RESOURCE ALLOCATION IN SELF ORGANIZING CLOUD – A SURVEY

S.Thejeswi
Department of CSE
K.S.R. College Of Engineering
Tiruchengodu,Tamil Nadu
India
thejusenthil@gmail.com

C.Thirumalaiselvan
Assistant Professor
Department of CSE
K.S.R. College Of Engineering
Tiruchengodu,Tamil Nadu
India
ctsn027@gmail.com

***Abstract*- By gearing virtual machine (VM) technology which provides performance and fault seclusion, cloud resources can be provisioned on demand in a pulverized, multiplexed manner rather than in monumental pieces. By incorporating unpaid assistant computing into cloud architectures, we foresee a colossal self-organizing cloud being twisted to gather the vast perspective of unexploited service computing power over the Internet. Toward this new architecture where each participant may separately act as both resource consumer and provider, we propose a fully distributed, VM-multiplexing resource allocation format to supervise decentralized resources. Our approach not only achieves maximized resource utilization using the proportional share model but also delivers provably and adaptively finest execution efficiency. We also devise a new multi attribute range query protocol for locating best fit nodes. Divergent to existing solutions which often produce immense number of messages per request, our protocol produces only one lightweight query message per task on the Content Addressable Network . It works efficiently to discover for each task its best fit resources under a randomized policy that mitigates the contention among requesters. We show the self organizing cloud with our optimized algorithms can make an enhancement by 15-60 percent in system throughput than a P2P Grid model. Our solution also exhibits fairly high adaptability in a self-motivated node- agitating environment.**

***Keywords-Cloud computing, VM-multiplexing resource provision, convex optimization, P2P multiattribute range query*.**

I. INTRODUCTION

Increasingly, infrastructure providers are supplying the cloud marketplace with storage and on-demand compute resources to host cloud applications. From an application user's point of view, it is desirable to identify the most appropriate set of available resources on which to execute an application. Resource choice can be complex and may involve comparing available hardware specifications, operating systems, value-added services (such as network configuration or data replication) and operating costs (such as hosting cost and data throughput). Providers' cost models often change and new commodity cost models (such as spot pricing) can offer significant savings.

As cloud computing services rapidly expand their customer base, it has become important to provide them economically. To do so, it is essential to optimize resource allocation under the assumption that the required amount of resource can be taken from a common resource pool and rented out to the user on an hourly basis. In addition, to be able to provide processing ability and storage capacity, it is necessary to allocate simultaneously a network bandwidth to access them and the necessary power capacity. Therefore, it is necessary to allocate multiple types of resources (such as processing ability, bandwidth, and storage capacity) simultaneously in a coordinated manner, instead of allocating each type of resource independently.

Infrastructure providers offer flexible and cost-effective resources for hosting network-centric and cloud applications. An infrastructure provider rents computing and storage resources together with network bandwidth and supporting services, according to prespecified user requirements, for precisely the length of time that a user requires them. Rented resources may be used to (i) host an application's entire infrastructure, or (ii) support overflow capabilities during high-load situations, or (iii) provide disaster recovery capabilities. The cost of renting infrastructure resources is inexpensive due to economies of scale. Moreover, an infrastructure can be tuned to the current load of an application or the current revenue generated by an application.

II. RESOURCE ALLOCATION

Infrastructure providers are increasingly supplying the cloud marketplace with storage and on-demand compute resources to host cloud applications. Amazon Elastic Compute Cloud (EC2) [1], ElasticHost [2], GoGrid [3], Flexiscale [4] and Rackspace [5] all

supply resources to the *IaaS* (Infrastructure as a Service) market. Each infrastructure provider offers a particular infrastructure capacity, with a variety of hardware configurations, operating systems and supporting services. Different providers offer different pricing structures for using their infrastructure resources and they may have different application programming interfaces (APIs) for requesting and configuring resources. This makes it difficult for users to migrate between providers within a multi-provider cloud market- place.

One way to utilize the cloud marketplace is to develop models for mapping application constraints onto ranges of infrastructure products. As the infrastructure provider marketplace develops and user expectations increase, providers are introducing a richer set of pricing models and value-added services. For example, Amazon now offers *Spot Prices* for their resources; these allow resources to be obtained at significant discounts to their normal fixed price structures. Other providers offer ranges of network bandwidth options and services, such as resilience and load scaling. These capabilities are added dynamically in order to automate the process of resource selection for a particular application, techniques for expressing infrastructure and software requirements are needed. From the application user's point of view, there is a challenge to find appropriate resources within a multi-provider cloud.

Searching for infrastructure resources by hardware specifications, such as CPU or memory, may not be sufficient. A user may be interested in searching for resources by operating system type or by requiring particular software items or services. For example, a user may require two compute nodes with hardware configurations of 2.4GHz dual core Intel CPU, 2GB memory, 250GB local disk, running a 32- bit Ubuntu 9.10 Karmic system pre-installed with JDK 1.6 and Tomcat 5.0 web server, with each compute node being located at a different geographical location (for resilience) with a low network latency. A user should be able to compare easily alternative options and select resources from different infrastructure providers.

A built in infrastructure resource discovery engine which operates in a multi provider cloud marketplace with multiple kinds of user requirements (including cost) is proposed. in particular we utilize a constraints based model so as to provide flexibility in terms of provider independence as well as giving an abstract view of infrastructure capabilities. A two-phase software abstraction model which facilitates infrastructure resource discovery across multiple providers is being analyzed. An application's hosting requirements are specified by means of constraints. An abstract interface for managing resources in a multi-provider environment and study how infrastructure features and user requirements can be expressed and used to find suitable resources for hosting an application is required.

An infrastructure provider is an entity that offers resources for lease according to a specific pricing model, along with a management interface for users to browse, purchase, monitor and control resources. A resource maybe a compute resource with a selected operating system, a storage resource or a service providing predefined functionality . For most providers a compute resource is a virtual machine (VM) with a specific hardware configuration and operating system type. It may also contain other software or data required by a user. Infrastructure providers offer different hardware configurations with varying pricing models. Each physical infrastructure resource is owned by an infrastructure provider and will usually be shared between a number of users. In this section, we compare the resources offered by three popular infrastructure providers. We discuss some of the research works related to this field.

Currently, it is difficult to compare the different resource options. Comparision of resources is likely to become more complex as additional value-added services are offered by providers directly. There are a number of alternative approaches for offering and selecting resources for the cloud marketplace. For example, there are a number of open source IaaS provider implementations that expose clusters of VM hosts as IaaS resources. These include Eucalyptus [6], OpenNebula [7] and Nimbus [8]. Most clone Amazon's EC2 API to allow easy access through existing technologies. These implementations offer a set of cloud management operations; however, they are resource-centric, giving external definitions of the capabilities of particular resource types and prices.

III. RESOURCE ALLOCATION FRAMEWORKS

A. RESERVOIR Framework

The RESERVOIR [9] framework creates an environment where different cloud services (or resources) can be managed together. Services are encapsulated in a Virtual Execution Environment (VEE). Virtualisation technology and physical resources are represented using a Virtual Execution Environment Host (VEEH), which is managed by the Virtual Execution Environment

Management (VEEM) system. VEEM utilises OpenNebula and manages the deployment and migration of VEE on VEEH. A Service Manager (SM) is introduced to instantiate theservice applications and manage the service level agreement (SLA). The RESERVOIR framework provides a service specification mechanism [10] to define application configurations by extending DMTF's OVF standard [11]. This service specification includes VM details, application settings and deployment setings. The RESERVOIR's approach gives a provider-focused view of resources. It allows service providers to specify a complete definition of their cloud services.

B. *SLA@SOI Framework*

The SLA@SOI [12] framework introduces a multi layer SLA management framework for service oriented infrastructures. The SLA@SOI model consists of terms, service level objectives (SLO) and conditional rules. Terms are the attributes that define the infrastructure resources, such as number of CPU cores, memory allocation, or resource's geographical region. SLOs are the attributes for monitoring resource performance, such as CPU utilisation. Conditional rules define the actions that need to be executed if there are changes in resource performance (SLA). The SLA@SOI approach is focused on the management,monitoring and readjustment of SLAs.

The Intercloud project [13] implements an ontology based resource catalog that captures the features and capabilities offered by cloud providers. The proposed ontology defines the physical attributes of resources, such as CPU and storage. It also defines other features of resources, such as security, recovery, and compliance capabilities from the provider's point of view.

The Mosaic [14,15] project studies the cloud interoperability and portability issues. It also tries to address service discovery, service composition and SLA management issues. It proposes an ontology [16] for resource annotations. The Mosaic's ontology defines a set of non-functional and functional properties which are used to describe cloud resources. The definition of SLA in Mosaic follows the SLA@SOI concepts.

IV. A RESOURCE DISCOVERY MODEL FOR MULTI-CLOUD APPLICATIONS

The proposed resource discovery model is based on the notion of service-centric systems which are subsequently deployed as dynamic applications which reside in a multi-provider cloud. The usage of infrastructure resources is application dependent: some applications are implemented using a dynamic collection of infrastructure resources; others only use cloud resources during periods of high load, or to provide disaster recovery.

A. *An Infrastructure Provider Agnostic Approach*

In the model, applications are agnostic to the underlying provider; this means that applications can use the most appropriate cloud resources when they are needed. The Zeel/i framework[17] uses the customised mechanisms of a host provider to configure an environment for a user. The environment is wrapped in a provider agnostic API which insulates applications from provider APIs . The first step in using Zeel/i is to determine the current resource marketplace, i.e. the cloud of resources currently made available from resource providers which satisfy the application's requirements. The choice of the most suitable provider depends on the constraints of the application and also on the kind of financial arrangements that suit the application owner (forming a financial agreement with a provider may be as simple as creating an account and providing credit card details, but it could also involve personally auditing the provider's data centres for particular application domains, such as financial services).

Once a pool of resources is defined, an application can select and allocate appropriate resources. In order to be provider-neutral, Zeel/i takes a discovery-based approach where an application attempts to find resources (from the pool of acceptable providers) that meet its requirements. Once a collection of feasible resources, called resource templates, have been identified for an application, they can be filtered using a heuristic (e.g. cheapest cost) to select the best match for an application.

Once an appropriate resource has been identified, it can be reserved. A reservation is a short term hold on resources designed to provide a guarantee that the resources will be available when needed. Once resources are needed they are instantiated, which provides the application with a running resource which it can use.

When the application finishes with a resource, it discards it by returning it to the provider. The resource model allows application users to specify resources without the knowledge of the internal implementation of the underlying infrastructure provider. This insulates users from the frequent changes that arise in an underlying provider's APIs (for managing infrastructure resources). The resource model allows applications to

take a dynamic, commodity-based approach to resource usage. An application cloud can be deployed and scaled according to application and system constraints, costed within a given budget and have portability over the set of available infrastructure providers.

B. Using Two-Phase Constraints-Based Discovery Approach

Currently, cloud providers advertise their resources through websites. Typically, users write code targeted for a single cloud provider, statically selecting resource types for their applications. Often they build customised operating system images to enable their software to be deployed onto these resources. When a provider changes its pricing structure or adds new features and value added services (which they do on a near-weekly basis), users must again evaluate their cloud environment and decide whether they should switch provider, resources, or even modify their operating system images.

This static human-driven approach does not scale to a world in which multiple providers, each with their own unique features, compete for users' workloads. Providers are interested in packaging and selling infrastructure products their products include templates for resource types, each with a given amount of RAM, CPU count and often some scratch storage. Typically, the requirements of an application are more complex than available resource templates; for example,

- a database component may require storage resilience of 99.9999%;
- a transcoder used in a digital media video application may prefer a CPU with the SSE 4.2 instruction set;
- low latency between some services is required if acceptable performance is to be guaranteed.

Currently, there is a mismatch between the provider perspective of neatly packaged ranges of infrastructure products and the application perspective of constraints between services constraints optimisation engine.In[18], a model to formulate the application requirements into constraints. Various types of constraints are identified:

- a hardware constraint refers to hardware requirements of the application, such as CPU or RAM.
- a storage constraint refers to storage requirements of the application, such as persistent storage space.
- a software constraint refers to any software or service utilised by the application, such as Java JDK 1.6.
- a performance constraint refers to the Quality of Service (QoS) or Service Level Agreement (SLA) that the application need to achieve, such as response time and network latency.
- a cost constraint refers to the budget or cost restriction of the application, such as a maximum of $1 per hour per infrastructure resource.
- a compliance constraint refers to any standard, regulatory or compliance requirements that the application need to comform with, such as data center standard TIA-942 or UK Data Protection Act 1998.

V.TWO-PHASE CONSTRAINT MODEL

An application's constraints are partitioned into hard constraints and soft constraints. Hard constraints refer to must-have requirements which persist and are invariant during execution, such as CPU, memory, or legislation regulations; soft constraints refer to desired requirements which can change or be re-prioritised, such as the cost of consuming resources. The classification of requirements into hard constraints or soft constraints depends on user's need. For example, network latency can be classified as soft constraints if an application prefer a fast response time; however, one could classify network latency as hard constraints if the application's response time must not exceed certain threshold limit.

In the first phase, the proposed model only utilises hard constraints. A set of possible resources is generated which satisfies the constraints;as shown in figure 1 each resource is associated with a cost model which can be interrogated with usage scenarios (such as "4 days of use, 8GB of inbound internet bandwidth, 4GB of outbound internet bandwidth, 1TB of data stored for 4 days") to determine a cost, taking into account of bulk discounts, special offer periods, etc.

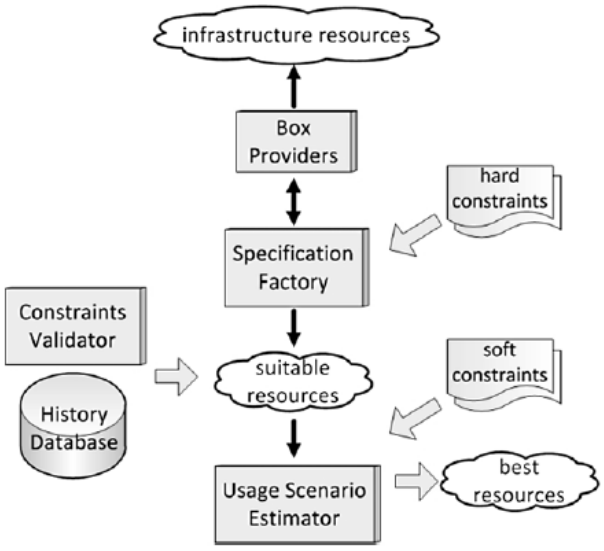

Figure 1:Constraint optimization process

The History Database is a component that holds historical monitoring data. This includes information such as provisioning latency, probabilities of failure for particular providers, hardware offers, application-specific performance and other general performance benchmarks such as CPU, RAM, Disk, or Network data. Historical monitoring data allows the system to select the most appropriate resource based on actual observed behaviour, rather than provider-advertised performance data. This allows users to react immediately to cloud problems, such as provider failure. Consequently, the provisioning system can avoid resources that have a history of unreliability.

The Specification Factory component can combine all of the hardware and software options that are made available by infrastructure providers in order to generate a pool of available resources–resource templates (search space). For example, a compute resource offered by Amazon AWS EC2 could be a High-CPU Medium instance combined with a Ubuntu 11.04 Natty AMI image in the AWS EU region; or it could be a Standard Small instance combined with a Windows Server 2008 AMI image in the AWS US-East region. In order to allow runtime discovery of resources which satisfies the constraints of the application it is necessary to have a clearly defined description of cloud resources.

There are similarities between the Specification Factory and intercloud research; for example, in Bernstein et al. [19], cloud resources are described using semantic web techniques. The proposed solution is based on high-level constraints which can be applied to multiple description formats for example, against a set of Java interfaces, SPARQL queries against an RDF ontology or constraints in a rules engine–whilst keeping an application-based viewpoint to keep the model at a level appropriate to users.

The Constraints Validator is used to filter the resource set (search space) by ensuring that all application constraints are satisfied. The engine makes a move by choosing a resource from the search space, and checking if that particular resource meets the application constraints; if the constraints are satisfied, then the move is valid.

A solution path is formed by combining (chaining) each valid move (see Figure 2). A set of solution paths (solution space), which satisfy the application constraints, will be generated. If none of the available resources satisfy the constraints, no solution is generated. A “degraded” solution path, which only partially satisfies the application constraints (e.g. with lower hardware specifications) could be offered. The use of “degraded” solution paths is not considered in this paper.

The Usage Scenario Estimator can be used to estimate the cost of a specific usage scenario. The engine generates the solution space (identified from the Constraints Validator) and assigns a cost to each solution path. The cost is an estimate of a typical usage scenario for a particular application, with chargeable operations, such as:

- the cost of allocating/deallocating a resource;
- the cost of reserving a resource;
- the cost of keeping a resource switched on for an application-specified period of time; the cost of using a certain amount of incoming/outgoing bandwidth within the provider (for an application-specified period of time);
- the cost of performing a number of I/O operations on a local or a SAN disk (for an application specified period of time);
- the cost of storing a given volume of data on a local or a SAN disk (for an application-specified period of time).

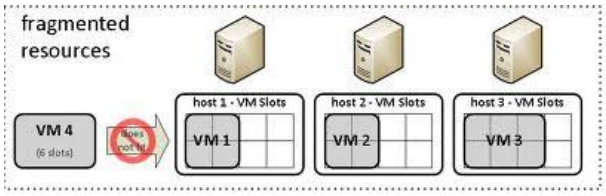

Figure 2:Resource fragmentation

A heuristic can give weights to each of these factors in order to obtain an application-centric metric. In the proposed model, cloud application developers create costing scenarios as an integral part of the development process, this adds an additional development cost but it simplifies the process of integration in a multi-provider marketplace. The model provides a number of predefined scenarios, such as, compute-only, network-only and storage-only. These scenarios allow developers to use the model without significant extra effort. Once a metric is identified, the engine can select the best solution path that match the application requirements. For example, suppose that a user needs to provision a web application which requires three compute resources with the following requirements:

- Ubuntu Maverick operating system(hard constraint 1)
- At least 1 CPU, 1 GB Memory, 20 GB disk space for each compute resources (hard constraint 2)
- Must be located within EU region (hard constraint 3)
- Must be from a reliable infrastructure provider with at least 99% uptime (hard constraint 4)
- A maximum budget of $0.50 per hour for 30 days period (soft constraint 1)
- A network latency of less than 100 milliseconds between each compute resource (soft constraint 2)

The Box Provider component establishes a provider-agnostic view of a multi-provider infrastructure cloud. The Specification Factory generates a list of resource templates which denote the resource offerings from different infrastructure providers. The initial set of resource templates could be huge in size, depending on the number of infrastructure providers being used. For example, nearly half a million resource templates could be generated for the Amazon AWS provider (by combining different AWS instance types with different AMI images, regions and availability zones). The size of the search space varies depending on the search algorithm. A naive exhaustive search to find Y candidates from a pool of X resources could generate (X Y) different combinations (assuming the same type of resource template can be reused). In practice, it is highly desirable to limit the search space by imposing sufficient hard constraints using different heuristic algorithm. For example, if location treated as a hard constraint, then the search space could be significantly reduced.

In phase-one of the proposed model, the Constraints Optimisation Engine identifies a subset of suitable resource templates which meet all of the hard constraints. These templates are then validated using information from the History Database and Constraints Validator. In phase-two search of the model, the engine applies a cost model to each feasible resource template, allowing the Usage Scenario Estimator to compare the soft constraints. For a usage scenario where the cost constraint is the major factor, the engine assigns a higher weighting to this constraint and tries to identify the cheapest solution path at the time that the infrastructure is being deployed. In practise, usage scenarios may change over time. If the network latency becomes more important than cost, then the engine assigns a higher weighting to the network latency (and possibly selects a different solution path). This model provides a unified framework for comparing infrastructure resources in order to select the most appropriate combination of resources for a given application.

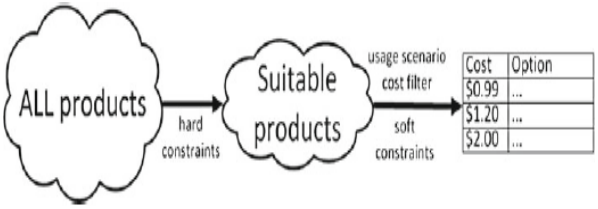

Figure 3:Two-Phase Resource selection From Providers

In the model, applications are insulated from the rapidly changing provider APIs and from the dynamically changing collection of infrastructure products and services. The view of the model is application oriented and infrastructure products are chosen to satisfy the application requirements.

The following constraints (see Figure 3) are used to guide infrastructure product selection. The data coming from the financial services feed has an approximate flow rate of 500Kbps from 8.00am to 4.30pm (approximately 2 gigabytes per day). This information is sufficient to model the bandwidth cost involved in a potential database replica placement. The database replicas require:

- incoming internet bandwidth of at least 500Kbps from the financial services feed and outgoing 500Kbps to the other database replica
- between 10Mbps and 20Mbps of bandwidth to the condor pool nodes (to satisfy each node's 2Mbps requirement)
- minimal latency to ensure that the condor pool acquires timely data feeds despite synchronization

• sufficient redundancy and site resilience to ensure that there is a very high probability that at least one infrastructure branch survives a major provider failure.

• legislation compliance to ensure that no data is stored or transmitted outside of the EU region.

The condor pool should have the lowest possible latency link to the database server. They do not have to be geographically distributed. This latency requirement, along with the condor pool bandwidth constraint results in a pool being placed on the same provider's resources. A key benefit of our proposed approach is its two-phase nature.

The initial search phase identifies only resources which are suitable for the Database Replica Nodes and the Condor Nodes. The hard constraints include the hardware specifications and advertised/observed reliability, as well as the software configuration, such as the operating system type associated with it.

A financial services application and a high performance computing application are used to illustrate the execution of the proposed resource discovery mechanism.

VI. A FINANCIAL SERVICES APPLICATION

The financial services domain provides a challenging environment for infrastructure deployment. Other than general requirements, such as hardware (CPU, memory), software (C++ compiler) and storage (disk space), the financial services domain requires the use of other stringent requirements, such as data quality and integrity, security (encryption and authorisation), performance (response time, network latency) and compliance (legislation regulations) [20]. In order to satisfy the constraints of high availability and high site resilience, multiple mirror infrastructures can be provisioned in different geographical locations (location constraints); however, the performance constraints may require low network latency between different sites. These opposing requirements of geographical separation and low latency have the potential to severely prune the solution space.

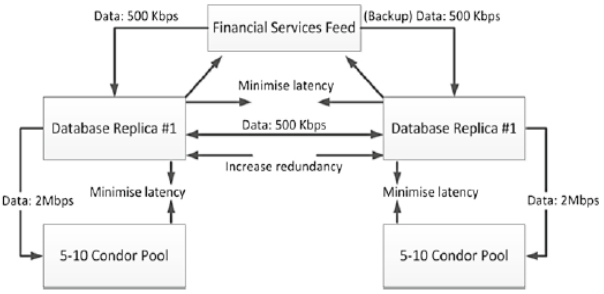

Figure 4: A simplified infrastructure for financial feed application

For example, (see figure 4) as part of the financial services compliance requirements, no data should be hosted or transmitted outside the EU region. The second phase selection balances the complexities of cost, non-functional requirements and preferences across many providers, known as soft constraints in our proposed model. Delaying consideration of cost until the second phase also enables a much more sophisticated view of the price of a system as a whole rather than the cost of individual resources from providers–considering costs per resource may not result in the most appropriate infrastructure for an application. The infrastructure for the financial system is shown in Figure 6; the compute nodes to be used for Condor are expensive while the database replica server is relatively cheap. This results in an application infrastructure with low cost. A search based solely on individual resources could reject the condor segment of the infrastructure as being too expensive.

A. A High Performance Computing Application

Amazon AWS is considered to be 5 sub-providers in different regions–US east, US west, EU west, AP north east and AP south east. Infrastructure resources are described in terms of resource templates, annotated with appropriate metadata. Metadata information includes hardware specification (e.g CPU, RAM, storage capacity), operating system type, resource location and performance reliability.

Resource selection result for financial services application conventional high performance computing (HPC) applications run on super-computers, compute clusters or grids, which typically reside within organisation boundaries. Cloud Computing offers an alternative platform for executing HPC application.

Block Matrix Multiplication is used as an example of how HPC applications can be mapped onto Cloud infrastructure resources. The multiplication of an m × p block matrix A and a p × n block matrix B can be defined as:

$$1 \le i \le m \;.\; 1 \le j \le n \;.\; Cij = \sum_{K=1}^{P} Aik\ Bkj$$

where each element of matrices A and B is defined as a sub-matrix. The following example illustrates the cases where m, n, p = 2 (i.e. 2 × 2 block matrices).

Consider two 16K × 16K matrices, each composed of four 8K ×8K square blocks. A straightforward multiplication requires 8 matrix-matrix multiplications followed by four matrix additions. The operations can be carried out in parallel using 8 Cloud compute resources. Each matrix-matrix multiplication requires three matrices (two input matrices and one result matrix) in memory during execution. Using 64-bit precision, each 8K × 8K matrix requires 512 MB RAM. Therefore each Cloud compute resource must have at least 1.5 GB RAM. Another possible constraint is fast turn around time. In this case reliable high performance compute resources are preferred.

VII. EXPERIMENTAL RESULT FOR RESOURCE SELECTION

In the financial services experiment, Amazon AWS and Flexiscale are chosen as the infrastructure providers. Resource templates and pricing structures are generated dynamically during runtime depending on providers' offerings (e.g. a new AWS AMI image is created). Initially, before any constraints are imposed, there are nearly half a million resources templates that are made available by different providers. The number of suitable resource templates is significantly reduced once hard constraints and soft constraints are taken into account. In the experiment, the demonstrator takes an average of 32 seconds to execute the resource discovery process. However, almost 95% of the execution time is spent in establishing a connection to the infrastructure providers' API across the internet and populating the resource templates in real-time. This execution time could be improved by regularly caching the resource templates in a local database.

The prototype demonstrator selects one infrastructure site at Amazon Dublin using a m1.large instance for the Database and c1.xlarge instance for the Condor Nodes; and another infrastructure site at Flexiscale London using 4 CPU 8GB RAM for the Database and 6 CPU 6GB RAM for the Condor Nodes. This fulfils the general requirements (e.g. hardware, software, etc) as well as other performance and compliance requirements that each site must be reliable and geographically separated from the other (while maintaining a low-latency link to the data feed and each other).

A. Resource Selection Result For Hpc Application

A prototype demonstrator has been implemented using the proposed two-phase resource discovery model. Some infrastructure providers offer resources across multiple geographical regions. For example, in the demonstrator, In the HPC experiment, 90 resource templates are found by using the proposed resource discovery model to search for a 64-bit machine with at least 1.5 GB RAM and a Ubuntu Oneiric operating system across all of the available infrastructure providers, with the cheapest being an AWS resource template, m1.large–64-bit with 2 CPU cores and 7 GB RAM. By provisioning a cluster of machines of the same type, the cheapest cost per hour for 8 compute resources is $2.80. If cost is of higher priority than turnaround-time, then AWS spot instances should be preferred. In the proposed model, AWS spot instances are differentiated by resource reliability attribute. Spot instances are annonated as potentially unreliable resource type since they may be terminated by the provider if demand for the spot instances is strong.

If cost and turnaround-time are both high priorities, then a different candidate could be considered. One way of ensuring faster execution time is to utilize multiple CPU cores on a compute resource. A multi-core compute resource is typically more expensive, but by taking the advantage of AWS spot instance, a more affordable compute resource may be identified.

The cheapest cost per hour for an 8 compute resource is $1.76. A cluster of 8 c1.xlarge compute resources is provisioned. A Java based parallel Block Matrix Multiplication orchestration has been executed on the cluster. The orchestration is dynamically installed using a multi-threaded optimised GotoBLAS library on each machine instance. Table 6 shows the average, minimum and maximum execution times of 20 executions for both 8K ×8K and 16K ×16K block matrix multiplications.

The benefits of using two-phase approach the same in both cases; however, a batch system does not require high network bandwidth and puts more emphasis on storage size and speed. For the financial services example, the demonstrator could select an alternative infrastructure provider if the cost of provisioning such

infrastructure is cheaper at the time than the infrastructure in the current one. For example, if the Amazon AWS data transfer price is reduced, then the tool may give an alternative resource solution. In the financial services domain, market conditions may alter the priority of an application's constraints: for example, after a major incident, such as Japan's tsunami or a major downgrade of US credit ratings, the financial analyst may need to perform computation intensive analysis rapidly. Under these circumstances, the cost of provisioning the infrastructure might become an insignificant constraint; low latency between resources.

VIII. CONCLUSION

Cloud infrastructure providers offer highly flexible and cost effective resources for use by a new generation of network-centric infrastructure applications. The infrastructure marketplace is developing rapidly with new providers, infrastructure products and value-added services coming to the market. This rapid development environment places significant strain on existing infrastructure application users because of the complexity of selecting appropriate resources from a dynamic market place. Mapping an application's requirements onto a set of resources is challenging. We have developed a two-phase resource selection model using a constraints-based approach which enables users to match their applications' requirements to infrastructure resources. The model provides an application focused, rather than a provider-focused, view of resources. This enables application requirements to be expressed in a domain specific way rather than using the terms used by particular providers. By adopting this application-centric approach, constraints are used to express the requirements of an application– sets of such constraints determine the multi-resource requirements of an application. This approach is usually applied in a two-phase manner that enables users to select appropriate resources and then balance the needs of the application infrastructure with functional and non-functional requirements.

The constraints optimisation engine proposed in our model is still in preliminary stage. We are currently improving the engine by considering other optimization techniques and rules engines. We are also investigating the possibility of describing application requirements using domain specific ontologies. Scalability is likely to be a problem when a large number of infrastructure providers are considered. We are investigating a three-phase (or multi-phase) approach, which incorporate an additional phase to filter suitable providers before searching for resources. We believe that our approach offers an effective mechanism to compare and select resources from the myriad of providers and infrastructureproducts.

IX. REFERENCES

1.Amazon Elastic Compute Cloud (EC2). http://aws.amazon.com/ec2. Accessed 04 Jan 2012

2. ElasticHosts. http://www.elastichosts.com/. Accessed 06 Jan 2012.

3. GoGrid. http://www.gogrid.com/. Accessed 04 Jan 2012.

4.FlexiScale.http://www.flexiant.com/products/flexiscale/ . Accessed 04 Jan 2012.

5. RackSpace. http://www.rackspace.com/. Accessed 06 Jan 2012.

6. Nurmi D, Wolski R, Grzegorczyk C, Obertelli G, Soman S, Youseff L, Zagorodnov D (2009) The Eucalyptus Open-Source Cloud-Computing System. In CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International, Symposium on Cluster Computing and the Grid 124–131.Washington, DC, USA: IEEE Computer Society.

7. Montero RS (2008) OpenNebula: The Open Source Virtual Machine Manager for Cluster Computing. In Open Source Grid and Cluster Conference. Oakland, CA.

8. Marshall P, Keahey K, Freeman T (2010) Elastic Site: Using Clouds to Elastically Extend Site Resources. In 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing 43–52. IEEE.

9. Rochwerger B, Breitgand D, Levy E, Galis A, Nagin K, Llorente IM, Montero R, Wolfsthal Y, Elmroth E, Caceres J, Ben-Yehuda M, Emmerich W, Galan F (2009) The Reservoir model and architecture for open federated cloud computing. IBM Journal of Research.

10. Gal´ n F, Sampaio A, Rodero-Merino L, Loy I, Gil V, Vaquero LM (2009)a Service specification in cloud environments based on extensions to open standards. In Proceedings of the, Fourth International ICST Conference on Communication System software and middleware, COMSWARE '09 19:1–19:12. New York, NY,USA:ACM,http://doi.acm.org/10.1145/1621890.1621915.

11. Open Virtualization Format (OVF) Specification. DSP0243 1.0.0.Distributed Management Task Force. Feb 2009. http://www.dmtf.org/standards/ovf; Access date: 2012-01-09.

12. Theilmann W, Yahyapour R, Butler J (2008) Multi-level SLA Management for, Service-Oriented Infrastructures. In Towards a Service-Based Internet, Volume 5377 of Lecture Notes in Computer Science, ed. M¨ honen P, Pohla ¨K, Priol T 324–335. Springer Berlin / Heidelberg.

13. Bernstein D, Vij D (2010) Using Semantic Web Ontology for Intercloud Directories and Exchanges. In International Conference on Internet Computing 18–24.

14. Di Martino B, Petcu D, Cossu R, Goncalves P, M´ hr

T, Loichate M (2011)a Building a Mosaic of Clouds. In Euro-Par 2010 Parallel Processing, Workshops, Volume 6586 of Lecture Notes in Computer Science, ed.

15. Guarracino M, Vivien F, Tr¨ ff J, Cannatoro M, Danelutto M, Hast A, Perla F,a ¨Knupfer A, Di Martino B, Alexander M 571–578. Springer Berlin / Heidelberg, http://dx.doi.org/10.1007/978-3-642-21878-1 70.

16. Petcu D, Craciun C, Rak M (2011) Towards a cross-platform cloud API. Components for Cloud Federation. In 1st International Conference on Cloud Computing & Services Science 166–169.

17. Moscato F, Aversa R, Di Martino B, Fortis T, Munteanu V (2011) An analysis of mOSAIC ontology for Cloud resources annotation. In Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on 973–980.

18. Harmer T, Wright P, Cunningham C, Hawkins J, Perrott R (2010) An application-centric model for cloud management. In Proceedings of the 2010 IEEE 6th World, Congress on Services 439–446. IEEE.

19. Sun YL, Harmer T, Stewart A, Wright P (2011) Mapping Application Requirements to Cloud Resources. In Proceedings of the Euro-Par 2011 Parallel Processing

20. Bernstein D, Vij D (2010) Intercloud Directory and Exchange Protocol Detail using XMPP and RDF. IEEE Services 2010.