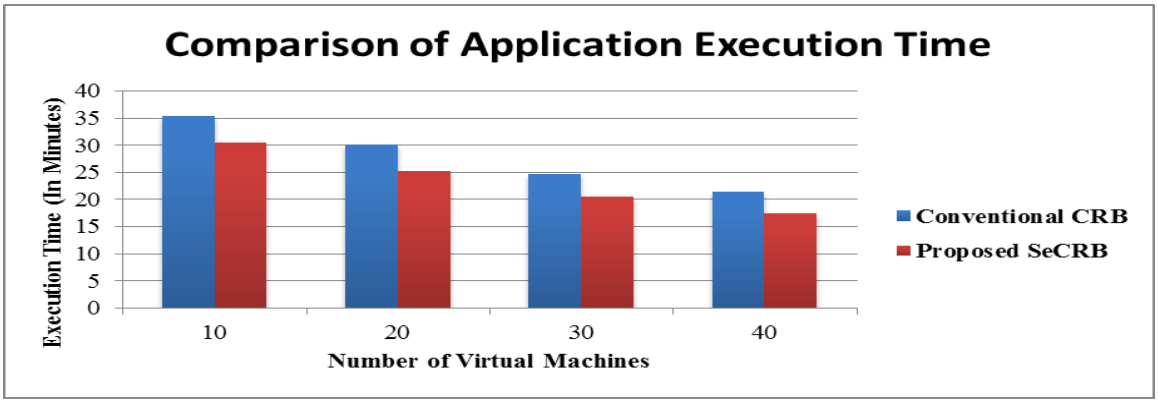

Figure 13: Comparison of Application Execution Time

This experiment is carried out to analyze the overhead imposed by running the same set of jobs in the same physical resource. Sometimes, the resource broker allocates the virtual instance creation in the same machine to reduce the job rejection. It is tested by running three jobs where each job requires 10 nodes. The execution time for the three jobs is represented in Figure 14. From the figure 14, it is evident that the workload of three virtual resources inside the same machine increases the execution time of a job in a linear manner. Once the simulation process is successfully completed, we use the Visual Molecular Dynamics (VMD) tool to analyse the results. The output file generated during the GROMACS simulation process is transferred to the user's site for analysis. The generated output file in our simulation process is run.trr. It is visualized using the VMD as shown in Figure 15.

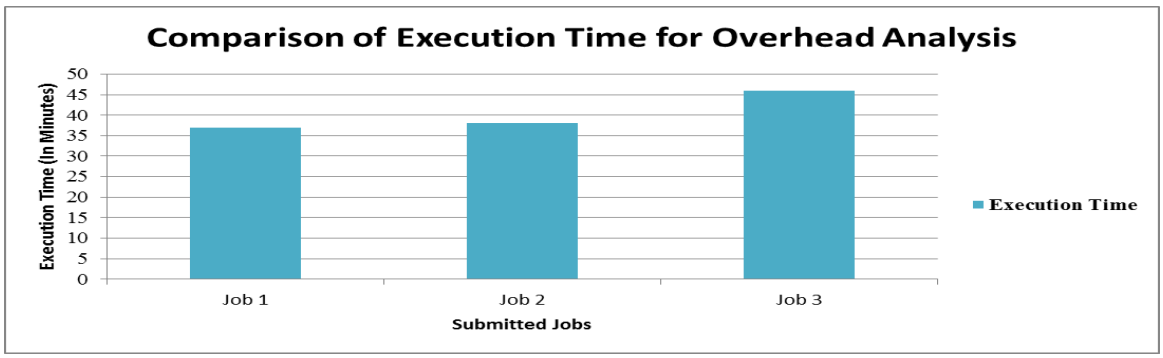

Figure 14: Application Execution for Overhead Analysis

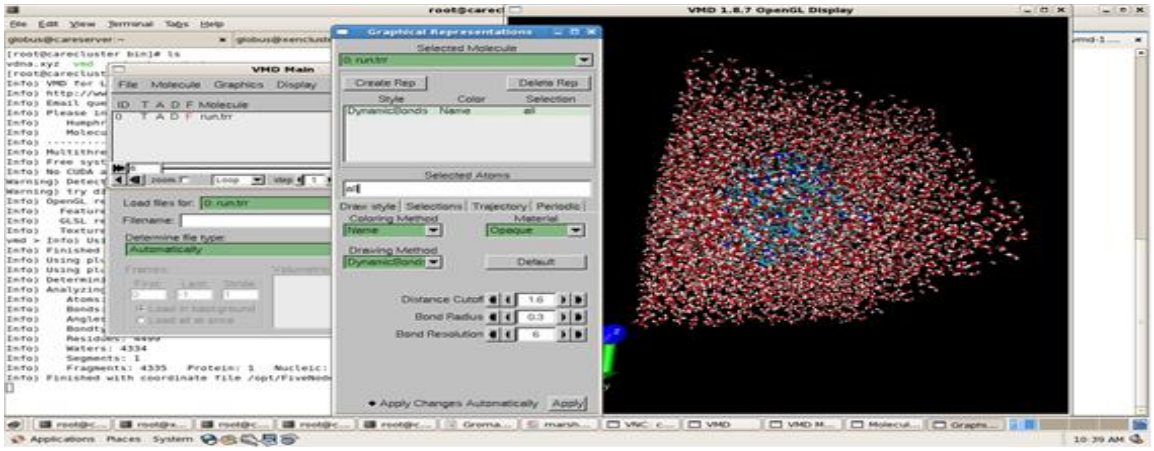

Figure 15: GROMACS Simulation Output

Similar to the previous two experiments, we have carried out this experiment to analyze the consistency of application execution time by varying the number of nodes from 10 – 40 for a job. This experiment is carried out using the same protein database and the required input files as discussed earlier. This simulation experiment is carried out by varying the number of nodes for a job and fixed the value of number of steps in pr.mdp and run.mdp. From the figure 16, it is evident that our proposed system maintains the consistency of application execution time for two iterations.

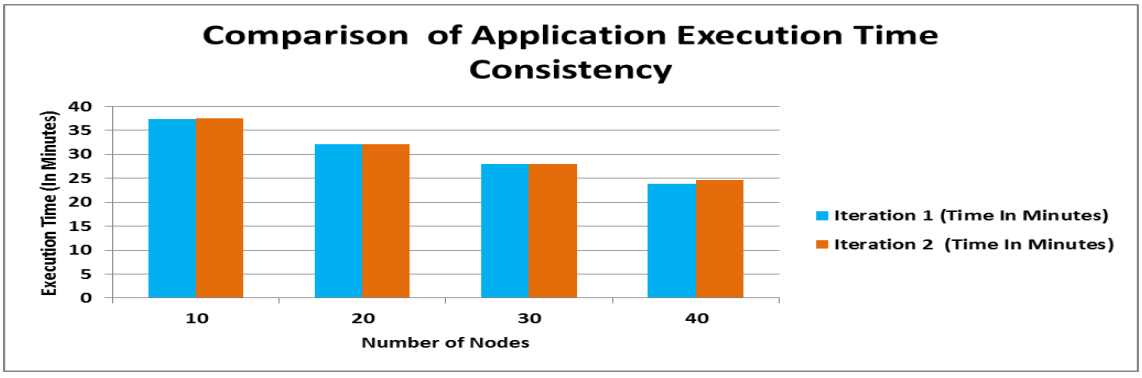

Figure 16: Comparison of Consistency of Application Execution Time

This experiment is carried out to analyze the throughput of jobs submitted per schedule in CRB versus SeCRB. We have generated five schedules and each schedule is varied with 10 – 50 short-term and long-term applications of Bio-Informatics and Image Processing applications in a random manner. We used the real grid and cloud resources that consist of totally 60 nodes. The generated job requests are submitted to resource broker and the comparative study is made. The number of jobs successfully completed within deadline versus time for CRB and SeCRB is tested in a real grid and cloud environment. The conventional CRB discovers the potential resources only from the grid environment. If the resources are not available in the grid environment for virtual instance creation, the jobs goes to waiting state in the resource broker queue, after sometime it is getting rejected. The conventional CRB submits the jobs only to the exact region. However, the proposed work discovers the potential resources from both grid and cloud environment. Nevertheless, the proposed SeCRB discovers the potential resources from subsume as well as plugin region. After the resource match making process, SLA negotiation brings the suitable resources from the exact, plugin and subsumes regions from both the grid and cloud resources that guarantee to satisfy the user QoS requirements of response time and deadline. Hence, the number of jobs successfully completed per schedule in SeCRB increases in a linear manner, whereas the successful completion of jobs per schedule in CRB decreases drastically, if we increase the number of job requests. This can be seen in Figure 17, where the maximum number of jobs completed for all the five schedules is more in SeCRB than CRB.

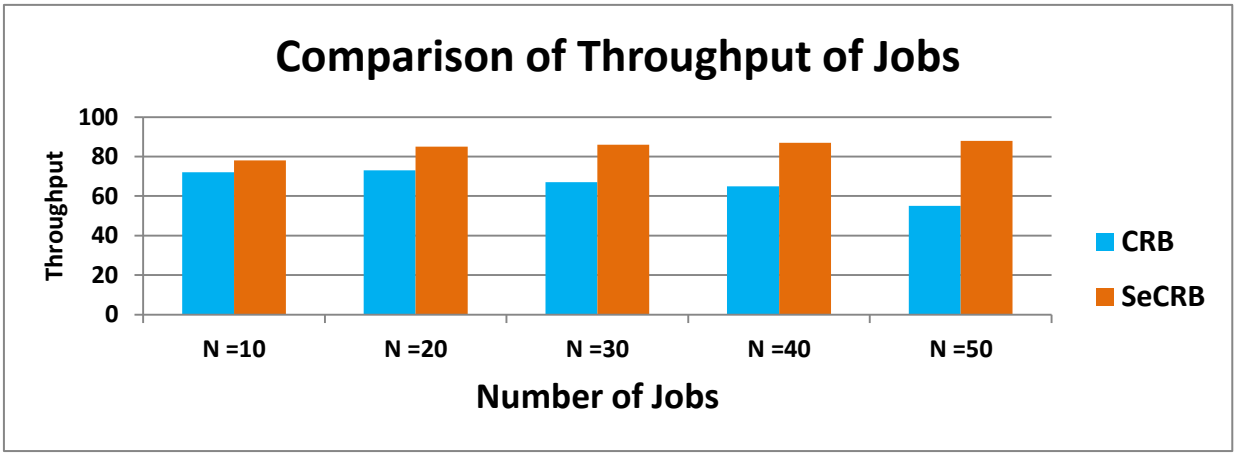

Figure 17: Comparison of Throughput of Jobs

6.4 Image Processing Application Execution for Detection of Defects in Apple

The Central Electronics Engineering Research Institute (CEERI), India has developed an image processing application namely Surface Detection of Defects on fruits. Detection of defects in the fruits is a complex task due to stem areas, diverse types of defects present and natural variability of skin color. The Defect Detection, in particular, requires the precise segmentation of the defects. Also, the identification of surface defects in fruits helps in automatic rejection of fruits. Peel defects in apple were detected using RGB thresholding method; whereas the mechanical defects are detected by Bit-Plane slicing method and Line Defects are detected using sub-image thresholding. In order to improve the execution time, we have parallelized this application as parameter sweep application that takes the picture of fruits in four distinct directions, which are coming through the conveyor belt. These images are compared with test images, which are stored in the database to identify whether there is any black spots surface defect in apples. In real-time, this application requires huge amount of computational resources in an on demand manner. The experimentation setup for processing an image processing application is shown in Figure 18.

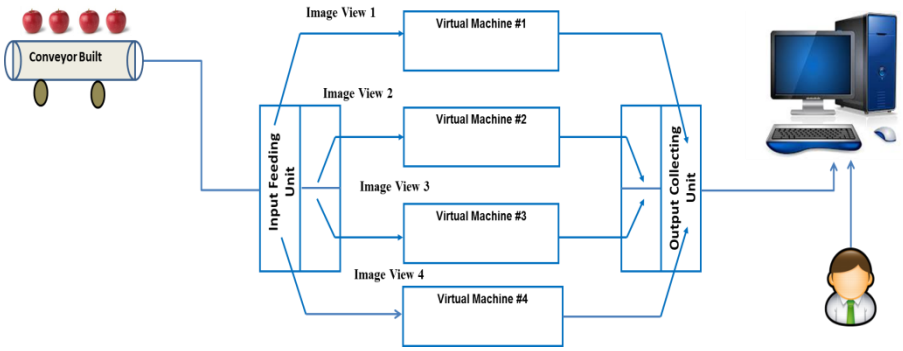

Figure 18: Experimentation Setup

The Sobel defect detection algorithm first captures the fruit images, remove the background and noise using minimal filter as shown in Figure 19. The identified defects in fruit images are shown in Figure 20. The flow chart of the defect detection algorithm is presented in Figure 21. The resultant region related to black spots in fruit images have been counted by human vision and proposed technique. After analyzing the results carefully, we have identified that global thresholding method failed to detect many black spots whereas the marker controlled watershed segmentation method identified the black spots count closer to human eye. Due to the presence of noise in original image such as very high contrast and rough surface, the proposed technique shows some extra black spots but it is negligible. This application requires four different views of apple image. Every view of apple images is sent to four different virtual machines as shown in Figure 18. Finally, the output is aggregated from those processors. The area of each defected region is calculated and returned from all four processors in four different variables each belongs to separate fruit. The total pixels defected for each fruit is calculated using (4). Here, A_iS_j represent the number of pixels defected in j_{th} view of i_{th} apple. The application is tested in CRB as well as in SeCRB for ten iterations to know the consistency of application execution time. The execution time is noted in Table 6.

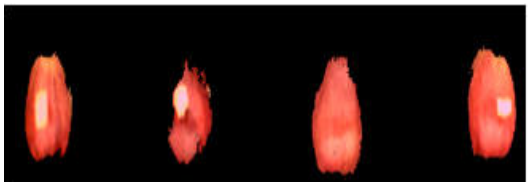

Figure 19: Four views of an apple before processing

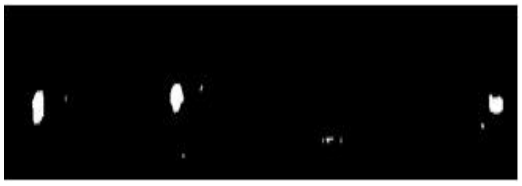

Figure 20: Segmented images showing Defects

$$TPDefected_{ith\_fruit} = \sum_{j=1}^{4} A_iS_j \qquad (4)$$

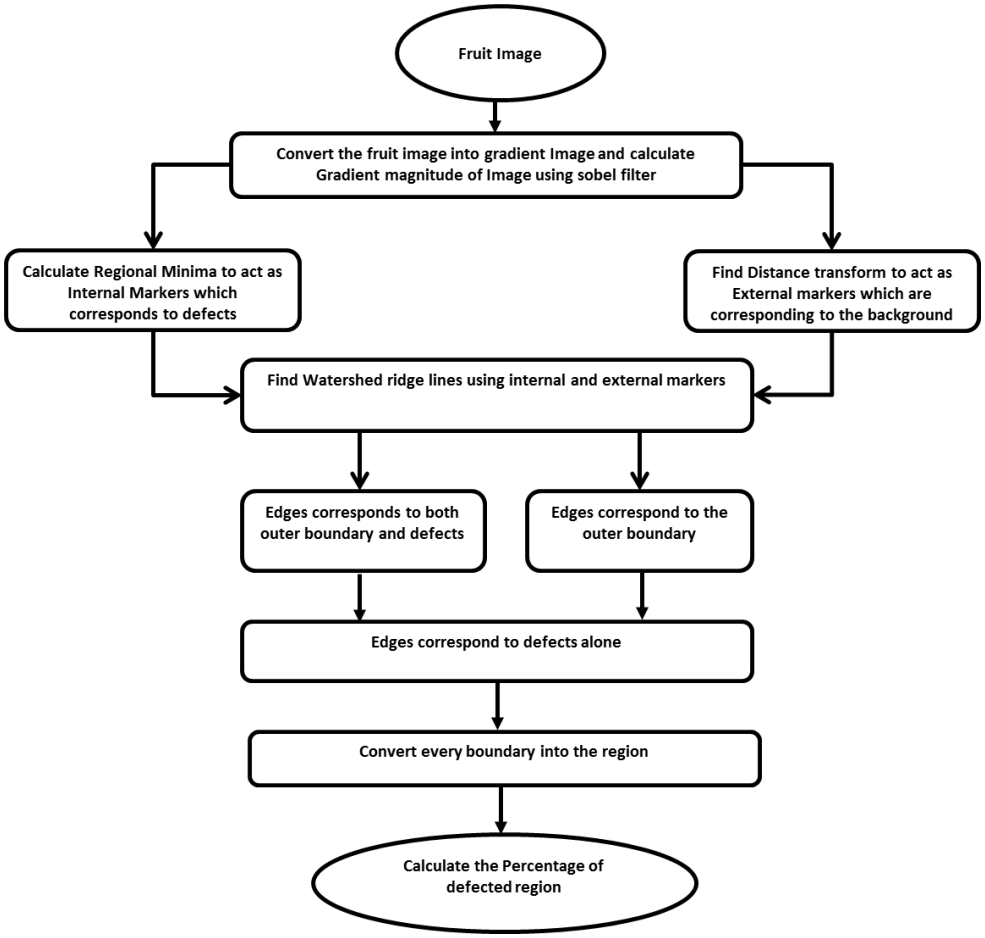

Figure 21: Flow Chart for Defect Detection in an Image Processing Application

Table 6: Application Execution Time for Detection of Defects in Fruits using CRB and SeCRB

Iteration	CRB	SeCRB
	Application Execution Time (Consider Execution Time Only) (In Minutes)	Application Execution Time (Consider Booting Time + Transfer Time + Execution Time) (In Minutes)
1	11.1853	14.9786
2	11.95596	15.8834
3	14.27329	20.72738
4	11.37914	16.12189
5	11.83616	14.38732
6	12.18261	15.34276
7	11.83616	15.64823
8	12.18261	17.36219
9	13.98613	17.49416
10	12.91121	17.32839

7.0 Simulation Results and Discussion

In the absence of a real large-scale testbed to study the impact of the proposed semantic-based resource brokering approach, we have developed a discrete event-based simulator to model the semantic-based resource brokering algorithm explained in this paper. The developed simulator is shown in Figure 22. This section discusses the evaluation of our proposed resource brokering approach by analyzing various performance metrics such as job rejection rate, job success rate, scheduling success rate and the quantification of overhead introduced by our semantic-based resource brokering approach.

7.1 Performance Metrics

To evaluate the performance of our resource brokering approach, we have defined the following metrics in our systems:

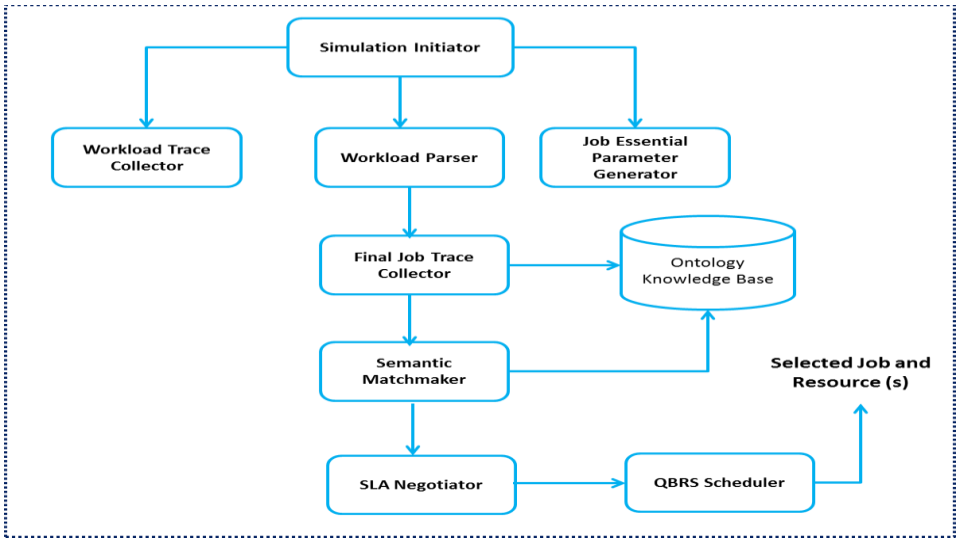

Figure 22: Discrete Event Simulator

Job Rejection Rate: It is the ratio of the number of jobs rejected to the total number of jobs submitted to the resource broker. The job rejection rate is calculated using Equation (5).

$$JRR_{S_p} = \frac{JR_{S_p}}{JS_{S_p}} \quad (5)$$

Job Success Rate: It is defined as the ratio of number of jobs that successfully meet the deadline to the number of jobs submitted to the scheduler. The scheduling success rate is calculated using (6) & (7). Equation (6)

represents the number of jobs successfully completed within the deadline. If the job meets the deadline it is considered a success, else it is a failure. Equation (7) represents the job success rate.

$$JSD_{S_p} = JSTS_{S_p} - JFD_{S_p} \quad (6)$$

$$JSR_{S_p} = \frac{JSD_{S_p}}{JSTS_{S_p}} \quad (7)$$

Scheduling Success Rate: It is defined as the number of jobs completed per unit time. The scheduling success rate is calculated using (8).

$$SSR_{S_p} = \frac{JSTS_{S_p}}{T} \quad (8)$$

7.2 Workload Traces

To evaluate our proposed work, we have used the job workload traces of Feitelson's Parallel Workload Archive (PWA) [62] to model the HPC applications. The collected job traces are parallel and iterative in nature. From the collected traces, we obtained only the submit time, the requested number of processors, and the execution time of the applications. The job traces do not contain the required hardware parameters such as RAM memory and Hard disk memory, and QoS parameters such as deadline and response time. Hence, deliberately we have generated the required job parameters. Our work focused mainly on completing the applications within a deadline. Hence, we have used the methodology proposed by Irwin et al. [63] for assigning deadline values based on the Low urgency (LU) and High Urgency (HU) classes. Statically, we have configured five sites as grid resources and the remaining five as cloud resources. The number of nodes available in each resource is varied from 100 – 500. The nodes are heterogeneous in terms of number of CPUs, RAM Memory, Processor Speed, Hard disk Memory, Bandwidth and Delay. Every host is capable of creating 'N_V' the number of virtual machine instances based on the user application requirements and the availability of resources. The simulation model maps the generated resources onto a semantic-enabled resource broker. The proposed system is accessed by the 'N_U' numbers of users at a regular interval of 'R_I' in a Poisson distribution manner.

7.3 Simulation Experiments

This section evaluates how our proposed resource approach plays a role in completing the jobs within the deadline specified by the user as the job success rate. In addition, this section also evaluates the efficiency of our resource management system by efficiently allocating the jobs in an optimal manner to reduce the job rejection rate and increase the scheduling success rate. The simulation experiment is conducted for five schedules.

Job Success Rate: We have evaluated this parameter based on the jobs that are completed within the deadline. If the job is completed within the deadline, the job is considered as success or else, the job is considered as failure. The job success rate depends upon the urgency factor and number of users submitting the number of jobs. The success rate of a job is less in CRB due to the submission of jobs with physical scheduling types and the provisioning of resources for co-alloc or virtual scheduling types from the shared and controlled grid environment. The proposed resource brokering approach not only creates the on-demand virtual instances in the grid environment but also invokes the cloud controller to create the on-demand virtual instances from the cloud environment. The QBRS algorithm selects the resources from the exact, high-similarity plug-in and high-similarity subsume in an optimal manner by considering deadline as one of the essential factor. Moreover, the guaranteed availability of resources from the SLA negotiation process also plays a main role in the completion of jobs within their deadline. The results as shown in Figure 23 depict the percentage of jobs successfully completed within the deadline specified by the user.

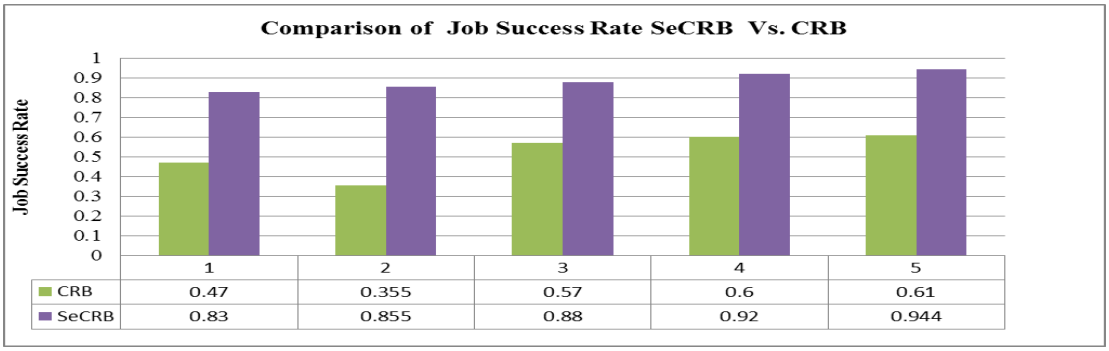

Figure 23: Comparison of Job Success Rate

Scheduling Success Rate: We have evaluated this parameter by considering the jobs which are completed per unit time. The proposed scheduling approach selects the resources in an optimal manner from the exact, high-similarity plug-in and high-similarity subsume regions. Moreover, it selects the resources that complete the jobs

efficiency and within a minimal time before the deadline specified by the user. This leads to an increase in the number of jobs completed before the deadline in a minimal time. Obviously, this increases the scheduling success rate of our proposed approach. The scheduling success rate is shown in Figure 24. The existing CRB gets the job requests, matches them to the resources, to identify the scheduling case, and submits the job into the selected resource. However, in this proposed research work, the resources are discovered in an efficient manner from the high-similarity subsume and high-similarity plug-in regions based on the threshold value that paves the way for increasing the scheduling success rate for the jobs.

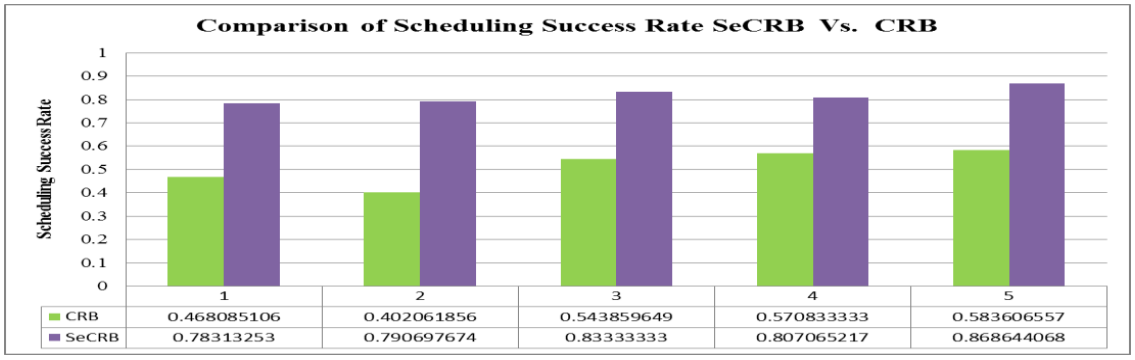

Figure 24: Scheduling Success Rate

7.3 Overhead Analysis of proposed SeCRB

This section is used to evaluate the performance of a proposed architecture in three processes — semantic matchmaking, SLA negotiation and scheduling process. We have measured the time taken for these processes. The total job processing time for five schedules — including the time it takes for semantic matchmaking, SLA negotiation and scheduling — is shown in Figure 25. The experimental results show that the total processing time varies linearly with the increase in the number of resources per schedule. When the number of resources increases, the total processing time also increases. After receiving the job requirements, SeCRB constructs the Algernon query based on them. The generated query is executed in the Ontology Knowledge Base. It identifies the various scheduling use cases — including physical, hybrid, virtual cluster and lease creation — based on the job requirements and in consideration of the hardware and software resource requirements. Finally, the semantic match-making module classifies the matched resources that may fall into three regions namely, exact, subsume and plug-in based on a cosine similarity value that lies between 0 and 1. The resources which may fall in the exact region are capable of running the jobs in the physical grid resources and there is no need to create the virtual resources. The resource which has cosine similarity value greater than or equal to a threshold value of 0.90 has been taken into account for the SLA negotiation process. The semantic-based resource discovery mechanism retrieves more closely matching resources that are relevant to the job requirements. It results in increasing the efficiency of the resource discovery mechanism implemented in the proposed work over a conventional keyword-based resource discovery mechanism, because, even when the resource requested by the user does not perfectly match the available resources described in the knowledge base, the semantic component retrieves the resources that have a high-similarity subsumption relationship and a high-similarity plug-in relationship with the request.

This experiment is carried out by generating five schedules. Each schedule is varied with hosts ranging from 100 - 500. Figure 25 shows the semantic matchmaking time required to find the resources whose requirements match the job's needs. The results show that the higher the number of resources, the longer the query processing time. The variation in the job does not create an impact in the query processing time. The semantic matchmaking time involves querying and updating the ontology knowledge base. The SLA negotiation process calculates the utility value and sends a proposal to the resources in the exact region. If the resource providers in the exact region do not agreed with the utility value, the broker will send the proposal to the resources in the high-similarity subsume and high-similarity plug-in regions. Finally, the QBRS algorithm implemented with the scheduler selects the resources from the exact, subsume and plug-in regions in an optimal manner. The main motivation of the scheduling is selecting resources that satisfy the hardware, software and Quality of Service (QoS) requirements, and completing jobs within the deadline specified by the user.

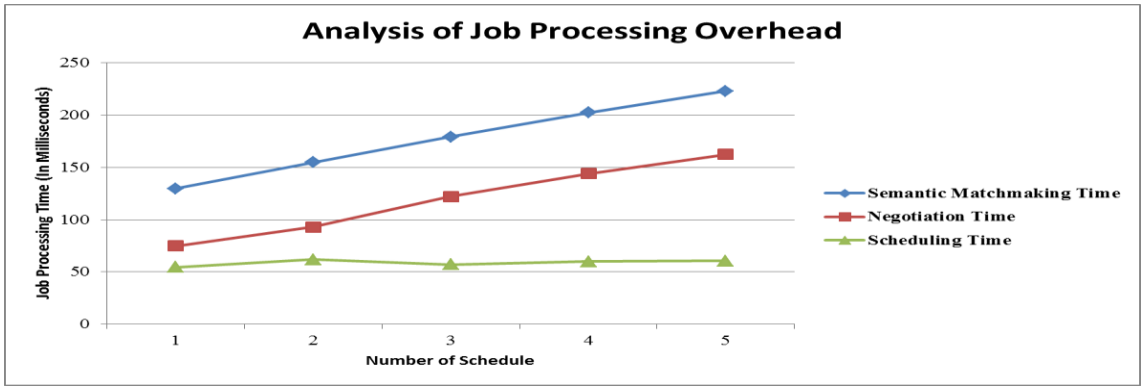

Figure 25: Analysis of Job Processing Overhead

8. Conclusion and Future Work

This paper begins by extending CRB to Semantic-enabled CRB. This proposed semantic-based resource management framework, which manages the grid and cloud resources, provides a better environment for HPC applications. It starts with a description of the grid and cloud resources in a semantic manner. Next, it devises a semantic-based matchmaking mechanism for discovering the resources from the grid and cloud resources. The discovered resources are classified into the exact, high-similarity subsume and high-similarity plug-in regions based on a cosine similarity and threshold value. The proposed work is integrated with an SLA-driven negotiation mechanism that negotiates the user's job requests to satisfy the user's Quality of Service (QoS) requirements. The Quality of Service (QoS) Based Resource Scheduling (QBRS) mechanism implemented in SeCRB selects the grid and cloud resources in a near optimal manner. The proposed mechanism is simulated as well as empirically evaluated by running the real-world bio-informatics application of GROMACS both in the grid and cloud resources, and the results are represented as graphs. The proposed mechanism decreases the rejection rate and increases the success and scheduling success rates for jobs submitted to the resource broker, which enhances the efficiency of a resource management system. In the future, the semantic-based resource management can be extended as a PaaS Engine for federating ~~the~~ cloud resources that belongs to different types of providers along with the development environment. In addition, the feasibility of integrating Particle Swarm Optimization (PSO) - based scheduling algorithm can be explored in order to select resources in an optimal manner, monitor the SLA of virtual instances, and enforce the rules based on SLAs.

ACKNOWLEDGMENT

The authors sincerely thank the Ministry of communication and Information Technology and the Government of India for financially supporting the Centre for Advanced Computing Research and Education of Anna University Chennai, India. Furthermore, the authors acknowledge the Central Electronics Engineering Research Institute (CEERI), India and Department of Biotechnology at the Anna University for their help in guiding the execution of applications.

REFERENCES

[1] Price Water House Coopers, 2002, Powerful technology trends continue despite downturn. PWC Global Technology Center, Menlo Park, CA.
[2] NIST, National Institute of Standards and Technology (2011), http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145_Cloud-definition.pdf.
[3] http://theCloudtutorial.com/what-is-virtualization.html.
[4] http://theCloudtutorial.com/multitenancy.html.
[5] Karl Czajkowski, Ian Foster, Nick Karonis, Carl Kesselman, Stuart Martin, Warren Smith and Steven Tuecke, A Resource Management Architecture for Metacomputing Systems, 1997.
[6] Paolo Missier, Wolfgang Ziegler, Philipp Wieder, "Semantic Support for Meta-Scheduling in Grids" CoreGRID Technical Report Number TR-0030, April 19, 2006, Institute on Knowledge and Data Management & Institute on Resource Management and Scheduling.
[7] Berners-Lee T., Hendler J. and Lassila O. (2001), 'The Semantic Web', Scientific American Magazine. Retrieved on May 15, 2006.
[8] (1992) Ontology, http://www.ksl.stanford.edu/kst/what-is-an-ontology.html.
[9] (2007) Semantic Grid, http://www.semanticGrid.com.
[10] RDF Resource Description Framework (2004), A W3C Recommendation, http://www.w3.org/RDF.
[11] OWL Web Ontology Language Overview (2004), A W3C Recommendation.
[12] Protégé (2011), http://protege.stanford.edu/.
[13] Thamarai Selvi Somasundaram, Balachandar R. Amarnath, R. Kumar, P.Balakrishnan, K. Rajendar, R.Rajiv, G. Kannan, G. Rajesh Britto, E. Mahendran & B. Madusudhanan, "CARE Resource Broker: A framework for

scheduling and supporting virtual resource management", Journal: Future Generation Computer Systems, Volume 26, Issue 3, March 2010, Pages 337–347, doi: http://dx.doi.org/10.1016/j.future.2009.10.005.
[14] Globus Toolkit (2010), http://www.globus.org/toolkit/downloads/4.0.7/.
[15] Thamarai Selvi Somasundaram and et al, "Achieving Co-allocation through Virtualization in Grid Environment" Proceedings from 4th International Conference, GPC 2009, Geneva, May 200, pp. 235-243.
[16] Thamarai Selvi Somasundaram and Balachandar R.A. (2009), "Extending Conventional Grid Scheduler for Leasing Resources", accepted for Poster Session in 10th IEEE/ACM International Conference in Grid Computing 2009 (Grid 2009), Banff, Canada, December 2009.
[17] Balachandar Ramachandriya Amarnath, Thamarai Selvi Somasundaram, Mahendran E, Rajkumar Buyya, "Ontology based Grid Resource Management", Software: Practice and Experience, John Wiley Publications, Vol. 39 (17), October 2009, pp. 1419-1438.
[18] Thamarai Selvi Somasundaram, Rangasamy Kumar and Kannan Govindarajan, "Intelligent Semantic Discovery in Virtualized Grid Environment", Recent Trends in Information Technology (ICRTIT), 3-5 June 2011, pages 644 – 649, ISBN: 978-1-4577-0588-5, Accession Number: 12159128, DOI: 10.1109/ICRTIT.2011.5972362.
[19] The Condor Project (2011), http://research.cs.wisc.edu/condor/.
[20] Computing Resource Execution and Management (CREAM) (2012), http://grid.pd.infn.it/cream/
[21] gLite Workload Management System (WMS) (2012), http://iopscience.iop.org/1742/6596/119/5/052009/pdf/1742-6596_119_5_052009.pdf.
[22] Classified Advertisements (ClassAds) (2012), http://research.cs.wisc.edu/condor/classad/.
[23] Job Description Language (JDL) (2012), http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0102-0_2-Document.pdf.
[24] Grid Interoperability Project (2005), http://www.Grid-interoperability.org.
[25] InteliGrid (2004), http://inteliGrid.eu-project.info/.
[26] The myGrid Project (2008), http://www.myGrid.org.uk.
[27] Semantic Grid (2007), http://www.semanticGrid.com.
[28] Thamarai Selvi Somasundaram and et al, "Semantic-based Grid Resource Discovery and its Integration with the Grid Service Broker", Proceedings of the 14th International Conference on Advanced Computing and Communications, 2006.
[29] Andreas Harth, Stefan Decker, Yu He, Hongsuda Tangmunarunkit, Carl Kesselman, A semantic matchmaker service on the grid, Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters, May 19-21, 2004, New York, NY, USA, DOI: 10.1145/1013367.1013458.
[30] Daniel Nurmi, Rich Wolski et al, "The Eucalyptus Open-source Cloud-computing System, Cloud Computing and Its Applications", October 2008.
[31] OpenNebula. http://www.opennebula.org, 2005.
[32] Emeneker, W., Jackson, D., Butikofer, J. and Stanzione, D. "Dynamic Virtual Clustering with Xen and Moab", in Workshop on XEN in HPC Cluster and Grid Computing Environments (XHPC), 2006.
[33] Agarwal, A., Desmarais, R., Gable, I., Norton, A., Sobie, R. and Vanderster D. "Evaluation of Virtual Machines for HEP Grids", Department of Physics, University of University of Victoria, Technical Report, Canada, 2006.
[34] Xen (2011), http://www.xen.org/.
[35] Vmware (2011), http://www.vmware.com/.
[36] Kernel-based Virtual Machine (KVM) (2011), http://www.linux-kvm.org/.
[37] Amazon EC2 Cloud (2011), http://www.amazon.org/ec2.
[38] Yong Beom Ma, Sung Ho Jang and Jong Sik Lee, "Ontology-Based Resource Management for Cloud Computing", Intelligent Information and Database Systems, Lecture Notes in Computer Science 2011, Springer Berlin / Heidelberg, 978-3-642-20041-0, 343,6592, http://dx.doi.org/10.1007/978-3-642-20042-7_35, 10.1007/978-3-642-20042-7_35.
[39] Baomin Xu, Ning Wang , and Chunyan Li, "Providing a Cloud Infrastructure on Clusters", Proceedings of the Third International Symposium on Electronic Commerce and Security Workshops (ISECS '10),Guangzhou, P. R. China, 29-31,July 2010, pp. 317-320.
[40] Rajkumar Buyya, Suraj Pandey, Christian Vecchiola, "Cloudbus Toolkit for Market-Oriented Cloud Computing", CloudCom '09 Proceedings of the 1st International Conference on Cloud Computing, Springer-Verlag Berlin, Heidelberg ©2009, SBN: 978-3-642-10664-4 doi>10.1007/978-3-642-10665-1_4.
[41] Jorge Ejarque, Raul Sirvent and Rosa M. Badia, "A Multi-agent approach for Semantic Resource Allocation", CloudCom 2010: 335-342.
[42] Balakrishnan Ponnusamy, and Thamarai Selvi Somasundaram, "SLA Enabled Care Resource Broker", Journal: Future Generation Computer Systems, Volume 27 Issue 3, March, 2011 Pages 265-279, doi>10.1016/j.future.2010.09.006.
[43] Jorge Ejarque, Marc de Palol, Inigo Goiri, Ferran Juli`a, Jordi Guitart, Rosa M. Badia, and Jordi Torres, "SLA-Driven Semantically-Enhanced Dynamic Resource Allocator for Virtualized Service Providers" , Fourth IEEE International Conference on eScience, 2009, 978-0-7695-3535-7/08,DOI 10.1109/eScience.2008.15.
[44] Karl Czajkowski, Ian T. Foster, Carl Kesselman, Volker Sander, and Steven Tuecke, SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems. 8th Workshop on Job Scheduling Strategies for Parallel Processing, Edinburgh, Scotland, July 2002.
[45] Srikumar Venugopal, Xingchen Chu, Rajkumar Buyya. A Negotiation Mechanism for Advance Resource Reservation using the Alternate Offers Protocol. 16th International Workshop on Quality of Service (IWQoS 2008), June 2-4, 2008, University of Twente, Enschede, The Netherlands.

[46] Jun Yan, Ryszard Kowalczyk, Jian Lin, Mohan B. Chhetri, Suk Keong Goh, Jianying Zhang: Autonomous service level agreement negotiation for service composition provision. Future Generation Computing Systems, 23(6), Elsevier, 2007.
[47] Seokho Son, Kwang Mong Sim. A Negotiation Mechanism that Facilitates the Price-Timeslot-QoS Negotiation for Establishing SLAs of Cloud Service Reservation. Networked Digital Technologies, Communications in Computer and Information Science, Vol. No. 136, Pg. No. 432 – 446, 2011, Springer Berlin Heidelberg.
[48] StratusLab (2012), http://stratuslab.eu/doku.php/start.
[49] WNoDeS (2012), http://web.infn.it/wnodes/index.php/wnodes
[50] GLUE Schema v1.3, https://forge.cnaf.infn.it/plugins/scmsvn/viewcvs.php/*checkout*/v_1_3/spec/draft-2/pdf/GLUESchema.pdf?rev=24&root=glueschema.
[51] Open Cloud Computing Interface (OCCI) (2011), http://occi-wg.org/.
[52] Marcos Dias de Assunção, Alexandre di Costanzo, Rajkumar Buyya: A cost-benefit analysis of using cloud computing to extend the capacity of clusters. Cluster Computing 13(3): 335-347 (2010).
[53] Simon Ostermann, Radu Prodan, Thomas Fahringer: Extending Grids with cloud resource management for scientific computing. GRID 2009: 42-49.
[54] Alexandru Iosup, Simon Ostermann, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, Dick H. J. Epema: Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. IEEE Trans. Parallel Distrib. Syst. 22(6): 931-945 (2011)
[55] Ming Mao and Marty Humphrey, A Performance Study on the VM Startup Time in the Cloud, 2012 IEEE Fifth International Conference on Cloud Computing, Vol.0,2012,Pp. 423-430.
[56] Algernon (2011), http://algernon-j.sourceforge.net/doc/overview.html.
[57] Fast Fourier Transfer (FFT) (2011), http://www.fftw.org/.
[58] Message Passing Interface (MPI)(2011), http://www.mcs.anl.gov/research/projects/mpi/.
[59] GROMACS (2011), http://www.gromacs.org/.
[60] GTAI, http://dev.globus.org/wiki/Incubator/GT_Auto_Install.
[61] Protein Data Bank (PDB) (2011), http://www.pdb.org/pdb/
[62] D. Feitelson, Parallel workloads archive (1997), http://www.cs.huji.ac.il/labs/parallel/workload.
[63] D. Irwin, L. Grit, J. Chase, Balancing risk and reward in a market-based task service, in: Proc. of the 13th IEEE International Symposium on High Performance Distributed Computing, Honolulu, USA, 2004.

Table 1: Notations & Description

Notations	Description
NS	Number of Schedule
$S \leftarrow \{S_1, S_2..S_p..S_{NS}\}$	It represents set of schedules
S_j	It represents p[th] schedule
M	It represents the total number of resources
$R \leftarrow \{R_1, R_2..R_M\}$	It represents the list of resources
J_i	It represents i[th] job
R_j	It represents j[th] resource
CB	Cloud Broker
$(ER)_{J_i}$	Exact region contain resources that exactly satisfies the demands of job J_i
$(SR)_{J_i}$	Subsume region contain resources that has more capability than demand of job J_i
$(PR)_{J_i}$	Plug-in region contain resources that has less capability than demand of Job J_i
Sim_mat	Similarity matrix holds the cosine similar measure between jobs and resources in $(SR)_{J_i}$ and $(PR)_{J_i}$. Its dimension are $\left(1 * \left\|(SR)_{J_i}\right\|\right)$ and $\left(1 * \left\|(PR)_{J_i}\right\|\right)$ respectively.
$(hs\_SR)_{J_i}$	High-similarity resources that satisfy demand specified by job J_i from subsume region.
$(hs\_PR)_{J_i}$	High-similarity resources that satisfy demand specified by job J_i from plug-in region
$threshold_{SR}$	To filter the resources from subsume region whose similarity is greater or equal to specified value.

$threshold_{PR}$	To filter the resources from plug-in region whose similarity is greater or equal to specified value
$\mathrm{Re}q(J_i)$	$\left\{J_i^{ID}, no\_Nodes_{J_i}, Ram\_needed_{J_i}, Hd\_needed_{J_i}, Deadline_{J_i}, OS_{J_i}, Sw_{J_i}\right\}$
$Avail(R_j)$	$\left\{\begin{array}{l} R_j^{ID}, no\_Nodes_{R_j}, Ram\_Capacity_{R_j}, Hd\_Capacity_{R_j}, \\ Availability_{R_j}, OS_{R_j}, Sw_{R_j}, Type_{R_j} \end{array}\right\}$
$\left\|(ER)_{J_i}\right\|$	Number of resources in exact region of job J_i that is represented as N_ER
$\left\|(hs\_SR)_{J_i}\right\|$	Number of resources in subsume region of job J_i that is represented as N_SR
$\left\|(hs\_PR)_{J_i}\right\|$	Number of resources in plug-in region that satisfies job J_i that is represented as N_PR
$N_{(ER)_{J_i}}$	List of resources to which negotiation is agreed in exact region by broker.
$N_{(SR)_{J_i}}$	List of resources to which negotiation is agreed in subsume region by broker.
$N_{(PR)_{J_i}}$	List of resources to which negotiation is agreed in plug-in region by broker.
P_{Init}	Initial proposal send by cloud broker.
U_{Init}	Utility value of the cloud broker.
$(P_c)_{(R_j)}$	Counter proposal given by resource R_j that receives P_{Init}
$(No.proposal)_{ER}$	Number of resources in exact region that send counter proposals which is received by broker
$(No.proposal)_{SR}$	Number of resources in subsume region that send counter proposals which is received by broker
$(No.proposal)_{PR}$	Number of resources in plug-in region that send counter proposals which is received by broker
$(U_c)_{(R_j)}$	Utility counter of the resource R_j
$U_{(ER)_{J_i}}$	Set of utilities received by broker from the resources in exact region of the job J_i
$U_{(SR)_{J_i}}$	Set of utilities received by broker from the resources in qualified subsume region of the job J_i
$U_{(PR)_{J_i}}$	Set of utilities received by broker from the resources in qualified plug-in region of the job J_i
P_{Avg}	Average proposal received by the cloud broker
$\left(R_{(PR)}\right)_{J_i}$	Resources in plug-in region that is scheduled to Job J_i
JS_{S_p}	Number of Jobs submitted for the Schedule S_p
JR_{S_p}	Number of Jobs rejected for the schedule S_p
JRR_{S_p}	Job rejection Rate for the schedule S_p
$JSTS_{S_p}$	Number of Jobs submitted to the scheduler in the schedule S_p
JSR_{S_p}	Job Success Rate for the schedule S_p
JSD_{S_p}	Number of Jobs successfully met deadline in the schedule S_p
JFD_{S_p}	Number of jobs failed to meet deadline in the schedule S_p
SSR_{S_p}	Scheduling success rate for the schedule S_p
$\mathrm{TPDefected}_{\mathrm{ith\_fruit}}$	Total Pixels Defected in ith fruit

Table 3: Cloud Resources Information

<CloudResourceInfo>
< ResourceInfo 1>
<CloudControllerName>
eucaCloudserver.care.mit.in
</ CloudControllerName >
<ClusterControllerName 1>
<HostName>
eucaclusterserver1.care.mit.in
</HostName>
<NodeControllerName 1>
<HostName>
eucanc1.care.mit.in
</HostName>
<TotalHardDiskMemory>
150000 (GB)
</ TotalHardDiskMemory >
<FreeHardDiskMemory>
10000 (GB)
</ FreeHardDiskMemory >
<TotalAvailableRam>
2048 (MB)
</ TotalAvailableRam >
<FreeRam>
1024 (MB)
</FreeRam>
</NodeControllerName 1>
…
< NodeControllerName n>
…
</Node Controller Name n>
</ClusterControllerName 1>
…
</ClusterControllerName n>
…
< ClusterControllerName n>
< /ResourceInfo1 >
…
< ResourceInfo n>
…
.< /ResourceInfo n >
</CloudResourceInfo >

Table 4: OS Images Metadata Information

<OperatingSystemImageInformation>
<OperatingSystemImages>
<ImageName>
Fedora Core 5.3
</ImageName>
< CloudControllerName >
eucaCloudserver.care.mit.in
</ CloudControllerName>
<SoftwareAvailable>
<fft>
<version>
3.2.x
</fft>
<mpi>
<version>
1.2.6
</version>
</mpi>
</SoftwareAvailable>
<ImageURI 1>
<RootImage>
emi-223457
</RootImage>
<KernelImage>
eki-234344
</KernelImage>
<RamdiskImage>
eri-34333
</RamdiskImage>
</ImageURI 1>
...

<ImageURI N>
...

</ImageURI N>

<OperatingSystemImages>
</OperatingSystemImageInformation>