2012 IEEE Fifth International Conference on Cloud Computing

Introducing STRATOS: A Cloud Broker Service

Przemyslaw Pawluk, Bradley Simmons, Michael Smit, Marin Litoiu
York University, Canada
{ppawluk,bsimmons,msmit,mlitoiu}@yorku.ca

Serge Mankovski
CA Inc.
serge.mankovski@ca.com

***Abstract*—This paper introduces a cloud broker service (STRATOS) which facilitates the deployment and runtime management of cloud application topologies using cloud elements/services sourced on the fly from multiple providers, based on requirements specified in higher level objectives. Its implementation and use is evaluated in a set of experiments.**

***Keywords*-Cloud Broker Service, Cloud Computing**

I. Introduction

In the evolving cloud ecosystem, it is difficult to determine from which provider(s) a particular cloud resource should be acquired. At present, providers describe their offerings according to a self-defined methodology. For example, Amazon uses what they refer to as Computation Units and number of cores to express the CPU capabilities of its various compute offerings while Rackspace defines CPU as a proportion of the physical host machine. The result is the absence of standardized comparisons among otherwise comparable offerings, which makes deciding on a provider at design time challenging. Once a provider is chosen, a second decision regarding which of their offerings is appropriate must be made.

One solution is to decide on a provider automatically at runtime, instead of making a manual decision at design time. The advantages of delaying the decision include flexibility, portability, and removing the expectation that the software architect be equipped to make the decision. runtime decision making is a natural fit for adaptive systems. For example, at the Adaptive Systems Research Lab, we are constructing a business-driven, cloud management framework that uses models at runtime to determine resource needs and provision them automatically in accordance with requirements and in alignment with business objectives. Presently, we are able to deploy an application environment to a public cloud (e.g., EC2). Associated with this application are a set of models which are used to implement the elasticity policy of its application server tier [1]. The complementary question of from where to source resources is addressed in this paper.

A resource acquisition decision (RAD) problem[1] involves the selection of n resources from a set of m providers such that the deployer's set of constraints, requirements, and preferences (collectively, the deployer's *objectives*) are met (or best approximated). In order to automate resource acquisition decisions (RADs), a broker service layer is envisioned between adjacent layers of the three layered (i.e., IaaS, PaaS and SaaS) cloud architecture [2]. This broker is used to aggregate knowledge about various services offered by all providers from the subordinate architectural layer and provide a unified interface / API.

Ideally this architecture will allow higher-level specification of objectives that can be evaluated in a standardized, cross-provider manner. A consortium, led by Carnegie Mellon and CA, has developed the Service Measurement Index (SMI) as a possible approach to facilitate the comparison of cloud-based business services[2]. The SMI framework is a hierarchical partitioning of a service's description into seven categories (i.e., accountability, agility, assurance, financial, performance, security and privacy and usability) with each category being further refined to a set of attributes [3]. An attribute is then expressed as a set of key performance indicators (KPIs) which specify the requisite data to be acquired for every measure / metric [3].

In this paper we introduce an initial version of our broker service layer, the cloud broker service STRATOS[3] which represents an initial step toward the automated cross-cloud resource provisioning and *intercloud* platform envisioned by [4]. While abstraction layers to IaaS already exist, their focus is on delaying choosing a cloud provider from design/develop time to deployment time, with the ability to change the decision with limited development effort. We automate the decision entirely, and move the decision point from deployment time to runtime. STRATOS allows the application *deployer* to specify what is important to them in terms of KPIs so that when a request for resource acquisition is made it is able to consider the request against all providers' offerings and select the acquisition that is best aligned with the objectives. Some experiments are presented that demonstrate how cross-cloud distribution of an application can decrease the cost of the topology and create one that is better fitted to the deployer's objectives. We conclude with a discussion exploring the design issues and challenges.

II. Designing a Cloud Broker Service

We will use the following scenario as a running example throughout this paper. An application provider intends to deploy a stateless, multi-tiered application environment to the cloud; there are two cloud providers, P_A and P_B. There are two situations where a RAD problem is encountered: (i) Initialization of the application environment topology and (ii) at runtime whenever a change in resource allocation is

[1]Although we use the term acquisition, our solution also addresses adaptively releasing resources.

[2]http://beta-www.cloudcommons.com/web/cc/about-smi

[3]The stratosphere is the second major layer of Earth's atmosphere, directly above the troposphere where almost all clouds exist.

978-0-7695-4755-8/12 $26.00 © 2012 IEEE
DOI 10.1109/CLOUD.2012.24

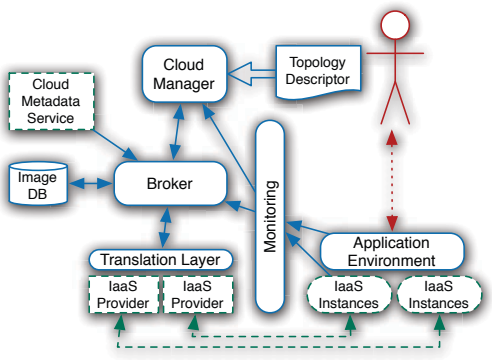


Fig. 1: High-level architectural view of the cloud management framework. The Application Manager is not shown.

determined by the elasticity policy of the application server tier. After the initial deployment, the decision to add or remove resources is made by a *Cloud Manager*. There is an *Application Manager* which controls the runtime management of the application according to models (i.e., layered queueing model, linear models, policy rules, etc.). The latter two components are described in more detail in [5] but will be introduced briefly here. The *Broker* is responsible for solving the RAD problem; it must also connect (automatically) with the set of selected providers to be used and acquire/release the collection of resources.

Consider the architecture presented in Figure 1. A topology document, referred to as a topology descriptor file (TDF), is used to define the application topology to be deployed on the cloud (§II-A). The *deployer* specifies all the details of its application environment in this document[4]. Details include structural concerns (e.g., numbers of tiers in the application, numbers of nodes in each tier, etc.), monitoring directives (e.g., which metrics to monitor and how often), management directives (e.g., a set of models to control the elasticity policy of the application server tier at runtime), and the deployer's objectives.

Upon receiving the topology document, the Cloud Manager contacts the Broker to instantiate the topology. The Broker performs the initial RAD calculation, namely the most *efficient* allocation of resources across the two providers. *Efficient* is an intentionally non-specific term: it determines, based on the specifications in the TDF, whether the entire topology should be built on P_A (or P_B) or whether it should be partially sourced from both (and in what precise ratio). Further, it selects the specific configuration of nodes (i.e., CPU, RAM, disk) from the set of all available configurations offered by each provider. These configurations and their properties are obtained from a third-party API (e.g. CloudHarmony or CloudyMetrics [6]). The chosen nodes are instantiated through a translation layer (which allows the Broker to communicate with either provider and the instances created by each). The CloudManager and Broker make use of monitoring information, the former to make ongoing elasticity decisions (via the Application Manager) and the latter to assist in the decision process.

The current implementation requires that the deployer create accounts with each provider and include that information at deployment time. The deployer must also specify which images the broker should use for the required nodes; the broker can deploy and configure these images based on the TDF. The broker maps the provider-specific images to a set of abstract identifiers used in the TDF. To add a new provider to the broker, the deployer provides account details and a mapping from the provider-specific image identifiers to the abstract image identifiers[5].

The remainder of this section discusses the broker solution in more detail: specifying the problem in a TDF, specifying higher-level objectives, deciding on what resources to acquire and acquiring those resources and making them available.

A. Topology

A topology represents a managed application environment. An application is composed of a set of clusters and a cluster is composed of a set of nodes. A node is a representation of a virtual machine instance characterized by various factors such as hardware specification (CPU, RAM, disk) as well as more abstract things such as performance, security, etc. Nodes in the same cluster have identical function, for instance one cluster may contain web servers while another contains database servers. Containers can be deployed to nodes. A container represents software that provides a place for services to run (e.g. Tomcat). A service is then deployed into a container (e.g. web service deployed in Tomcat). The TDF describes this structure in XML.

Figure 2 shows a small snippet of the TDF. Lines 3-19 describe a simplified topology (with some clusters omitted), including a web host with a Tomcat container and two services: Simple_Application and SNMP. The application is connected to a MySQL cluster and Tomcat is connected to a Load Balancing cluster. The node is expected to generate 500GB of outgoing bandwidth. Lines 22-32 describe the broker input: configuration *small* described by three properties (CPU, RAM, disk), and the objective function definition pointing to the class that implements the function.

B. Defining Objectives

The objectives represent the constraints, requirements, and preferences established by the deployer. They are specified as objective functions; the broker model allows for any implementation of an objective function.

In order for any comparison to be performed, a set of measurable indicators must be defined. It is also important that these measurements can be normalized in order to compare various providers, as most use self-defined descriptors to describe their offerings. The Service Measurement Index (SMI)

[4]As the broker is further developed, we hope to reduce the amount of information that must be specified manually in favor of automatic determination.

[5]The current implementation also requires manual addition of available configurations for the new provider.

```
<configuration>
  <topology name="Awesome Cloud" id="topology1">
    <cluster name="Web Cluster" id="Cluster_Web_1">
      <node name="Web Host" id="" type="worker" config="
          small"
        img="img-ef38fa86" publicIP="" privateIP="" region=
            "us-east-1d" band-in="" band-out="500">
        <container name="Tomcat 6" id="">
          <service name="Simple_Application" id="tom1" />
          <service name="SNMP" id="snmp" />
        </container>
      </node>
    </cluster>
    <!-- additional clusters omitted -->
    <dependencies>
      <!-- Connect Simple App to MySql -->
      <dependency from="tom1" to="mysql1" />
      <!-- Connect Load Balancer to Tomcat -->
      <dependency from="bal1" to="tom1" />
    </dependencies>
  </topology>

  <broker>
    <configurations>
      <config name="small" id="small">
        <property name="disk" value="160"   unit="GB"/>
        <property name="cpu" value="2" unit="units"/>
        <property name="ram" value="4" unit="GB"/>
      </config>
    </configurations>

    <objective name="cost"
        class="strato.broker.objectives.CostObjective" />
  </broker>
</configuration>
```

Fig. 2: Selected Topology Description File content.

offers one promising approach to business service comparison; however, mappings from higher-level characteristics to lower-level measures (KPIs) are not well defined. Garg [7] attempts to define a basic set of metrics to be used for cloud services comparison. Similarly, Zachos et al. [8] provide an alternative set of metrics. In this work we use KPIs defined in their work and extend this set when necessary.

The experiments are focused on two particular objectives: cost and avoiding lock-in. Lock-in is measured using the balance of the topology across the providers; as an objective, avoiding lock-in represents the desire to have an entire application running with a single provider.

Cost is a more complex objective, as there are many factors involved in calculating cost: instance type, length of time, hourly price, spot versus on-demand, bandwidth, storage, and more. To demonstrate the flexibility of the cost objective, we conduct experiments that include only the hourly price, as well as experiments that consider the bandwidth cost. Since bandwidth outside the cloud provider's data center incurs charges, the objective to minimize bandwidth cost is in direct conflict with the objective of avoiding lock-in; the broker achieves a balance of these opposing objectives.

Assuming on-demand instances, the hourly prices are published and change rarely, and are therefore easily calculated. The cost of bandwidth involves three factors: *(i)* providers' pricing models, *(ii)* application's communication patterns and *(iii)* distribution of nodes over providers. Although pricing models vary (some providers charge a consistent price, while others offer volume discounts), once known the cost is easily calculated.

The expected communication patterns of the application are specified in the TDF[6]. The expectations/approximations are updated at runtime based on measured communication patterns. Both approximating and measuring requires consideration of several factors:

1) How much data do the end users receive from the application?
2) Is the data distributed equally among the nodes?
3) Do the nodes within a cluster exchange data (e.g. database synchronization)?
4) What volume of information is transmitted among the clusters (e.g. between application nodes and database node)?

The nodes can be distributed over the providers in various combinations; each distribution has bandwidth cost implications. If the application is deployed to a single provider, the bandwidth cost is based primarily on Factor #1[7]. If the topology is distributed over multiple providers, then the cost depends on which communication within the topology crosses the boundaries of a provider. If a three-tier web application were deployed with a load balancer and a database on one provider and all application servers on the other provider, each request would cause data to cross the provider boundaries at least five times (client to balancer, balancer to app server, app server to database, database to app server, app server to client).

In our experiments we consider a simple architecture with clearly defined communication paths (*load balancer* $\leftrightarrow$ *web server* $\leftrightarrow$ *database*), assume equal traffic to all nodes and between all clusters, and no intra-cluster data exchange. Estimation of communication costs for applications that rely on internal communication that is difficult to predict, such as Hadoop, represents a harder problem.

C. RAD Problem: Selecting Resources

The broker requires two pieces of information from the deployer to solve any particular RAD problem: *desired configuration* and a set of *objectives*. A configuration, c_i, is described by a list of properties $(p_1, \cdots, p_n)$, where each property p_i is a triple $(name, value, unit)$. An objective, on the other hand, represents a utility function calculated for a topology, with a configuration being a variable, e.g. cost of the topology can be viewed as an objective to be minimized. The goal of the broker is to optimize the set of objectives (as specified by the deployer in their TDF). By default we use a weighted sum of these objectives; however, in general the method of optimization can be chosen by the deployer.

The selection process occurs in two steps: (i) identification of feasible configurations and (ii) optimization of objectives. The broker selects the set of all configurations that satisfy the objectives specified in the desired configuration named in

[6]We plan to add heuristics that will provide estimates based on the application type in case these values are not provided by the deployer.

[7]Communication within a data center is generally not charged or an order of magnitude less expensive than communication that leaves the data center.

the TDF (e.g., *small*, *large*). This selection defines a space in which we will optimize objectives. Next, as a result of the multi-criteria optimization process, a set of equivalent configurations is selected. From this set one is selected and the appropriate instance is acquired from the provider (the current implementation chooses randomly). In the situation where there are no suitable configurations fitting the objectives, the broker makes an attempt to relax objectives by finding the closest configuration in each direction (property). Next, the optimization step is performed over the resultant set of relaxed results.

The RAD problem can be formulated as a multi-criteria optimization problem [9] as follows. A decision space Ω is a set of all offered configurations c_i. Let c_d denote a desired configuration that is defined by the deployer. The subset of Ω denoted as χ is a feasible set and is defined as

$$\chi = \{c_i \in \Omega | \forall_{p_a \in c_i} \forall_{p_b \in c_d} a = b \Rightarrow p_b \leq p_a\} \quad (1)$$

The objective function denoted as $O_i(T, c_i)$, where T is a topology, is a criteria in the optimization problem. Then the RAD problem can be stated mathematically as:

$$min_{c_i \in \chi}(O_1, \cdots, O_m) \quad (2)$$

To solve this formulated problem, various optimization techniques can be used [9]. Such optimization is performed for each node added to the topology. Let N be a set of nodes to be added, then we perform optimization for each node using weighted sum to solve the optimization problem which gives us the following formula:

$$\forall_{n \in N} \left(min_{c_i \in \chi} \left(\sum_{j=1}^{m} O_j(T, c_i) * w_j \right) \right) \quad (3)$$

where w_j is a weight assigned by the deployer to the objective O_j. In this work we use two objective functions to represent the objectives described in §II-B, cost and lock-in. The cost function is the price of a particular configuration normalized using the maximum price among all offered configurations (the factors involved in calculating the cost vary; as long as the same approach is used for all configurations it is unimportant). Lock-in is defined as follows:

$$L(T, c_i) = \sum_{j=1}^{P} |l_j - n_j| \quad (4)$$

where P is a number of providers, l_j is a desired percentage of nodes that should be acquired from provider j and n_j is actual percentage of nodes acquired from this provider. For our two provider scenario, the deployer may want to distribute their acquired nodes equally among the two providers setting $l_1 = l_2 = 0.5$. The broker when acquiring nodes will then try to minimize L and keep the topology balanced.

TABLE I: Configurations of instances offered by Amazon

Id	CPU	RAM [GB]	Disk [GB]	Price [$ per h]	Platform
m1.small	1	1.7	160	0.085	32bit
m1.large	4	7.5	850	0.340	64bit
m1.xlarge	8	15.0	1690	0.680	64bit

TABLE II: Configurations of instances offered by Rackspace

Id	CPU*	RAM [GB]	Disk [GB]	Price [$ per h]	Platform
256MB	1.6%	0.25	10	0.015	64-bit
512MB	3.1%	0.5	20	0.030	64-bit
1,024MB	6.3%	1.0	40	0.060	64-bit
2,048MB	12.5%	2.0	80	0.120	64-bit
4,096MB	25%	4.0	160	0.240	64-bit
8,192MB	50%	8.0	320	0.480	64-bit
15,872MB	100%	15.5	620	0.960	64-bit
30,720MB	8 cores	30.0	1200	1.800	64-bit

* Each has four virtual cores, with __% of the total core available.

D. Acquisition

To support multiple cloud providers we use δ-cloud [8]. It provides a service that exposes a standard RESTful API to developers, translating calls to its server to the appropriate API calls of the cloud provider. Provider support is supplied by drivers, which allows new providers to be added to our broker easily.

III. Experiments

We implemented the broker using Java, and tested brokering decisions for a simple stateless web application deployed on a standard three-tiered environment consisting of an Apache load balancer, Tomcat application server cluster and MySQL database backend. The experiments are based on the scenario introduced in Section II, using Amazon EC2 and Rackspace as the two cloud services providers. We considered only the Linux configurations shown in Tables I and II and used the bandwidth prices shown in Table III.

A. Experiment One: Cost optimization

This experiment demonstrates that STRATOS is able to optimize for the single objective of cost. The TDF describes two preferred configurations, *small* and *large*, by their respective

[8] http://deltacloud.apache.org/

TABLE III: Cost of the Data Transfer

Threshold	Amazon* [$/GB]	Rackspace [$/GB]
First 1 GB / month	0.000	
Up to 10 TB / month	0.120	
Next 40 TB / month	0.090	0.18
Next 100 TB / month	0.070	
Next 350 TB / month	0.050	
Inter-availability zone	0.01	N/A

*cost of transfer grater than 524TB/month is individually priced

TABLE IV: Desired configurations used in the experiment and the closest configurations offered by Amazon (EC2) and Rackspace (RS) (hardware properties)

Configuration	(CPU, RAM, Disk)	EC2	RS
small	(1,1,160)	m1.small	4,096MB (5)
large	(6,4,160)	m1.xlarge	4,096MB (5)

TABLE V: Cost of the topology ($ per hour) built on Amazon (EC2) and Rackspace (RS) exclusively and cross-cloud using broker (B)

Configuration	EC2	RS	B
small	0.085	0.24	0.085
large	0.68	0.24	0.24
total*	0.935	0.96	0.495

*$total = 3 * small + large$

hardware requirements as shown in Table IV. The load balancer and web server nodes require *small* configuration while the database node requires a *large* configuration. The broker tries to match these desired configurations while satisfying the objectives.

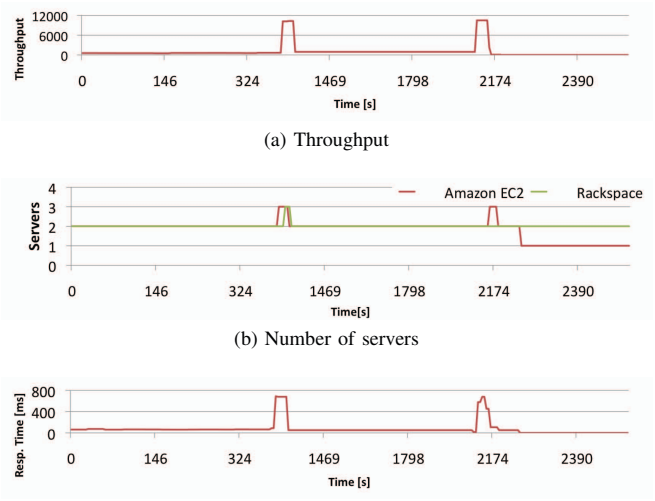
Cost is calculated as the normalized sum of per-hour fees for each instance (the simple cost); the Amazon/Rackspace prices are normalized using a method described in [10]. The cost objective was minimized and the resources acquired; Table V show fees paid by a deployer deploying the same topology to Amazon EC2 or Rackspace exclusively and distributing it as directed by the broker; cost savings of up to 48% are realized when using the broker.

B. Experiment Two: Lock-in and cost

In this experiment we optimize two objectives: cost optimization (the same objective used in previous experiment) and *lock-in minimization* which aims to distribute acquired nodes among available providers. In this experiment we distribute nodes equally; however it is possible to change the settings and choose preferred providers if desired. As a result of this additional objective all nodes were distributed over the two providers equally. Specifically, the load balancer was deployed on an EC2 m1.small node, the web servers were deployed one on each provider and the database server was deployed to Rackspace. When adding an additional web server node at runtime it was deployed to EC2 because the topology must remain balanced and configuration m1.small is less expensive than the corresponding configuration (4GB) from Rackspace. The cost of the topology (per hour) in this case is higher than when optimizing based on cost only ($0.64) but the desired property of splitting services among providers is met.

C. Experiment Three: Online Provisioning

In this experiment we allow the policy engine component (of the Application Manager) to make decisions in response to workload. The decision to *add a node* or *remove a node* is taken based on performance metrics, which in this implementation was the mean response time over the application server



(a) Throughput

(b) Number of servers

(c) Response Time

Fig. 3: Adding and removing nodes at runtime.

cluster. We use a combination of two constantly increasing workloads: the first starts by generating 200 users and adds 10 users per iteration (every 3 minutes) and after seven iterations decreases number of users the same way. The second workload, that runs from from 5th minute to 20th minute and 30th to 45th min of the test, is used to generate spikes in the workload adding an additional 500 users and then another 10 per iteration. Users send mixed select and insert requests with randomized think time between consecutive calls. The decision *where to provision* is made by the broker based on the specified objectives, for this experiment including the simple cost and lock-in.

We ran two series of experiments. In the first series we used only hardware properties to select a configuration. In this case one of our workloads was capable of producing 70% CPU utilization on the EC2 instance with m1.small configuration, while the corresponding instance on Rackspace handled similar workload utilizing about 15% of the CPU. We hypothesize that it is the result of a difference in the number of cores (m1.small has one core while Rackspace 4GB instance has 4 cores) and the potential to use more CPU time when available. The difficulty of directly comparing even two providers strengthens the argument for a better way of expressing requirements and capabilities (e.g. through benchmarking).

In the second series we have used a WEB benchmark acquired from Cloud Harmony[9] to compare configurations in the selection process. Configurations used in this experiment are shown in Table VI. It allowed us to acquire instances that have comparable power and more predictable behaviour.

Figure 3 shows the results of the second series of this experiment in the form of a graph of the throughput, the number of application servers deployed on each provider

[9]http://cloudharmony.com/benchmarks

TABLE VI: Desired configurations used in the experiment and the closest configurations offered by Amazon (EC2) and Rackspace (RS) (benchmark results)

Configuration	WEB Bench.	EC2	RS
small	25	m1.small	1024MB (3)
large	82	—*	2,048MB (4)

* The largest considered configuration from Amazon EC2 (m1.xlarge) received 66 points in the benchmark which does not satisfy the requirement

and the response time. In this experiment, objectives are maintained when adding the nodes[10]. STRATOS maintains the balance of the topology by adding nodes from both providers; however, when the topology is balanced it prefers a less expensive provider.

D. Experiment Four: Including Bandwidth Cost

In this experiment we include bandwidth charges (Table III) in the cost function calculation[11]. Although the EC2 costs decrease the more bandwidth you use (after the first 1GB), our application does not approach the 10TB threshold to receive reduced pricing, so the two cost models are the same in practice (though with differing prices).

We did some initial benchmarking of the application to create an initial cost approximation, based on the factors described in §II-B. We assume 20KB per request per tier (application and database tiers), 1000 requests per hour, evenly balanced among all nodes. The estimate is 15GB per month from the database server, another 15GB from the application which is replicated by the load balancer for another 15GB. These totals include 1GB of monitoring data.

Using the same weights as in experiment three (lock-in and cost are equally important), the addition of bandwidth to the cost calculations did not affect the broker's decisions substantively. The bandwidth cost is low and does not substantially skew the total cost, and lock-in is considered important. However, as the weight associated with the cost is increased, a single provider (in this configuration Amazon EC2) is preferred and more instances are deployed to the Amazon EC2 cloud.

E. Startup Time as an Objective

In the course of our experiments, we found that different providers had different startup times due to the different virtualization technology in place. We conducted an experiment to systematically assess the startup time of Rackspace and Amazon EC2. We started 10 instances of each configuration (one at a time), using δ-cloud to access both providers so the overhead introduced by this additional layer is present in all cases. We polled every 30s to see if the instance was active (the interval was chosen based on our observations and is a trade-off between prompt notification and respecting rate limits). The results are shown in Figure 4: the startup time of the

[10]Nodes are removed in the order they have been added

[11]We exclude the price of inter-availability zone bandwidth as it is substantially lower in cost than outgoing bandwidth.

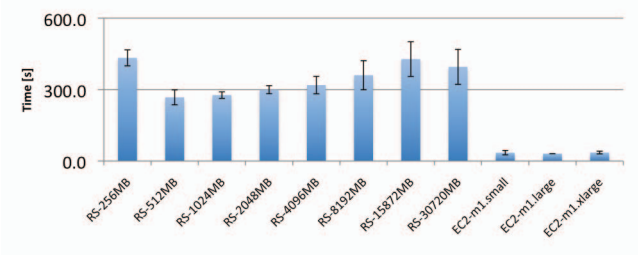


Fig. 4: Startup time of various sizes of instances on both Amazon EC2 and Rackspace.

Amazon EC2 instances is an order of magnitude lower than Rackspace instances.

In experiments 1-4, we configured the Application Manager to make decisions earlier to accommodate longer startup times. This is an imperfect solution, as it results in over-provisioning when using Amazon instances. A next step is to include provisioning time in the online decision-making process; for instance, the broker could decide to acquire instances that will start up faster to respond to rapidly changing conditions, but have more freedom to choose among providers when responding to a gradual trend or predicted load.

IV. Related Work

The term *intercloud* refers to the notion of a cloud of clouds [4]. STRATOS has been designed as an initial step toward this ideal. While there exist many cloud service providers on the market (e.g., Amazon Elastic Compute Cloud, Rackspace, Microsoft Windows Azure, Google AppEngine, Eucalyptus, or GoGrid[12]), none offers a common platform for cooperative cross-cloud usage.

The broker service is a component of an adaptive management system. Such systems [11]–[14] have previously focused solely on automating the provisioning of resources from (and releasing them to) a single cloud provider. Further, in many cases, the acquired resources are assumed to be homogeneous in nature. STRATOS removes this assumption of acquired resource homogeneity and represents an initial attempt at facilitating automated cross-cloud resource provisioning and hence, the realization of an actual intercloud platform.

The RAD problem has been described in [15] in the context of migration of the enterprise into the cloud. They provide limited support for building a topology using a static approach and do not consider the evolution of the topology over time. Their approach requires significant knowledge of the application.

Han et al. [16] describe a recommendation system (RS) for cloud computing. This approach is most suitable for design-time decisions as it is used statically to provide a ranking of available cloud providers. We employ runtime online decision making.

[12]Respectively, http://aws.amazon.com/ec2/, http://www.rackspace.com/cloud/, http://www.windowsazure.com/en-us/, http://code.google.com/appengine/, http://www.eucalyptus.com/, http://www.gogrid.com/.

The need for interoperability and intercloud protocols has been advanced by [17], while [18] presented work on a vision of a utility-oriented federation of cloud computing environments. While both are interesting, neither present an implementation nor experimental results. Projects like δ-cloud, whose ultimate goal is to provide unified access to multiple clouds, offer a unified API which can be viewed as a productive first step toward the realization of the intercloud vision. They do not provide any mechanisms to automate the resource acquisition process and offer advice only.

Service measurement problem has been explored extensively in the context of web services and QoS [19]–[21]. Aggarwal et al. [19] consider measurement specifically in the context of web service composition. Measurement of cloud services is a fairly new area of research. Significant work on systemizing measurement and comparison methods for cloud services is offered by the SMI project described in the introduction and in §II-B. SMI is proposed as a tool for decision makers and managers and as such represents a business point of view rather than a technical one and proposes multiple soft criteria that are hard to measure. Garg [7] attempts to define a basic set of metrics to be used for cloud services comparison. Similarly, Zachos et al. [8] provide an alternative set of metrics.

There have also been attempts to measure cloud service performance [22]–[24]. In all cases services offered by Amazon EC2 have been compared with services offered by other providers: Rackspace [22] and GoGrid [24]. In this work we use KPIs and metrics proposed in the earlier work, but we do not use hardware properties to normalize as done in [7]. The idea of using benchmarks to normalize configurations is new and has not been found in the literature.

Armbrust et al. [25] point to several obstacles and opportunities in cloud computing. They consider availability as one of the top obstacles and analyze high-availability mechanisms used by chosen providers. While considering the opportunity to use multiple providers, they do not refer to any particular solution allowing for such cross-cloud provisioning to be performed.

The Aeolus project[13] is an open project built on top of δ-cloud that offers an interface and tools for cross-cloud acquisition and instance management; however, it "lets a [deployer] make an informed choice of which of the available clouds to use" while STRATOS makes that decision for the deployer.

V. Discussion

To reach our goal of automatic cross-cloud resource acquisition, we must address several challenges. In this section, we outline the main research challenges going forward, each of which we are actively developing.

a) Service measurement and comparison: Measurement methods and metrics are important aspects of cross-cloud acquisition since they facilitate comparison. Service measurement is a vital part of this research. We intend to explore in much greater detail the notions of categories, attributes and KPIs of the SMI including the various relationships and dependencies among them. In our experiments we have shown that there is a need for characteristics expressing more abstract and complex metrics than hardware specification (e.g., CPU, RAM and disk). The deployer needs to be able to express their requirements in terms of expected performance rather than hardware specification. Even such simple measure like benchmark result gives better results than hardware specification. Such characteristics will improve the optimization and allow the broker to find better configurations that may result in more homogeneous clusters that are easier to model. There is also work required in other areas of service measurement such as security, accessibility etc. Those properties are harder to measure hence they are also harder to compare.

b) Decision making and optimization: Both multi-criteria optimization and requirements relaxation are necessary for the function of the broker. The efficiency of the entire solution depends upon the efficiency of the decision making process. Moreover, as a result of optimization we may receive a set of equivalent (in terms of satisfaction of requirements and objectives' values) configurations. This set can be further refined to optimize the cost-benefit ratio and provide superior solutions.[14]

c) Automated application-driven acquisition: It aims to minimize the human factor and include mechanisms for applications to acquire resources on-demand without intervention. The decision on which configuration to acquire and from whom is dependent on the deployed application. Such application-driven deployment can be done either by using templates based on the application type (e.g. hadoop, e-commerce, etc) or as a result of *just-in-time benchmarking (JITB)* – a process in which the system runs benchmarks for a specific application and based on the results a decision is taken. The later approach requires effective deployment, benchmarking and assessment methods. At this point of time the deployment process can be fairly easily automated, yet JITB and assessment remain an unexplored area. JITB in this context is not standard procedure used to assess hardware or virtual instance performance, but rather an approach to asses the performance of such instance running a specific application.

d) Issues of model design: The modelling issues need to be explored. We elected to utilize both an Application Manager and a Broker in order to separate the decision making logic into two steps. The first step in the process, handled by the application manager, determines the number of resources required at any point in time while the second step, addressed by STRATOS determines from where to procure these resources. This separation may be incorrect and needs to be

[13]http://www.aeolusproject.org/

[14]Selected configurations are equivalent but not equal, especially when a subset of available properties is considered. In such situations the broker can refine the optimized set based on, for example, remaining properties and heuristics.

explored further. Specifically, what is the impact on adding and removing non-homogeneous resources (i.e, resources sourced from multiple providers) on the correctness of the models guiding the elasticity policy of the various application tiers? It is unclear at present how negative this de-coupling is in terms of the trade off benefit of ease of understanding; however, much work remains to be done with regards to exploring this design choice.

VI. Conclusion and Future Work

This paper introduces STRATOS, a broker service to facilitate cross-cloud, application topology platform construction and runtime modification in accordance with deployer's objectives. STRATOS represents an initial step toward developing a broker service to facilitate the use of cross-provider cloud offerings. RAD problems were introduced and mapped to a multi-criteria optimization problem. A prototype of STRATOS was described as a component element of a larger autonomic management framework and experiments were presented demonstrating STRATOS's ability to facilitate cross-cloud resource acquisition in accordance with deployment directives. The importance of mechanisms to compare and normalize offerings from multiple providers was also emphasized by this work.

Our future work includes testing STRATOS in more complex settings that include consideration of various SMI attributes (e.g., QoS characteristics). We also intend to test STRATOS with different objectives such as availability and/or latency. Eventually, we would like to eliminate the need for configuration specifications and move toward an application-driven resource acquisition process. This requires the definition of template configurations for different application types and mechanisms to select these template based on JITB.

Interesting part of the CMO that we want to explore in the future is identification of KPIs. Work in this area is partially done and has to be continued. Moreover, in the wider context of intercloud management system, the verification of the approach is required. It will be especially interesting to see if broker's decisions fit into deployer's expectation and how JITB and CMO can help us to understand the decision process.

Acknowledgment

This research was supported by CA Inc., the Natural Sciences and Engineering Research Council of Canada (NSERC), Ontario Centre of Excellence (OCE) and Amazon Web Services (AWS).

References

[1] H. Ghanbari, B. Simmons, M. Litoiu, and G. Iszlai, "Exploring alternative approaches to implement an elasticity policy," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, 2011, pp. 716 –723.

[2] P. Mell and T. Grance, "The NIST definition of cloud computing (draft)," NIST special publication 800-146, 2011.

[3] CSMIC, "Service measurement index version 1.0," Carnegie Mellon University Silicon Valley, Tech. Rep., 2011.

[4] K. Kelly, "A cloudbook for the cloud," http://www.kk.org/thetechnium/archives/2007/11/a_cloudbook_for.php, 2007.

[5] C. Barna, M. Litoiu, and H. Ghanbari, "Model-based performance testing (NIER track)," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: ACM, 2011, pp. 872–875.

[6] M. Smit, P. Pawluk, B. Simmons, and M. Litoiu, "A Web Service for Cloud Metadata," in *Services (SERVICES), 2012 IEEE World Congress on*, June 24-29 2012.

[7] S. K. Garg, S. Versteeg, and R. Buyya, "SMICloud: A framework for comparing and ranking cloud services," in *Proceedings of the 4th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2011, IEEE CS Press, USA), Melbourne, Australia*, December 5-7 2011.

[8] K. Zachos, J. Lockerbie, B. Hughes, and P. Matthews, "Towards a framework for describing cloud service characteristics for use by chief information officers," in *Requirements Engineering for Systems, Services and Systems-of-Systems (RESS)*, 2011, pp. 16 –23.

[9] M. Ehrgott, *Multicriteria Optimization (2. ed.)*. Springer, 2005.

[10] Y. Balasko, *General Equilibrium Theory of Value*, ser. Princeton University Press. Princeton University Press, 2011.

[11] W. Iqbal, M. Dailey, and D. Carrera, "SLA-driven adaptive resource management for web applications on a heterogeneous compute cloud," *Cloud Computing*, pp. 243–253, 2009.

[12] M. Litoiu, C. M. Woodside, J. Wong, J. Ng, and G. Iszlai, "A business driven cloud optimization architecture," in *SAC*, 2010, pp. 380–385.

[13] C. Barna, B. Simmons, M. Litoiu, and G. Iszlai, "Multi-model adaptive cloud environments (MACE)," November 2011, http://www.ceraslabs.com/projects/management-services-for-cloud-computing.

[14] B. Simmons, M. Litoiu, D. Ionescu, and G. Iszlai, "Towards a cloud optimization architecture using strategy-trees," in *Proceedings of The 9th International Information and Telecommunication Technologies Symposium (I2TS 2010), 13-15, December, Rio de Janeiro, Brazil*, December 2010.

[15] A. Khajeh-Hosseini, I. Sommerville, J. Bogaerts, and P. B. Teregowda, "Decision support tools for cloud migration in the enterprise." in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, L. Liu and M. Parashar, Eds. IEEE, 2011, pp. 541–548.

[16] S.-M. Han, M. M. Hassan, C.-W. Yoon, and E.-N. Huh, "Efficient service recommendation system for cloud computing market," in *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, ser. ICIS '09. New York, NY, USA: ACM, 2009, pp. 839–845.

[17] D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond, and M. Morrow, "Blueprint for the intercloud - protocols and formats for cloud computing interoperability," in *Proceedings of the 2009 Fourth International Conference on Internet and Web Applications and Services*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 328–336.

[18] R. Buyya, R. Ranjan, and R. N. Calheiros, "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services," in *ICA3PP (1)*, 2010, pp. 13–31.

[19] R. Aggarwal, K. Verma, J. Miller, and W. Milnor, "Constraint driven web service composition in METEOR-S," in *Proceedings of the 2004 IEEE International Conference on Services Computing (SCC 2004*, 2004, pp. 23–30.

[20] Y. N. Li, K. C. Tan, and M. Xie, "Measuring web-based service quality," *Total Quality Management*, vol. 13, no. 5, pp. 685–700, 2002.

[21] A. E. Saddik, "Performance measurements of web services-based applications," pp. 1599–1605, 2006.

[22] "Rackspace cloud servers versus Amazon EC2: Performance analysis," http://www.thebitsource.com/featured-posts/rackspace-cloud-servers-versus-amazon-ec2-performance-analysis/, 2009.

[23] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: comparing public cloud providers," in *Proceedings of the 10th annual conference on Internet measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 1–14.

[24] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. H. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, pp. 931–945, 2011.

[25] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A Berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, 2009.