Paradigm-Based Adaptive Provisioning in Virtualized Data Centers

Rafael Pereira Esteves*, Lisandro Zambenedetti Granville*, Hadi Bannazadeh†, Raouf Boutaba‡§
* Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS)
Av. Bento Gonçalves, 9500, Porto Alegre, Rio Grande do Sul, Brazil, 91501-970
† Department of Electrical and Computer Engineering, University of Toronto
10 King's College Road, Toronto, Ontario, Canada, M5S 3G4
‡ D.R. Cheriton School of Computer Science, University of Waterloo
200 University Avenue West, Waterloo, Ontario, Canada, N2L 3G1
§ Pohang University of Science and Technology (POSTECH), Pohang 790-784, Korea

***Abstract*—Virtualized data centers host multiple applications with distinct objectives in a shared infrastructure. Accommodating several dynamic applications in virtual data centers is a challenging task for cloud providers. Current provisioning solutions focus on a limited set of objectives that may not be suited for the increasing number of applications deployed in data centers everyday. In this paper we propose an adaptive provisioning architecture for virtualized data centers based on allocation paradigms. A paradigm translates high-level application goals to objectives, allocator instances, and actions that actually provision customized virtual infrastructures to applications. A paradigm policy language is defined to express the relationship between paradigms, objectives, and actions. A performance evaluation of the proposed approach considers four main aspects: acceptance ratio, provisioning cost, and CPU and link utilization. Simulation results show that our proposal is able to select the most appropriate set of allocation actions based on the particularities of the applications.**

I. Introduction

Modern data centers are aggregates of computing, storage, and networking resources deployed to support compute-intensive applications and large-scale storage. Companies such as Google, Amazon, Facebook, and Microsoft rely on data centers to support a variety of services such as Web search, e-mail, social networking, and e-commerce. With the increasing popularity of cloud computing, data centers become even more important by forming the core of advanced cloud environments. Since building and maintaining large data center architectures is expensive, cloud platforms such as Amazon EC2 [1] and Windows Azure [2] offer data center resources to external customers to run a variety of applications. By leasing physical infrastructure to external customers, infrastructure providers (InPs) encourage the development of novel services and, at the same time, generate revenue to cover deployment and operation costs of data center infrastructures.

To allow multiple customers (or tenants) to share a data center, infrastructure providers rely on virtualization technologies (*e.g.*, VMWare, Xen, QEMU) to build *virtual infrastructures* (VIs) composed of logical instances of physical resources (*e.g.*, servers, network, storage). The provisioning of VIs must consider requirements of both infrastructure providers and tenants. While the main objective of infrastructure providers is to generate revenue by accommodating a large number of VIs, tenants, in their turn, have specific needs, such as guaranteed bandwidth among virtual machines, load balancing, and high availability. Inefficiencies in the provisioning process can lead to disastrous consequences for infrastructure providers including reduced number of tenants, money penalties when SLAs are not satisfied, and low utilization of the physical infrastructure.

Current cloud provisioning systems allow tenants to select among different resource configurations (*e.g.* CPU, memory, disk) to build a VI. The tenant is the main responsible for choosing the resources that will better fit his/her applications. The infrastructure provider, in turn, either allocates resources for the VI on the physical data center or rejects the allocation if there are not enough resources to satisfy the tenant's request. Infrastructure providers run allocation algorithms to find the best way to map virtual infrastructures onto the physical substrate according to well-defined objectives, such as minimizing the allocation cost, reducing energy consumption, or maximizing residual capacity of the infrastructure. Mapping virtual to physical resources is commonly referred to as *embedding* and has been extensively studied in the context of network virtualization [3] [4] [5].

The main limitation of current resource allocation schemes in the context of virtualized data centers is that the specifics of the applications to be deployed over a VI are commonly ignored in the provisioning process. Infrastructure providers do not know in advance which applications tenants will deploy over a VI. In a typical IaaS (Infrastructure as a Service) model, if the provisioned VI is not able to achieve the desirable performance, tenants are encouraged to acquire additional resources from infrastructure providers. While the "pay-as-you-go" model [6] for cloud computing works for most applications, there are specific requirements that cannot be satisfied only by adding more resources to a VI. For example, business-critical applications (*e.g.*, ticket reservation, order processing) may require that VM replicas are placed in distinct locations (*i.e.*, different physical servers). On the other hand, network-sensitive applications benefit if VMs are placed in a single machine, to avoid network bottlenecks.

In this paper we address the problem of provisioning VIs considering multiple (possibly conflicting) application requirements to define how virtual resources are mapped in the

978-3-901882-50-0

data center. Adaptive, application-driven resource provisioning allows multiple tenants and a large diversity of applications to efficiently share a data center infrastructure. To enable such flexible resource allocation, we propose a provisioning framework for virtualized data centers that allows tenants to express high-level requirements for the requested VIs, which ultimately influence how VIs should be allocated in the physical substrate. The proposed provisioning approach is based on the concept of allocation *paradigms*. A *paradigm* defines a set of *objectives*, which, in turn, is associated with the high-level goals of the applications. An objective is realized through a set of allocation *actions* performed by an independent *allocator* entity per VI. Based on the performance achieved by applications running inside a data center, an active paradigm may need to be changed in order to meet requirements of all applications. A paradigm can also be modified to adapt to new applications arriving or old ones leaving the data center.

This research is conducted in the context of the SAVI (*Smart Applications on Virtual Infrastructure*) initiative [7], particularly in theme 5 (SAVI Application-Platform Testbed). One of the main objectives of this research theme is to design the control and management planes of a wide-scale testbed and to define strategies to guide resource allocation in large heterogeneous infrastructures composed of integrated wireless/optical access, smart converged edge, wide-area network connectivity, and data centers to support multiple advanced applications and future Internet alternatives.

The rest of this paper is organized as follows. Proposals from the literature related to multi-objective adaptive resource provisioning are discussed in section II. In Section III, a conceptual architecture of a paradigm-based provisioning system for virtualized data centers is described, including its basic concepts, main components, and a strategy to map application objectives to paradigm actions. Simulation results to evaluate the proposed paradigm-based provisioning approach are presented in section IV. Finally, we conclude the paper with final remarks and directions for future work in Section V.

II. Related Work

Multi-objective resource allocation in clouds has been an extensive research topic. Several research have proposed optimization models to find efficient mappings considering requirements of several applications running in the cloud. Other work focus on machine learning techniques to dynamically adjust resource allocation in clouds. In this section, we present the most relevant research carried out so far.

DynaQoS [8] proposes an extended self-tuning fuzzy controller (STFC) as the basis of a QoS provisioning framework for cloud environments that is able to support adaptive multi-objective allocation and service differentiation. DynaQoS is structured in two main layers. The first layer is composed of a set of STFCs (one per objective) and the second layer combines the requests from multiple STFCs and generates a single output to a cloud management system that is responsible for actually perform/modify resource allocation. However, DynaQoS is mainly focused in offering different performance levels in terms of response time. Also, there is limited flexibility for the customer to specify how resources are allocated.

iBalloon [9] uses a reinforcement learning scheme for self-adaptive VM provisioning. In iBalloon, each VM requests adjustments in its capacity in terms of CPU, memory, and I/O, enabling a distributed capacity management framework. The architecture of iBalloon is composed of three main components: host-agent, responsible for allocating resources to the VMs of a physical server; app-agent, which collects real-time information about application performance; and decision-maker, which performs automatic capacity adjustment through a reinforcement learning agent placed at each VM. The main limitation of iBalloon is that it is restricted to the capacity management of VMs and neither consider other resources such as virtual links and storage, nor supports other operations that can help improving application performance (*e.g.*, VM migration).

To cope with the volatility of cloud environments and reduce the resource provisioning time when a burst of requests arrives, Islam *et al.* proposes a prediction model to proactively scale resources to accommodate future requests [10]. The predictive model is based on machine learning techniques such as Neural Network and Linear Regression, and sliding window technique. The disadvantage of the prediction model proposed by Islam *et al.* is that it considers only one type of application (e-commerce) and there is no clear evidence on how the prediction model would perform for several applications having variable workloads. Besides, resource scale up is realized by increasing the number of VMs and there is no fine adjustment on the capacity of a single VM. Furthermore, the model only considers CPU to make a decision and ignores other types of resources, such as storage and network.

CloudOpt [11] is an optimization framework that combines bin-packing, network flows, and mixed integer programming to find efficient applications deployments in the cloud. The goal of CloudOpt is to minimize the cost and energy consumption of an application deployment, offer differentiated performance regarding response time, deploy multiple replicas of processes, and satisfy both CPU and memory constraints. Despite its benefits, CloudOpt lacks important features required in modern data center networks, such as bandwidth guarantees and it does not consider other performance metrics rather than response time.

Frincu and Craciun [12] propose a genetic scheduling algorithm for multi-objective application mapping in multi-cloud environments. The goal is to map applications considering four main aspects: low application running cost, high scalability, load sensitivity, and fault-tolerance. Although, genetic algorithms are well-known for solving multi-objective problems, they usually need a considerable time to converge. For long running applications, such as Web 2.0 applications, solutions based on genetic algorithms achieves acceptable performance. However, in cloud environments short-lived applications are constantly deployed and genetic algorithms may not be the best option for resource allocation.

Current research in adaptive resource allocation in the context of virtualized clouds allow application deployments considering multiple objectives and use different strategies to offer service differentiation. However, previous work share a number of limitations: 1) VMs (and physical servers) are mainly the only type of resource considered in the resource allocation process, network and storage related constraints are

usually ignored and both play a key role in the performance achieved by applications running in the cloud [13]; 2) response time is the main performance metric considered, different applications can have other requirements that are better defined by other metrics (*e.g.,* throughput, availability); 3) allocation algorithms do not allow the human administrator to change the allocation process on-the-fly. This flexibility is important to allow resource provisioning systems to adapt to the dynamics of cloud environments, without having to perform a full reconfiguration, which would be very costly.

III. Adaptive Provisioning Based on Allocation Paradigms

In this section we present the conceptual architecture of an adaptive provisioning system based on allocation paradigms. We first present the basic concepts that are used in the proposed architecture. Next, the architecture itself of the provisioning system is described. We then describe a methodology to map application high-level goals to allocation paradigms and a policy language used to describe allocation paradigms.

A. Basic concepts

A paradigm defines how VIs are allocated in the physical data center. Each paradigm comprises a set of objectives associated with application requirements. An objective is derived in actions, which, in turn, allocate individual VI resources. The main concepts of the paradigm-based provisioning approach are described below.

- **Paradigm:** a paradigm P represents a group of objectives $(O_1, O_2, ..., O_n)$ that are considered in the provisioning of VIs;
- **Objective:** an objective O is a single high-level goal requested by the application owner in the provisioning of VIs, such as "low latency", "resiliency", or "load balancing";
- **Action:** an action A is a single operation executed to achieve an objective O. Examples of individual actions include create VM, migrate VM, install data plane, and increase bandwidth of a virtual link;
- **Window:** a window W is a set of allocation Actions $(A_1, A_2, ..., A_n)$ executed sequentially according to an objective O. A paradigm P is thus realized by a set of windows $(W_1, W_2, ..., W_n)$ associated with the objectives $(O_1, O_2, ..., O_n)$ of the paradigm;
- **Allocator:** an allocator $Alloc$ executes provisioning of VIs through an allocation window W defined by an objective O. There is one allocator $Alloc$ entity associated with each objective O of a paradigm P.

Objectives can be dynamically added or removed from a paradigm. When provisioning VIs, each objective O is translated into a list of actions $(A_1, A_2, ..., A_n)$ that will be executed sequentially within a window W. Objectives reflect distinct allocation policies that coexist together, a characteristic which is not fully supported by current provisioning solutions. Several objectives can be combined together in the provisioning of a VI. The choice for specific allocation actions depends on the characteristics of the applications to be deployed over the requested VI, on the paradigm P that is currently active, and on the current configuration of the physical substrate (*e.g.*, available servers/links). Figure 1 depicts the main concepts of our paradigm-based adaptive provisioning system.

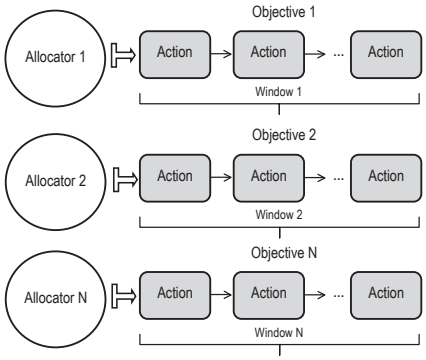
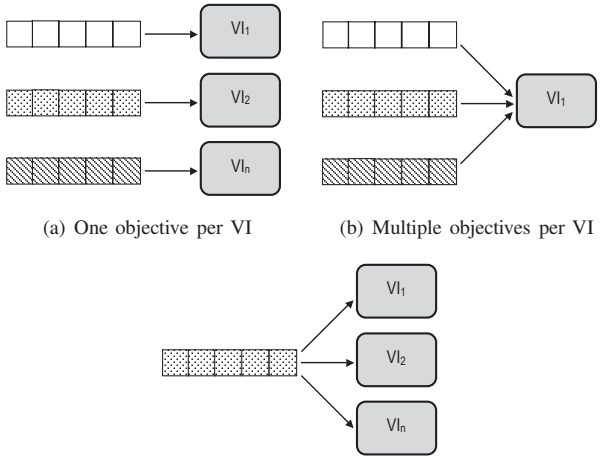


Fig. 1. Allocation paradigms, objectives, and actions

During the operation of a virtualized data center, the current allocation paradigm may need to be modified or another paradigm may need to be selected to adapt the provisioning process to other types of VI requests or to changes in the physical substrate (*e.g.*, new resources that became available after a VI has expired). The decision to switch to another allocation paradigm depends on the effectiveness of the current active one. The effectiveness of an allocation paradigm can be defined in terms of the performance achieved by the applications running over a VI. For example, if a VI is requested to be used by applications requiring 'load balancing' (*e.g.*, Web application) and the current paradigm does not include a corresponding objective, the VI can actually be deployed on a single physical resource, thus reducing the scalability of the application.

It is important to notice that several allocation windows can run in parallel under the same paradigm. In addition, there are three possible ways a window can be associated with a VI. The first and simplest way is to define a separate allocator $Alloc$ for each VI. In the first approach, only one objective is considered in the provisioning (or adjustment) of a VI. The second approach is to use multiple allocators $(Alloc_1, Alloc_2, ..., Alloc_n)$ for a single VI. Using multiple allocators allow diverse objectives to be considered in the provisioning of a VI and, consequently, applications with distinct requirements end up sharing the same VI. The third and last approach is to dedicate one allocator for multiple VIs. A single allocator is useful when similar VIs sharing the same objective need to be managed. Figure 2 illustrates these three allocation approaches.

The motivation behind the use of paradigm windows is to allow the rapid adaptation of a underway allocation to the dynamics of cloud environments. Allocation windows allow, for example, the mapping of virtual machines to be modified on-the-fly to select a physical server that turned out to be a better mapping option for a given objective (*e.g.* reduce number of active physical servers) and that was not available at the time of the first mapping. In most embedding solutions, mapping cannot be changed on the fly, even if appropriate resources become available during the allocation process. The size of the



(a) One objective per VI (b) Multiple objectives per VI

(c) One objective, multiple VIs

Fig. 2. Allocation approaches

paradigm window influences both the ability of an allocation to rapidly adapt to the dynamics of a data center and the signaling overhead required in the selection of the actions of a window, which in the end reflects in the provisioning time. A large paradigm window requires fewer turns to allocate a whole virtual infrastructure, but it is unlikely to take advantage of a better allocation strategy that becomes available. On the other hand, a small paradigm window is more adaptable to dynamic environments like cloud data centers, at the price of higher overhead, which can result in high provisioning times.

Allocation paradigms are not restricted to the initial provisioning of a VI. When a previously allocated VI needs to be adjusted to host new applications or to cope with changes in the high-level goals specified by the application provider, the paradigm performs the necessary actions to reconfigure the VI. For example, if the application provider wants to deploy a new network-sensitive application that is not supported by the active paradigm (because a corresponding objective is not included), a paradigm change is performed to include the new objective (*e.g.*, low latency). The actions of the subsequent allocation windows will reflect the new objective and perform the required actions to modify the previously allocated VI.

B. Conceptual architecture and components

The conceptual architecture of a paradigm-based provisioning system is depicted in Figure 3. The system is structured in four main layers: *Application Layer*, *Operator Layer*, *Service Layer*, and *Infrastructure Layer*. The application layer is the entry point for application providers to request VIs from the InP. In the operator layer, InP operators can define paradigms, objectives, and associated actions. The service layer implements the core logic of the system and comprises allocation and monitoring services. The infrastructure layer consists of a set of *resource agents* that are responsible for device-level resource management. The resource agents also collect device status data that is forwarded to the upper layers.

The application provider requests virtual resources from the physical infrastructure to build VIs using an API supported

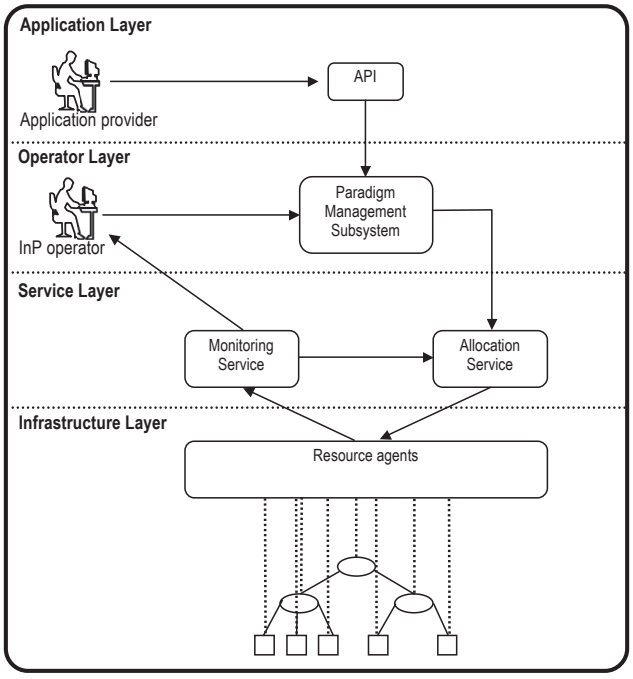


Fig. 3. Architecture of a paradigm-based provisioning system

by the InP. The InP operator defines paradigms using a *Paradigm Management System (PMS)*. The PMS allows InP operators to create new objectives, define the set of allocation actions available to an objective, and determine the current active paradigm. The *Allocation Service (AS)* is composed of allocation instances that are responsible to deploy the active paradigm and execute allocation actions. The AS supports two operation modes: *system-driven* or *human-driven*. In the system-driven mode, the AS automatically determines the set of allocation actions to be executed during the next paradigm window. In contrast, the human-driven mode requires the InP operator to explicitly select the actions to be performed in the window. The *Monitoring Service (MS)* provides updated status of the applications running on the physical substrate to the InP operator to the AS component. The monitored data collected by the MS is used in AS system-driven mode to determine the next set of allocation actions. Similarly, the InP operator can use the information provided by the MS to manually decide upon the actions to be included in the subsequent window.

C. Mapping application goals to paradigms

The choice of a paradigm by the InP operator is influenced by the requirements of the customers of the physical infrastructure (*i.e.*, application providers). However, in virtualized data centers, InP operator has no means to know in advance which applications will run in his/her infrastructure. Application providers usually request a number of virtual resources (*e.g*, VMs) with specific capacities (*e.g.*, high memory, large disk space) and do not specify particular performance objectives. Thus, specific requirements of the application(s) running in a virtualized data center are not considered in resource allocation. Such application-driven provisioning can ultimately improve the overall performance of the services running in

a VI. For example, a VI hosting services that need high reliability can be duplicated, while virtual resources running applications requiring low response times can be placed close to one another.

The application provider must express the requirements of the applications that will be deployed in his/her VI to allow InP operators to change between allocation paradigms and choose the best one. To enable such application-oriented adaptive allocation we argue that the interface between application providers and InP operators should be extended to allow the specification of high-level objectives that need to be fulfilled by the virtual infrastructure. The high-level objectives defined by the application provider will be mapped to the paradigm objectives and actions, which, in turn, actually allocate resources.

In our solution, the application provider can choose among a set of predefined high-level goals those that better fit the application(s) he/she wants to deploy in a VI. The application provider can specifically select, for example, "resilience","processing", and "reserved", among all available goals, to build a VI able to support applications requiring high availability, high computations, and exclusivity over the physical resources, respectively. With this information, the InP operator selects the most appropriate allocation paradigm to satisfy the goals defined by the application provider. If there is no paradigm suitable to handle a specific set of goals, the InP operator will have to either create a new one by defining a policy to allow the system to automatically define the best allocation actions (system-driven mode) or define the allocation actions manually (user-driven mode).

D. Paradigm policy specification

As a part of our paradigm-based provisioning system, we have developed a policy language to allow InP operators to specify the relationship between paradigms, objectives, and allocation actions. Although there are many policies available [14] designed for a variety of purposes, none of them define adequate constructs to allow InP operators expressing allocation paradigms. Our paradigm policies are used by the PMS to automatically define the allocation actions in the system-driven mode. The main constructs of our paradigm-policy language are described below and an example of paradigm policy is given in Figure 4.

- **objective:** an objective o defines a customized allocation objective for a paradigm p. Each objective o has a corresponding *window* attribute w specifying the identifier of the allocation window hosting the actions $(a_1, a_2, ..., a_n)$ associated with the objective;
- **action:** an action a specifies an individual allocation. Each action a is defined by an identifier, a list of conditions $(c_1, c_2, ..., c_n)$ that trigger the action, and an operation that actually implements the action, such as *SelectServer* or *MoveToServer* that are used to place or migrate a VM to a given physical server, respectively;
- **window:** a window w is a logical structure hosting actions that are executed sequentially. A window w has a *size* attribute that defines the number of actions that are executed before the next scheduling. A window also defines the *order* on which the actions are verified in each turn.

```
HSU_LB_LCC {
      objective HighServerUtilization {
            action HSU-CreateVM1 {
                  conditions  = {Num_Allocated_VMs = 0, VMs_toAllocate > 0}
                  operation = {SelectServer : random(all)}
            }
            action HSU-CreateVM2 {
                  conditions  = {Num_Allocated_VMs > 0, VMs_toAllocate > 0}
                  operation = {SelectServer : lowest_residual_capacity(all)}
            }
            action HSU-MigrateVM {
                  conditions  = {VMs_toAllocate = 0}
                  operation = {MoveToServer : lowest_residual_capacity(all)}
            }
            window HSU {
                  size = 5
                  order = {HSU-CreateVM1, HSU-CreateVM2, HSU-MigrateVM}
            }
      }
      objective LoadBalancing {
            action LB-CreateVM1 (HSU-CreateVM1)
            action LB-CreateVM2 {
                  conditions  = {Num_Allocated_VMs > 0, VMs_toAllocate > 0}
                  operation = {SelectServer : highest_residual_capacity(all)}
            }
            action LB-CreateVL1 {
                  conditions  = {Num_Allocated_VLs = 0, VLs_toAllocate > 0}
                  operation = {SelectPath : shortest_path(all)}
            }
            action LB-CreateVL2 {
                  conditions  = {Num_Allocated_VLs > 0, VLs_toAllocate > 0}
                  operation = {SelectPath : highest_residual_capacity(all)}
            }
            window LB {
                  size = 3
                  order = {LB-CreateVM1, LB-CreateVM2, LB-CreateVL1,  LB-
                  CreateVL2}
            }
      }
      objective LowCommunicationCost {
            action LCC-CreateVM1 (HSU-CreateVM1)
            action LCC-CreateVM2 {
                  conditions  = {Num_Allocated_VMs > 0, VMs_toAllocate > 0}
                  operation = {SelectServer : closest_server(all)}
            }
            action LCC-MigrateVM {
                  conditions  = {VMs_toAllocate = 0}
                  operation = {MoveToServer : closest_server(all-used)}
            }
            window LCC {
                  size = 4
                  order = {LCC-CreateVM1, LCC-CreateVM2, LCC-MigrateVM}
            }
      }
}
```

Fig. 4. Example of paradigm policy

In addition to the main constructs, the paradigm policy language defines a set of auxiliary functions to support the main constructs. The auxiliary functions provide updated information about the physical infrastructure and underway requests. Such functions can also refer to individual allocation actions (*e.g.*, VM creation, VM migration). A list of the auxiliary functions of our policy language is given below. The functions provided here are used for a proof of concept. They can, however, be easily extended to include other operations.

- *Num_Allocated_VMs*: returns the number of already allocated VMs of a given VI request;
- *VMs_ToAllocate*: returns the number of remaining VMs of a given VI request that was not yet allocated;
- *UnusedServers*: returns the number of physical servers that do not host a VM of a given VI request;
- *UnusedLinks*: returns the number of physical links that do not host a VL of a given VI request;
- *SelectServer*: maps a VM to a physical server;
- *MoveToServer*: migrates a VM from a physical server to another;
- *SelectPath*: maps a VL to a physical link;

- *random*: returns a random server;
- *lowest_residual_capacity*: returns the server with the lowest residual capacity in terms of a given resource (*e.g.*, CPU, memory, disk) or a combination of the available resources;
- *highest_residual_capacity*: returns the server with the highest residual capacity in terms of CPU, memory, or disk;
- *all*: returns all available servers and links of the physical substrate;
- *used_servers*: returns all servers used to host VMs of a given VI request;
- *used_links*: returns all links used to host VLs of a given VI request;
- *closest_server*: returns the closest server to a given one in terms of number of hops;
- *shortest_path*: returns the shortest physical path for a VL of a given VI request.

In the paradigm policy example of Figure 4 there are three defined objectives : *HighServerUtilization*, *LoadBalancing*, and *LowCommunicationCost*. The HighServerUtilization objective tries to map all VMs to the smallest number of physical servers. If there is not enough capacity to map all VMs of a VI in a single physical server, the AS component finds the physical server with the lowest residual capacity. In contrast, the LoadBalancing objective spreads the virtual resources (*i.e.*, VMs and VLs) among all available physical servers and links. The LowCommunicationCost object is very similar to the HighServerUtilization objective, except that LowCommunicationCost tries to keep VMs of the same VI close to one another, even if they are not sharing the same physical server.

The paradigm policy language is flexible to allow InP operators to describe customized objectives and actions tailored for a variety of VI requests and associated applications. Embedding algorithms and heuristics can be directly translated into objectives and actions. In addition, the paradigm policy language can be easily extended to support other types of constructs and functions. Paradigm policies are used by the AS to automatically schedule actions during the provisioning of a VI (system-driven).

IV. Evaluation

In this section we describe the evaluation scenario and associated results. The goal of this evaluation is to quantify the benefits of our proposed paradigm-based adaptive provisioning approach in terms of acceptance ratio, provisioning cost, and resource (CPU and link) utilization when multiple objectives are considered in VI provisioning.

A. Simulation scenario

We have adapted the discrete-event simulator used by Chowdhury *et al.* [3] [15] to implement the core logic of the paradigm-based VI allocation. The ViNE-Yard simulator was adapted for convenience purposes. It has classes to represent both the physical substrate and virtual requests, and comes with a workload generator. New classes to represent paradigms, objectives, and actions were created and the mapping methods were replaced by the ones defined in the paradigm policies. The physical substrate is represented by a VL2-based [16] topology, which is a well-know data center topology. The physical topology is composed of 24 servers, 22 switches, and 72 links. Bandwidth capacity of links varies according to the type of switch used: 1000 bandwidth units for ToR (*Top-of-Rack*) switches, and 10000 for aggregate and intermediate switches. Server CPU and storage capacities are uniformly distributed between 50 and 100.

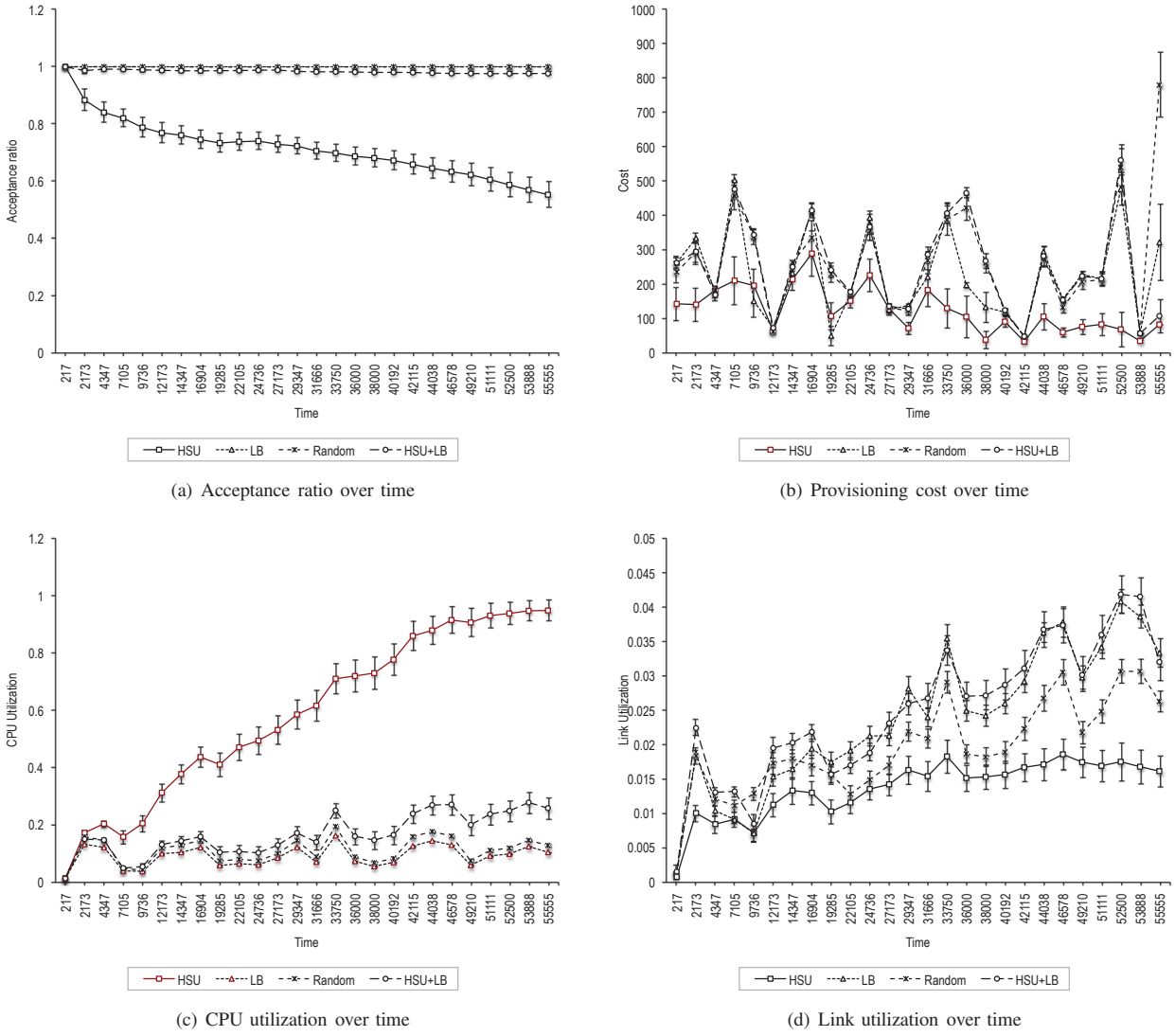
Similar to Chowdhury *et al.* [3], VI requests arrive in a Poisson process with an average rate of 4 VNs per 100 time and an average duration of 1000 time units, exponentially distributed. The number of VMs of each VI request is randomly defined by a uniform distribution between 2 and 10. CPU requirements of VMs are uniformly distributed between 0 to 20, and the bandwidth requirements of virtual links are uniformly distributed between 0 to 50. We have defined three main objectives to compose allocation paradigms: *HighServerUtilization (HSU)* and *LoadBalacing (LB)*, detailed in Section III-D, and a *Random* objective, which tries to map all VMs randomly. The window size is 1 for all paradigms, which means that one action of each objective of a paradigm is executed in each turn.

B. Performance metrics

We have defined four metrics in our evaluation. *Acceptance ratio* is the number of VI requests that are accepted over the total number of requests. *Provisioning cost* is defined in terms of allocated resources and calculated using the model of Chowdhury *et al.* [3]. *CPU utilization* and *link utilization* reflect the average usage of individual servers and links over their total capacity. For each experiment, we evaluate 3 paradigms composed of one single objective (HSU, LB, and Random), reflecting the operation of current provisioning approaches, and a paradigm combining the HSU and LB objectives. Figures 5(a) to 5(d) summarize the obtained results. Each experiment was repeated 30 times with a confidence level of 95%.

C. Acceptance ratio

The average acceptance ratio over time is shown in Figure 5(a). The *LB* objective achieves very high acceptance ratio indicating that distributing VMs over the substrate is better than concentrating requests in a limited subset of nodes. Such behavior can be explained by the fact that LB always try to find the resources with highest residual capacity, which increases the chances of a successful mapping. Randomizing resource allocation also results in high acceptance ratio, comparable to LB. On the other hand, HSU has the worst performance among all paradigms because HSU, in contrast to LB, looks for the nodes with the lowest residual capacity in order to reduce the number of used resources (*i.e.*, minimize resource fragmentation). The side effect is that such nodes run out of capacity very quickly, reducing the mapping possibilities. However, when HSU and LB objectives are combined under



(a) Acceptance ratio over time

(b) Provisioning cost over time

(c) CPU utilization over time

(d) Link utilization over time

Fig. 5. Simulation results: (a) Acceptance ratio. (b) Provisioning cost. (c) CPU utilization. (d) Link utilization

the same paradigm, the acceptance ratio improves considerably, achieving similar performance to LB and Random objectives.

D. Provisioning cost

The average cost of provisioning a VI in terms of allocated resources is depicted in Figure 5(b). Here, LB and Random objectives result in higher provisioning costs compared to HSU. The reason is twofold. First, since LB and Random objectives accept more requests, the number of allocated resources is also higher. Second, LB and Random objectives use more resources when provisioning a single VI (because virtual resources tend to be spread) and more links are allocated. In HSU, in turn, because multiple VMs of the same VI are mapped to a smaller subset of available servers (some can even share the same server), the number of used links is minimized, reducing the overall cost of the VI. Again, when HSU and LB are combined, the provisioning cost approximates to the ones achieved by LB and Random.

E. CPU and link utilization

Figure 5(c) depicts the average CPU utilization over time achieved by the four paradigms. As expected, HSU achieves the highest CPU utilization, since it attempts to increase server utilization mapping VMs on the servers with the lowest residual capacity. LB and Random, in turn, go on the opposite way by choosing resources with high residual capacity and low utilization. It is possible to observe that LB has a big influence in the combined (HSU+LB) paradigm compared to HSU because CPU utilization values are closer to the ones achieved by LB than to the ones obtained when HSU is applied isolated.

Finally, the average link utilization over time is illustrated in Figure 5(d). As stated before, HSU tends to use less links when provisioning VIs, which explains why HSU has the lowest link utilization. By using less links, VIs provisioned with

HSU are "cheaper", as discussed in the cost comparison. The Random objective has slightly superior performance compared to LB and HSU+LB. The similarity between link utilization achieved by both LB and HSU+LB confirms again that LB dominates over HSU, even in the combined paradigm.

F. Summary

Considering the presented results, it is possible to conclude that HSU is better suited for bandwidth-limited scenarios and useful to allocate VIs hosting network-sensitive applications. HSU uses less links, reducing the occurrence of bottlenecks in the network. On the other hand, LB is appropriate for best-effort scenarios and for VIs that need high availability (*i.e.*, VIs hosting critical applications) because resources are spread over the network. LB also results in increased revenue for InPs, since more VIs are allocated when LB is employed (isolated or combined). When HSU and LB are used together, provisioned VIs share characteristics of both objectives, although LB has a larger influence on the achieved performance.

V. Conclusions and Future Work

In this paper, we have proposed a provisioning framework based on the concept of allocation paradigms that enables adaptive allocation in virtualized data centers. Our proposal allows tenants to express high-level requirements for their VIs, which are used by InP operators to define how a VI is allocated on the physical substrate. Furthermore, multiple objectives can be considered when allocating a single VI, enabling multi-tenancy and allowing a large number of diverse applications to coexist in a virtualized data center infrastructure. We have evaluated our proposal in terms of acceptance ratio, provisioning cost, and resource utilization.

Simulation results shows that our proposed paradigm-based adaptive VI enables distinct allocation methods with different objectives to coexist under a single framework. Furthermore, using combined objectives in the same paradigm allows a VI to host diverse applications having different requirements. We could also identify a tradeoff between resource utilization, acceptance ratio, and the provisioning cost of VIs. For example, HSU offers the lowest VI provisioning cost but poor acceptance ratio of VI requests. In contrast, LB and Random improve acceptance ratio at the cost of more expensive VIs. HSU has high CPU utilization and low link utilization what makes paradigms using HSU suited for applications having strict constraints regarding bandwidth and latency (*e.g.*, multimedia). On the other hand, LB facilitates the deployment of applications that benefits from load balancing (*e.g*, critical applications, MapReduce). Randomization is able to achieve performance comparable to the LB objective.

A limitation of the proposal is the fact that there is no control over the objectives included in a paradigm. Since the effectiveness of an allocation paradigm depends on the choices performed at the application layer, the framework cannot predict the performance achieved by the applications before deployment. However, the framework is flexible enough to allow the definition of new allocation paradigms more suitable to a given setting. Besides, at this point, our simulator is not able to classify the paradigms in advance because it would required some feedback from the application provider.

As future work, we intend to evaluate a prototype of the paradigm-based provisioning system in the SAVI testbed, which is currently under development. In addition, we also plan to study the impact of varying paradigm parameters (*e.g.*, paradigm window size) in order to find a better balance between multiple objectives involved in VI provisioning. A model for computation within a VI to compare the effectiveness of different allocation paradigms in terms of application performance will also be considered in future evaluations.

Acknowledgment

This work was supported in part by the Natural Science and Engineering Council of Canada (NSERC) under the Smart Applications on Virtual Infrastructure (SAVI) Research Network, in part by the World Class University (WCU) Program under the Korea Science and Engineering Foundation funded by the Ministry of Education, Science and Technology (Project No. R31-2008-000-10100-0), and in part by the National Council for Scientific and Technological Development (CNPq), Brazil.

References

[1] Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/.

[2] Windows Azure. http://www.windowsazure.com.

[3] M. Chowdhury, M. Rahman, and R. Boutaba, "ViNEYard: Virtual Network Embedding Algorithms With Coordinated Node and Link Mapping," *IEEE/ACM Transactions on Networking*, vol. 20, no. 1, pp. 206–219, February 2012.

[4] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration," *ACM Computer Communication Review*, vol. 38, no. 2, pp. 17–29, April 2008.

[5] X. Cheng, S. Su, Z. Zhang, K. Shuang, F. Yang, Y. Luo, and J. Wang, "Virtual Network Embedding Through Topology Awareness and Optimization," *Computer Networks*, vol. 56, no. 6, pp. 1797–1813, April 2012.

[6] Amazon EC2 Pricing. http://aws.amazon.com/ec2/pricing/.

[7] Smart Applications on Virtual Infrastructure (SAVI). http://www.savinetwork.ca.

[8] J. Rao, Y. Wei, J. Gong, and C.-Z. Xu, "DynaQoS: Model-free Self-Tuning Fuzzy Control of Virtualized Resources for QoS Provisioning," in *Proceedings IWQoS 2011*, June 2011, pp. 1–9.

[9] J. Rao, X. Bu, C.-Z. Xu, and K. Wang, "A Distributed Self-learning Approach for Elastic Provisioning of Virtualized Cloud Resources," in *Proceedings IEEE MASCOTS 2011*, July 2011, pp. 45–54.

[10] S. Islam, J. Keung, K. Lee, and A. Liu, "Empirical Prediction Models for Adaptive Resource Provisioning in the Cloud," *Future Generation Comp. Syst.*, vol. 28, no. 1, pp. 155–162, January 2012.

[11] J. Li, M. Woodside, J. Chinneck, and M. Litoiu, "CloudOpt: Multi-goal Optimization of Application Deployments Across a Cloud," in *Proceedings CNSM 2011*, October 2011, pp. 1–9.

[12] M. Frincu and C. Craciun, "Multi-objective Meta-heuristics for Scheduling Applications with High Availability Requirements and Cost Constraints in Multi-Cloud Environments," in *Proceedings IEEE UCC 2011*, December 2011, pp. 267–274.

[13] C. Guo, G. Lu, H. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "SecondNet: A Data Center Network Virtualization Architecture with Bandwidth Guarantees," in *Proceedings ACM CoNEXT*, December 2010, p. 15.

[14] Policy Languages Review - Policy Languages Interest Group. http://http://www.w3.org/Policy/pling/wiki/PolicyLangReview.

[15] ViNE-Yard. http://www.mosharaf.com/ViNE-Yard.tar.gz.

[16] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," in *Proceedings ACM SIGCOMM*, August 2009, pp. 51–62.