Evaluating Allocation Paradigms for Multi-Objective Adaptive Provisioning in Virtualized Networks

Rafael Pereira Esteves*, Lisandro Zambenedetti Granville*, Mohamed Faten Zhani†, Raouf Boutaba†
* Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS), Brazil
† D.R. Cheriton School of Computer Science, University of Waterloo, Canada

***Abstract*—Recent advances in virtualization technology have made it possible to partition a network into multiple virtual networks managed by different users. Although virtual networks share the same physical infrastructure, they host diverse applications with different goals. Unfortunately, current virtual network provisioning solutions have only focused on achieving a single objective that may not be suited for all the applications deployed across the network.**

In this paper, we propose an adaptive provisioning framework for virtualized networks that takes into consideration the characteristics of multiple applications and their distinct performance objectives. The proposed framework is based on the concept of *allocation paradigm*, which is defined as a set of application-driven provisioning policies that guide the resource allocation process. To determine the efficiency of a particular paradigm, we propose a virtual network performance computation model based on data measured from existing benchmarks. Simulation results show that our model helps network providers to select the best allocation paradigms in terms of provisioning quality.

I. Introduction

Recent advances in virtualization technology have made it possible to partition a physical network into multiple virtual networks used and managed by different users. In a virtualized environment, the Infrastructure Provider (InP) who owns and manages the physical network offers resources to multiple Service Providers (SPs) in the form of Virtual Networks (VNs). Typically, a SP issues a VN request containing the VN specification that includes its topology, the number and capacity of its virtual nodes (*e.g.*, CPU, memory, and disk) and virtual links (*e.g.*, bandwidth and delay). The InP runs then an *embedding algorithm* aiming at allocating resources to the requested VN. The request is whether accepted or rejected depending on the availability of resources.

Traditionally, VN embedding algorithms try to find the best mapping of VNs onto the physical infrastructure such that the objectives of the InP and SPs are achieved [1], [2], [3], [4], [5]. For instance, the InP's goal is usually to maximize the number of embedded VNs in order to increase its revenue, while SPs require guaranteed and predictable performance for their VNs. Despite extensive study on VN embedding problem, there are common limitations shared by most proposals. First, current schemes are static. That is, InPs cannot dynamically change the goal of the embedding scheme and are not able to support specific application requirements, such as fault-tolerance or exclusivity usage of some resources. Second, most VN embedding solutions map all virtual resources of a VN request altogether, *i.e.*, upon receiving a VN request, the embedding algorithm generates as output the complete mapping of all the VN's virtual resources onto their physical counterparts. This can lead to many requests being rejected because it is not possible to embed few of their virtual resources. However, in practice, when previously allocated VNs expire, more resources are released. Therefore, it would be interesting to provide the possibility to embed VNs gradually as more resources may become available in the future.

Furthermore, current VN embedding schemes lack flexibility as they do not take into account possible changes of the InP's objectives, the characteristics of the applications running on the VNs, and also potential changes that could occur to the physical substrate. Consequently, there is a pressing need to design more flexible VN embedding schemes where InP's objectives change over time and applications goals are taken into account. Recent research on adaptive resource allocation has considered the achievement of multiple objectives [6], [7]. However, these proposals are limited to capacity adjustment of individual resources (*e.g.*, virtual servers) and the reconfiguration of previously deployed VNs.

In our previous work, we introduced the concept of *allocation paradigm* to guide resource provisioning in virtualized environments [8]. A paradigm encompasses a set of goals representing the high-level objectives of the InP and the SPs. Each objective is achieved through actions (*e.g.*, allocation) executed within a window (one per goal). The action is defined at run-time according to the objectives and the current status of the substrate network. This approach allows a quick adaptation of the provisioning process to the dynamics of the substrate and to the characteristics of the deployed applications. If some of the targeted objectives are not satisfied or if applications running on a particular VN exhibit poor performance, the current allocation paradigm may need to be changed. Evaluating the efficiency of an allocation paradigm and determining when a it should be revised is a critical challenge that should be addressed.

In this paper, we aim at evaluating the effectiveness of allocation paradigms in terms of provisioning performance.

978-1-4799-0913-1/14/$31.00

We propose a VN computation model that measures the performance of an allocation paradigm based on that of the applications running on the embedded VNs. Our model is based on three main metrics: paradigm quality, provisioning time of VNs, and provisioning cost. These metrics allow to evaluate the performance of paradigms so as to help InPs to better define provisioning approaches.

The rest of this paper is organized as follows. The problem formulation and the computation model are provided in Sections II and III, respectively. The proposed solution is evaluated in Section IV. Finally, we conclude the paper and follow up with future work in Section V.

II. Problem Formulation

In this section we formally define allocation paradigms. We first start by modeling the physical network and VN requests. Next, we discuss the main design aspects of allocation paradigms and provide the problem formulation.

A. Physical Network

We model the physical network as a weighted undirected graph $N^p = (M^p, R^p, L^p, O^p)$, where M^p is the set of physical machines, R^P is the set of physical network elements (*e.g.*, routers and switches), L^p is the set of physical links used to connect physical machines and network elements, and O^p is the set of InP objectives that can be considered during VN provisioning. Each physical machine $m^p \in M^p$ has an associated CPU capacity $c(m^p) \in \mathbb{R}^+$. Each physical link $l^p_{ij} \in L^p$ connecting two physical nodes $i, j \in M^p \cup R^p$ has an associated bandwidth $b(l^p_{ij}) \in \mathbb{R}^+$. An objective $o^v \in O^v$ is one of the objectives that can be chosen by the InP. Each objective is associated with a target index $t(o^v) \in \mathbb{N}$. Possible values that $t(o^v)$ can take are listed in Table I.

TABLE I
Examples of InP's Objectives

Target	Property	Description
0	Green	Virtual nodes and links should be mapped on the smallest set of physical nodes
1	Low latency	Virtual links should be mapped on physical paths with small hop number
2	Load balancing	Virtual nodes and links should be mapped in distinct locations and cannot share the same physical resource
3	Low communication cost	Virtual nodes with more capacity should be placed close to each other

B. Virtual Network Request

In our model, a VN request is defined as a weighted undirected graph $N^v = (M^v, L^v, P^v)$, where M^v is the set of virtual machines, L^v is the set of virtual links, and P^v is the set of properties desired for the applications running on the VN. Unlike a physical network, our model does not define intermediate routers and switches for a VN; virtual links are requested, however, in order to allocate bandwidth between virtual machines. Similar to the physical machine, each virtual machine $m^v \in M^v$ requests an amount of CPU capacity $c(m^v) \in \mathbb{R}^+$, and each virtual link $l^v_{ij} \in L^v$ connecting two virtual machines $i, j \in M^v$ has a bandwidth requirement denoted by $b(l^v_{ij}) \in \mathbb{R}^+$. A property $p^v \in P^v$ is a non-functional requirement defined by the applications running on the VN. Each property has a corresponding target index $t(p^v) \in \mathbb{N}$ that is associated with a high-level requirement requested for the VN. Values that $t(p^v)$ can take are listed in Table II. These values are used as reference. The model can be easily extended to include as many properties as required by the InP.

TABLE II
VN Properties Examples

Target	Property	Description
0	Reliability	Replicas of allocated resources should be placed in different locations
1	Security	A VM should not share the same physical machine of another SP
2	Best-effort	VM can be placed at any location

C. Paradigm Model

An allocation paradigm is a group of provisioning policies that are considered when VNs are provisioned and defines how VNs are allocated in the physical substrate. Each policy is associated with a high-level goal defined by the InP, such as "low latency" or "reliability". Allocation paradigms have four main design characteristics that make them suitable to guide resource allocation in virtualized environments.

Application awareness - Because VNs are hosting applications, the InP needs to define the relationship between VNs and the applications that ultimately run on them. A possible approach is to simplify the problem by allowing only one application per VN. The disadvantage of such approach is the underutilization of resources when the application is not active. On the other hand, deploying multiple applications over a single VN can improve resource utilization at the cost of more complex allocations.

Adaptive provisioning - Typical VN embedding schemes map the whole VN on the physical substrate at once upon receiving a VN request. Mapping all resources of a VN in a single operation is straightforward because the InP has the complete view of the substrate network and the associated capacities. However, some approaches assume that virtual nodes of the same VN request cannot share the samw physical node and have to be mapped at distinct locations [1] [9]. This limitation may reduce the chances of a successful embedding. In addition, most VN embedding approaches do not properly tackle the case where multiple VN requests arrive simultaneously, which is the typical case in realistic scenarios. Therefore, two or more ongoing VN requests can compete for the same physical resource, increasing the chances of rejected VN requests.

To overcome the aforementioned problems and allow rapid adaptation of current provisioning approaches to the dynamics

of the physical substrate, we argue that a VN request should be mapped *in parts*. To realize such concept, we propose the use of allocation *windows* and *rounds*. One allocation window encompasses a fixed number of individual allocation actions defined in real-time by the current allocation paradigm, such as virtual machine creation. The execution of the actions within an allocation window is called a round. Several rounds may be needed in order to complete the full allocation of a VN. In each round, the corresponding window executes the appropriate allocation actions defined by the active paradigm. Figure 1 illustrates how a VN would be mapped using the concepts of windows and rounds. The numbers represent the order in which the virtual resources are allocated. This is determined by the policies defined by in the active paradigm.

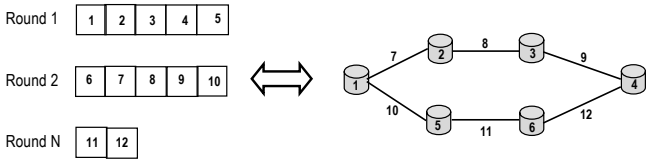


Fig. 1. Allocation windows and rounds

The benefits of this partial and multi-iteration mapping are threefold. First, it allows multiple virtual resources of the same VN request to be mapped on the same physical node. Second, windows allow rapid adaptation to changing network conditions. Two consecutive allocation rounds can result in different mappings compared to mapping all resources at once. For example, after the first round, the mapping of virtual machines can be modified dynamically 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 first round. Finally, a window can run actions from different VN requests making the problem of managing multiple ongoing VN requests more tractable.

Paradigm operations - The InP manager may create, remove, modify, or switch allocation paradigms. The active allocation paradigm may need to be modified by adding or removing policies from it, or another paradigm may be activated to allow rapid adaptation of the provisioning service to other types of VN requests or to changes in the physical substrate (*e.g.*, new resources that became available after VN release). The decision to modify or 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 on VMs.

An allocation paradigm $\mathcal{P}$ is defined by a set of goals $(G_1, G_2, ..., G_n)$ that are considered in VN provisioning. Each goal $G_i \in \mathcal{P}$ reflects an InP objective or a characteristic desired for an application running on the VN, having the same meaning of an objective $o^p \in O^p$ supported by the InP or a property $p^v \in P^v$ defined in a VN request, respectively. An individual goal G_i is realized by a set of allocation actions $(A_1, A_2, ..., A_n)$ executed sequentially within a window w_i. Each window w_i has a size attribute $s(w_i) \in \mathbb{N}^+$ corresponding to the number of actions that are executed in each round. An allocator entity $Alloc$ is responsible to trigger each window W. Each allocator is associated with one goal G_i and multiple allocators can run in parallel to speed up the provisioning process.

The provisioning of a VN is thus a function of the resources (*i.e.*, virtual machines and virtual links) that need to be allocated for the requested VN, the number of allocators deployed, and the maximum size of the window of each allocator, which can be dynamically adjusted in each round. The maximum number of rounds $\mathcal{R}$ required to provision a VN is given by:

$$\mathcal{R} = \left\lceil \frac{\sum m^v + \sum l^v}{\sum_{i=1}^{a} s(w_i)} \right\rceil \quad (1)$$

where $(\sum m^v + \sum l^v)$ is the total number of virtual resources (*i.e.*, machines and links) that need to be instantiated per VN, $(\sum_{i=1}^{a} s(w_i))$ is the maximum number of allocation actions allowed per round, and a is the number of allocators deployed.

The size of allocation windows can vary according to the current provisioning status of the requested VNs. If a VN is already deployed and no changes are expected in the short run, the size of the allocation window for that VN is zero. On the other hand, if the VN provisioning has just started or modifications on a previously allocated VN are scheduled, then the size of the window is greater than zero. The size of an allocation window can also be adjusted to prioritize one goal over the others. The higher is the priority of a goal, the larger is the size of its corresponding allocation window. There is a clear tradeoff between the size of the allocation windows and the provisioning time. A large paradigm window requires fewer rounds to allocate a whole VN, but it will not take advantage of a better allocation option that may become available. On the other hand, a small paradigm window is more adaptable to dynamic environments at the price of higher overhead, which can result in larger provisioning times.

Paradigm policies - Paradigm policies are used to guide VN allocation according to a specific goal. Several policies can be applied simultaneously during VN allocation. The general format of a paradigm policy is depicted below (more details can be found in our previous work [8]):

```
objective <name> := <list-of-actions>
                    <window>

action <name> := <conditions>
                 <operation>

window := <size> <order>
```

An action is triggered when a set of associated *conditions* is satisfied. The action is then realized by low-level *operations* (*e.g.*, create a VM) supported by the substrate. The *window* defines the size of allocation windows and the order that the actions of a policy are evaluated.

III. Determining the Efficiency of Allocation Paradigms

A. Applications

In our model, we consider three basic types of applications: Mail, Web 2.0, and E-commerce. Such applications represent typical workloads of virtualized environments [10] [11]. The Mail application is a typical mail server running on a virtual machine. The Web 2.0 application simulates a social network and is structured in two tiers (Web and database) running on separate virtual machines. The E-commerce application is a multi-tiered system composed of four virtual machines (three Web servers and one database server).

The performance of each application is defined by a particular metric. We use the same definitions of the metrics as adopted in measurements using the VMmark benchmark [10]. For each application, VMmark defines a reference value for each metric. Table III summarizes the characteristics of the applications considered in our model.

TABLE III
Application performance metrics

Application	VMs	Metric	Reference value
Mail	1	Actions/minute	330.25 actions/minute
Web 2.0	2	Operations/minute	4641.43 operations/minute
E-commerce-A	4	Transactions/minute	2199.18 Transactions/minute
E-commerce-B	4	Transactions/minute	1518.55 Transactions/minute
E-commerce-C	4	Transactions/minute	1058.05 Transactions/minute

B. VN Scoring Methodology

The VN computation model is based on the concepts of *tiles* and *scores*, typically found in benchmarking systems [10] [11]. A tile is a fixed-size group of virtual machines running multiple applications. In our case, a tile is composed by seven VMs belonging to the Mail (1), Web 2.0 (2), and E-commerce (4) applications, respectively. The score is a numerical value attributed to the VN reflecting the combined performance of all tiles (and applications). Our score calculation is adapted from the VMMark benchmarking system [10] [12], used to measure the performance of applications running on virtualized environments. The score metric $\mathcal{S}$ for a VN is calculated as follows:

$$\mathcal{S} = \sum_{i=1}^{m} \mathcal{T}_i \tag{2}$$

where $\mathcal{T}_i$ is the performance of the tile i and m is the total number of tiles a VN supports. The total score of a VN is thus the sum of the performance of all its tiles. The performance of a individual tile $\mathcal{T}$ is defined by:

$$\mathcal{T} = \left(\prod_{j=1}^{n} \frac{App_j}{Ref_j}\right)^{\frac{1}{n}} \tag{3}$$

where App_j refers to the performance achieved by the jth application in terms of the metrics defined in Table III, Ref_j is a reference value for the application App_j, and n is the total number of applications of the tile. The $\mathcal{T}$ value is thus the geometric mean of the normalized performance of all applications of a tile.

C. VN Performance Computation Model

In order to evaluate the efficiency of an allocation paradigm in terms of application performance, the InP needs to monitor the performance of the applications running on a VN. However, such evaluation may diminish the performance of the applications of ongoing VN requests and result in excessive monitoring traffic in complex environments. Therefore, computing the performance of the applications to be deployed over a VN and evaluating allocation paradigms *in advance* can improve VN provisioning. As a first step towards the definition of a VN performance model, we analyze the relation between the number of tiles and the total score of a VN by analyzing data submitted to the VMmark web site. Figure 2 depicts a scattered plot showing the VN score as a function of the number of tiles.

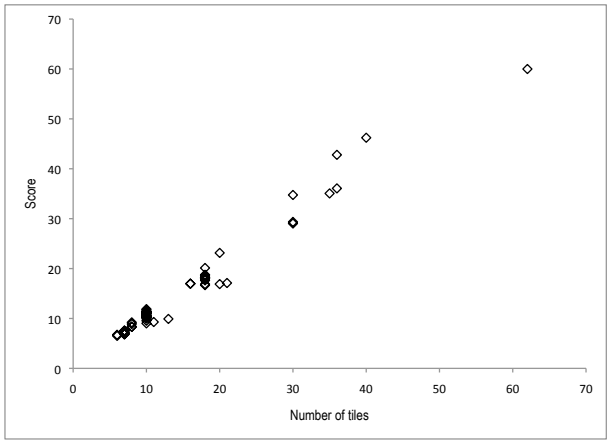


Fig. 2. Score vs. number of tiles

From Figure 2, it is possible to observe that the overall score of a VN grows linearly with the number of tiles. We thus propose a simple linear regression model to compute the performance of a VN given the number of tiles used. Using the R statistical package [13] we found that the computed score $\mathcal{S}'$ of a VN can be defined by:

$$\mathcal{S}' = 0.1573 + \tau \times 1.0206 \tag{4}$$

where τ is the number of tiles. The adjusted R-squared is 0.973.

D. Efficiency of an Allocation Paradigm

The efficiency of an allocation paradigm is influenced by two main factors: the number of rounds required to complete the provisioning of a VN and the score of the allocated VNs. The number of rounds directly impacts the provisioning time of the VN. The score, in turn, reflects the quality of the allocation paradigm. The quality of an allocation paradigm $\mathcal{Q}$ is thus given by:

$$\mathcal{Q} = \frac{S'}{\mathcal{U}} \tag{5}$$

where $\mathcal{U}$ is the performance of the system obtained when applying Equations 2 and 3 to the reference values of Table III.

The ultimate goal of an allocation paradigm is to reduce the number of necessary rounds to allocate a VN, which impacts VN deployment time, and avoid excessive paradigm changes, which is related to the stability of the provisioning system. A paradigm change occurs when the score of the provisioned VNs are below a threshold defined by the InP. Therefore, the main objective of our paradigm-based provisioning framework is to:

$$\text{minimize } \sum \mathcal{R} + \sum (1 - \mathcal{Q}) \tag{6}$$

subject to:

$$0 < \sum_{w \in \mathcal{P}} s(w) \leq \sum m^v + \sum l^v \tag{7}$$

$$\forall m^v \in M^v, \forall l^v \in L^v$$

$$\tau > 0 \tag{8}$$

The objective is to minimize the number of rounds $\mathcal{R}$ required to provision a VN and at the same time improve the quality $\mathcal{Q}$ of the allocated VNs. Constraint 7 assures that the number of rounds is bounded by the sum of the required virtual resources of a VN. Constraint 8 guarantees that at least one tile is allocated.

E. Resource Allocation Algorithm

We propose an algorithm to guide resource allocation in virtualized environments using the concept of paradigms. The algorithm is triggered upon receiving of a VN request. The algorithm checks each goal of the paradigm and the corresponding policy set (lines 10-12). For each policy, the algorithm verifies its set of conditions following the order predefined in the policy (lines 16 and 17). If a condition is met (line 18), the associated allocation action is added to the next allocation window (line 19) until the window reaches its maximum size defined in the policy. The algorithm then executes the allocation windows for each goal of the paradigm (lines 22-25).

In order to avoid that a request waits indefinitely for resources that may not be available in the subsequent rounds we define a parameter called *maxrounds*, which is the maximum number of waiting rounds for an ongoing request. If such number is reached then the request must be "rolled-back", that is, the actions executed previously must be reverted.

After each execution, the algorithm calculates the efficiency of the active paradigm. The number of used tiles is obtained from the VN request (line 29) and the score $\mathcal{S}'$ is calculated (line 30). The quality $\mathcal{Q}$ is also calculated (line 31). If $\mathcal{Q}$ is below a certain threshold defined by the InP, then the active paradigm is updated by adding or removing goals, or switching to another paradigm (lines 32-34).

Algorithm 1 Paradigm-Based Allocation Algorithm

```
1:  W: window of the paradigm P
2:  LR ← M^v ∪ L^v
3:  NLR : size of LR
4:  maxrounds : maximum number of rounds
5:  nrounds : round number
6:  nrounds ← 0
7:  while NLR > 0 and nrounds <= maxrounds do
8:      W ← ∅
9:      for all G ∈ P do
10:         LP ← policy set of the paradigm P
11:         for each p ∈ LP do
12:             C ← conditions of the policy p
13:             A ← actions of the policy p
14:             repeat
15:                 check next condition c ∈ C
16:             until c = true
17:             Add action a ∈ A triggered by c to W
18:         end for
19:         for i ← 1 to s(W) do
20:             Execute action W(i)
21:             NLR ← NLR − 1
22:         end for
23:     end for
24:     nrounds ← nrounds + 1
25:     determine the number of tiles m
26:     calculate the performance S'
27:     calculate the quality Q of the current paradigm
28:     if Q < threshold then
29:         Update current paradigm P
30:     end if
31: end while
```

For example, using a Paradigm Management Subsystem (PMS) [8] the InP initially defines that the allocation paradigm is composed of the Green goal only. Since the paradigm is composed by only one goal, there will be only one policy in LP. However, it is possible to have as many policies as there are goals in the allocation paradigm. The policy related to the Green goal has a condition regarding node allocation that states that if there are nodes to be allocated for a given request, the correspondent action is to select the node with lowest residual capacity (CPU, memory, disk) satisfying the request to host the virtual one (the Load Balancing goal policy on the other hand chooses the node with highest residual capacity instead). If such condition is met, the action (select the node with lowest residual capacity) is included in the allocation window to be executed until there are no action to be included or the window reaches its maximum size s(W).

The actions of an allocation window are executed sequentially until the window is empty, which completes a round. Next, the system calculates the efficiency of the allocation paradigm by comparing the paradigm quality with a threshold

that was manually defined by the InP. If the quality is below this threshold then the system notifies the InP, which, in turn, using the PMS, has the option to change the current allocation paradigm by adding or removing goals, modifying the policies, or switching to another paradigm.

IV. Evaluation

In this section we evaluate our paradigm-based provisioning framework. We aim to measure the efficiency of allocation paradigms in terms of provisioning quality, number of rounds, acceptance ratio, and cost of provisioning.

A. Scenario

We have used the ViNE-Yard simulator developed by Chowdhury *et al.* [14] [15]. We have extended the simulator to include our paradigm efficiency calculation and to allow paradigm changes on the fly. The scenario we consider in the simulations is of a virtualized data center network [16].

We used the VL2 topology [17] as the physical substrate. The network is composed of 24 servers, 22 switches, and 72 links. Each Server is connected to one of the Top-of-Rack switches with a 1-Gbps link whereas switches are connected to each other with 10-Gbps links. Server CPU capacities are uniformly distributed between 1 and 4 cores. For the sake of simplicity we do not consider other resources (*e.g.*, memory, storage) in the evaluation.

VN requests are received according to a Poisson process with an average rate of 4 VNs per 100 time units and an exponentially distributed duration with an average of 1000 time units. The number of VMs of each VN request is randomly generated between 2 and 10. CPU requirements of VMs are uniformly distributed between 1 to 4, and the bandwidth requirements of virtual links are uniformly distributed between 1 to 50 Mbps.

For each experiment, we vary the size of the allocation windows from 1 to 3 and evaluate two allocation paradigms composed of a single InP goal: Green and Load Balancing (LB) as defined in Table II. We also investigate the impact of switching from one paradigm (Green) to another (LB) during the simulation. The maximum number of rounds for each request is defined as the number of requested resources over the size of the allocation window. The number of tiles of each request is calculated by dividing the number of request nodes (M^v) by the size of one tile (Section III-B). Each experiment was repeated 30 times with a confidence level of 95%

B. Metrics

We defined three metrics in our evaluation: (1) *Paradigm quality* reflects the quality of an allocation paradigm in terms of the score computed for the provisioned VNs, (2) *Number of rounds* is the number of allocation rounds necessary to complete the provisioning of a VN as defined in Section II, (3) *Acceptance ratio* is the number of VN requests that are accepted over the total number of requests. Furthermore, we define the *Provisioning cost* which depends on the amount of allocated resources, and is computed as suggested in [14].

C. Results

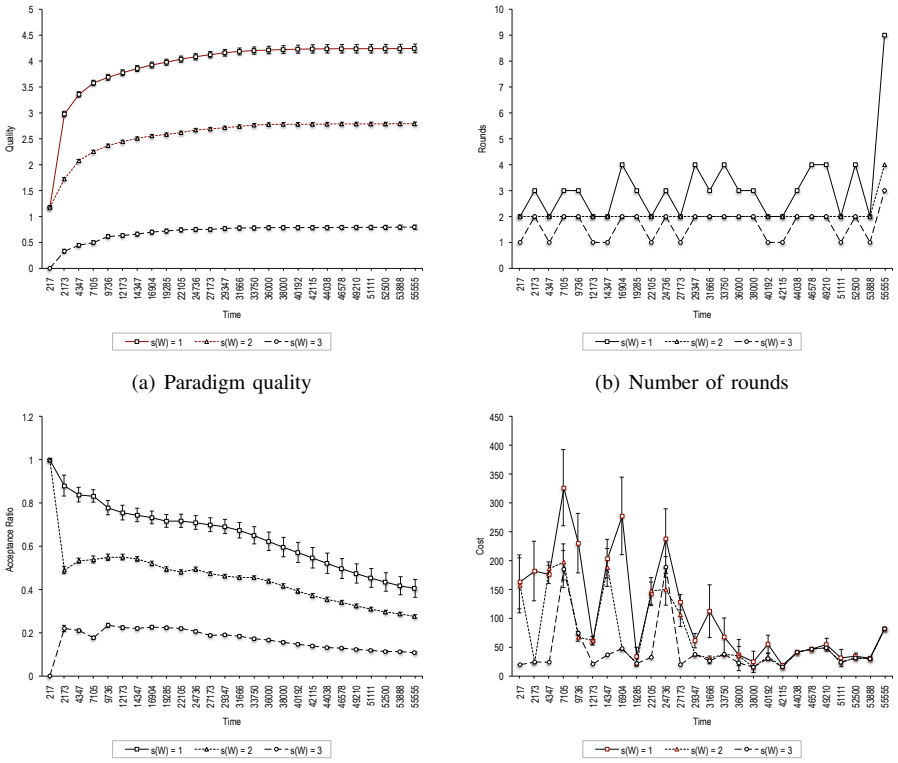
The quality of the Green allocation paradigm is depicted in Figure 3(a). It is easy to see that the paradigm quality is better for small-sized paradigm windows and it gets worse when the paradigm window size is large. This happens because mapping a high number of resources in a single round reduces the *adaptability* of VN provisioning. This is due to the fact that the effect of VN arrivals and departures will only be perceived in the next round, which happens when the actions of the current one are completed. As a consequence, the chances of taking advantage of better mapping alternatives are smaller for paradigms having large windows.

Figure 3(b) shows the average number of rounds required to complete the allocation of a VN with varying window sizes for the Green goal. In order to improve readability, we have decided to select 25 samples (spaced equally from each other) from all the requests. When the size of the paradigm window s(W) is equal to 1, the number of rounds to complete the allocation is the exact number of virtual nodes of the VN request. For s(W) = 2, the number of rounds is static because most of the virtual nodes were allocated in only two rounds. As expected, when s(W) = 3 most VN requests are completed in two rounds and some (the smallest ones) in just one round. The number of rounds influences the time needed to allocate a VN. VNs allocated in few rounds (*i.e.*, large paradigm windows) are rapidly available to SPs, while VN requests that need many rounds to be deployed have larger provisioning times.

The acceptance ratio for the Green goal is illustrated in Figure 3(c). For s(W) = 1, acceptance ratio remains above 40% for the whole simulation time. When s(W) = 2, the acceptance ratio rapidly decreases after 2000 time units and stays between 55% and 27% for the rest of the simulation. The situation is even worse for s(W) = 3, when acceptance ratio is no higher than 25%, indicating that similar to paradigm quality, there is a clear association between the size of the window and the acceptance ratio of VN requests. VN requests allocated using paradigms with small windows are more adaptable and less unlikely to be rejected. Furthermore, larger windows result in a higher number of rejected requests because the Green goal attempts to select the physical node having the highest residual capacity. If the paradigm window is large, the node will run out of capacity fast, because it will be selected more times in a single round, which explains the low acceptance ratio for s(W) = 2 and s(W) = 3.

Figure 3(d) presents the average cost of provisioning a VN. During most of the simulation (time $<$36000), the cost of allocated VNs is higher when s(W) = 1 compared to s(W) = 2, which, in turn, results is higher costs compared to s(W) = 3. This can be explained by the fact that the acceptance ratio is higher for s(W) = 1, which means that more resources are allocated and the overall cost for the InP increases. However, after 36000 time units, the cost of allocated VNs converge regardless of the size of the paradigm window.

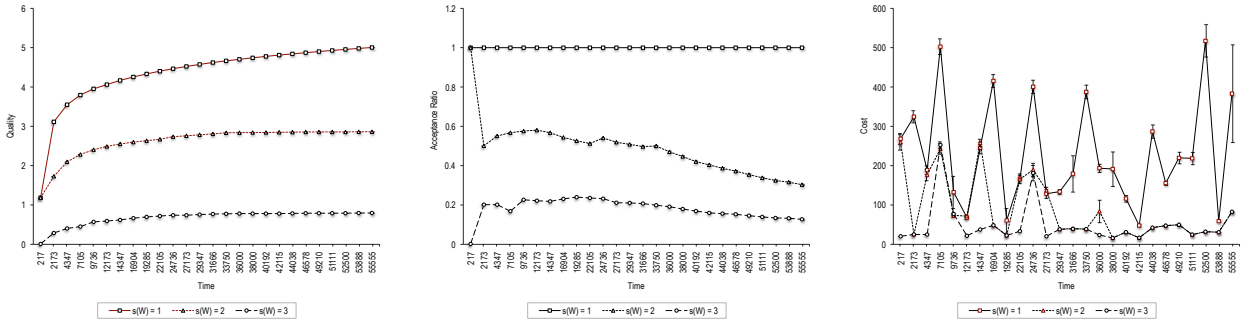
Figure 4(a) shows the quality of the allocation paradigm using the Load Balancing goal. The paradigm quality is slightly



(a) Paradigm quality (b) Number of rounds

(c) Acceptance ratio (d) Provisioning cost

Fig. 3. Simulation results (Green objective): (a) Paradigm quality. (b) Number of rounds. (c) Acceptance ratio. (d) Provisioning cost.



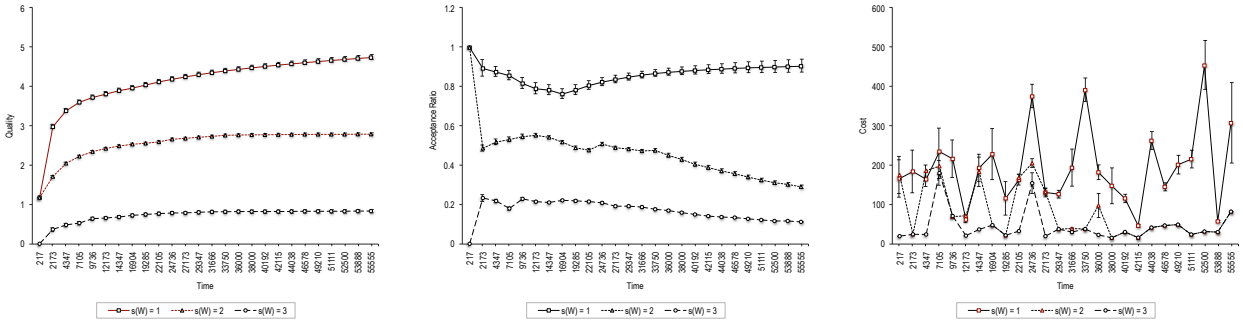
(a) Paradigm quality (b) Acceptance ratio (c) Provisioning cost

Fig. 4. Simulation results (Load balancing objective): (a) Paradigm quality. (b) Number of rounds. (c) Acceptance ratio. (d) Provisioning cost.

superior when LB is deployed compared to the paradigm using the Green goal. This happens because LB always searches for the physical node with the highest residual capacity. Since the initial configuration (*i.e.*, capacity) of physical nodes is the same, LB selects a different node in each turn, thus increasing the number of allocated tiles and, consequently, the paradigm quality. Regarding paradigm windows sizes, similar to the Green goal, s(W) = 1 presents higher performance when compared to s(W) = 2 and s(W) = 3 because small paradigm windows allows rapid adaptation of VN provisioning to changes in the substrate.

The number of rounds used by the LB goal to provision VNs are the same of the Green goal, which can be easily explained by the fact that the number of rounds depends only on s(W) and not on the goal employed in the paradigm.

Acceptance ratio for the LB goal is shown in Figure 4(b). For s(W) = 1, all VNs are successfully allocated. This can be explained by the fact that LB tends to select a different node with high residual capacity in each round, increasing the chances of accepting the VN request. For s(W) = 2 and 3, the acceptance ratio is between 30% and 55%, close to the results of the Green goal. This indicates that the performance of different allocation paradigms tends to be similar as the window size increases, regardless of the goal employed.



(a) Paradigm quality (b) Acceptance ratio (c) Provisioning cost

Fig. 5. Simulation results (Paradigm Switching): (a) Paradigm quality. (b) Number of rounds. (c) Acceptance ratio. (d) Provisioning cost.

The cost of VN provisioning for the LB goal is presented in Figure 4(c). Unlike Green goal, the cost of VNs provisioned with the LB goal when s(W) = 1 is higher than s(W) = 2 and s(W) = 3 for most of the simulation. This reflects the fact that more VNs are allocated when s(W) = 1. Furthermore, the average cost of allocated VNs for s(W) = 2 and s(W) = 3 converge after 36000 time units and is comparable to the cost obtained by the Green goal.

Figure 5(a) shows the paradigm quality when the goal of a current allocation paradigm is switched during the simulation. In our experiments, the used thresholds to switch between paradigms were defined manually (half of the simulation time). However, our approach is still valid even if we consider more sophisticated techniques to estimate these thresholds. The quality of the allocation in the first half of the simulation remains between 1.2 and 4.2, which reflects the performance of the Green goal (see Figure 3(a)). After that, the quality increases and remains between 4.5 and 4.7, which are close to the values obtained by the LB goal. This shows that changing between goals can improve paradigm quality. Again, the number of rounds for paradigm switching is the same of all previous scenarios since it does not depend on the goals employed in the paradigm.

The acceptance ratio for paradigm switching is shown in Figure 5(b). The impact of paradigm switching is more evident for s(W) = 1. After paradigm switching, the acceptance ratio starts to increase because LB offers very high acceptance ratio (Figure 4(b)).

The cost of provisioning when paradigms are switched is depicted in Figure 5(c). Again, for s(W) = 1 the cost does not reduce as in Green goal (see Figure 3(d)) after paradigm switching. Instead, the average provisioning cost stays higher than s(W) = 2 and s(W) = 3, confirming the impact of changing between goals during VN provisioning on the InP revenue.

D. Summary

In general, the LB paradigm provides better performance in terms of paradigm quality when compared to the Green paradigm. This happens because LB tries to spread VN requests over a high number of physical resources, thus increasing the acceptance ratio, which reflects in the quality of the provisioned VNs. However, this comes at the cost of more expensive VNs because more resources are allocated by LB compared to Green. When the window size is 1, the number of rounds required to provision VN is higher, which means that VNs will take longer time to be fully allocated.

When s(W) is set to 2, the quality of the provisioned VNs decreases. This in turn leads to the drop of the acceptance ratio. The lower quality of the allocated VNs when the window size is high is due to simultaneous mapping of multiple resources, which increases the number of rejected requests.

The benefit of switching between paradigms is more clear in Figure 5(b) when the allocation window size is 1. After time instant 16900 the acceptance ratio starts to increase. This paradigm switching feature is useful when the InP needs to adapt its provisioning strategy to support different application profiles that arrive dynamically in the substrate.

V. Conclusion

In this paper we have proposed an adaptive provisioning framework that considers specific application characteristics and performance objectives. Our proposal is based on the concept of allocation paradigms, which are groups of application-related policies that guide resource allocation. In addition, we also proposed a virtual network performance computation model to evaluate the efficiency of paradigm-based allocations.

We also illustrated the tradeoff between the size of the allocation window and the provisioning quality. We found that small windows produce high quality VNs at the cost of higher provisioning times and increased provisioning costs. Larger windows allow rapid VN deployment at the expense of quality.

Future work includes investigating other application performance models, such as queuing models and autoregressive models. We also plan to implement a VN provisioning scheme based on the concept of allocation paradigm.

Acknowledgment

This work was supported in part by the National Council for Scientific and Technological Development (CNPq), Brazil and in part by the Natural Science and Engineering Council of Canada (NSERC) under the Smart Applications on Virtual Infrastructure (SAVI) Research Network.

REFERENCES

[1] 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.

[2] 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.

[3] 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, 2012.

[4] M. F. Zhani, Q. Zhang, G. Simon, and R. Boutaba, "VDC Planner: Dynamic Migration-Aware Virtual Data Center Embedding for Clouds," in *IFIP/IEEE Integrated Network Management Symposium (IM)*, Ghent, Belgium, 2013.

[5] M. Rabbani, R. Esteves, M. Podlesny, G. Simon, L. Z. Granville, and R. Boutaba, "On Tackling Virtual Data Center Embedding Problem," in *IFIP/IEEE Integrated Network Management Symposium (IM)*, Ghent, Belgium, 2013.

[6] Q. Zhu and G. Agrawal, "Resource Provisioning with Budget Constraints for Adaptive Applications in Cloud Environments," in *International Symposium on High Performance Distributed Computing (HPDC)*, New York, NY, USA, 2010.

[7] M. Frincu and C. Craciun, "Multi-objective Meta-heuristics for Scheduling Applications with High Availability Requirements and Cost Constraints in Multi-Cloud Environments," in *IEEE International Conference on Utility and Cloud Computing (UCC)*, Washington, DC, USA, 2011.

[8] R. P. Esteves, L. Z. Granville, H. Bannazadeh, and R. Boutaba, "Paradigm-Based Adaptive Provisioning in Virtualized Data Centers," in *IFIP/IEEE Integrated Network Management Symposium (IM)*, Ghent, Belgium, May 2013.

[9] 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 *ACM CoNEXT*, New York, USA, 2010.

[10] VMmark. http://www.vmware.com/a/vmmark/.

[11] Standard Performance Evaluation Corporation (SPECvirt_sc2010). http://www.spec.org/virt_sc2010/.

[12] VMware VMmark Benchmarking Guide. http://www.vmware.com/go/download-vmmark/.

[13] The R Project for Statistical Computing. http://www.r-project.org.

[14] 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.

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

[16] M. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, "Data Center Network Virtualization: A Survey," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 2, pp. 909–928, 2013.

[17] 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 *ACM SIGCOMM*, August 2009, pp. 51–62.