NORTHWESTERN
UNIVERSITY

Electrical Engineering and Computer Science Department

Technical Report
NWU-EECS-06-06
July 30, 2006

Hardness of Approximation and Greedy Algorithms for the Adaptation Problem in Virtual Environments

Ananth I. Sundararaj **Manan Sanghi** **John R. Lange** **Peter A. Dinda**

Abstract

Over the past decade, wide-area distributed computing has emerged as a powerful computing paradigm. However, developing applications to execute over the wide-area has remained a challenge, primarily due to issues involved in providing automatic, dynamic and run-time adaptation. A virtual execution environment consisting of virtual machines (VMs) interconnected with virtual networks provides opportunities to dynamically optimize, at run-time, the performance of existing, unmodified distributed applications without any user or programmer intervention. Along with resource monitoring, inference and application-independent adaptation mechanisms, efficient adaptation algorithms are key to the success of such an effort. In this paper we formalize the adaptation problem in virtual execution environments. We show that this adaptation problem is NP-hard. Further, we characterize the adaptation problem's hardness of approximation and show that it is NP-hard to approximate within a factor of $m^{1/2-\delta}$ for any $\delta > 0$, where m is the number of edges in the virtual overlay graph. We then present greedy adaptation algorithms followed by an evaluation that shows that the greedy strategy works well in practice.

Effort sponsored by the National Science Foundation under Grants ANI-0093221, ANI-0301108, and EIA-0224449. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation (NSF).

Keywords: Adaptive systems, resource virtualization, bounds

Hardness of Approximation and Greedy Algorithms for the Adaptation Problem in Virtual Environments

Ananth I. Sundararaj, Manan Sanghi, John R. Lange and Peter A. Dinda
{ais,manan,jarusl,pdinda}@cs.northwestern.edu
Department of Electrical Engineering and Computer Science
Northwestern University, Evanston IL, USA

Abstract

Over the past decade, wide-area distributed computing has emerged as a powerful computing paradigm. However, developing applications to execute over the wide-area has remained a challenge, primarily due to issues involved in providing automatic, dynamic and run-time adaptation. A virtual execution environment consisting of virtual machines (VMs) interconnected with virtual networks provides opportunities to dynamically optimize, at run-time, the performance of existing, unmodified distributed applications without any user or programmer intervention. Along with resource monitoring, inference and application-independent adaptation mechanisms, efficient adaptation algorithms are key to the success of such an effort. In this paper we formalize the adaptation problem in virtual execution environments. We show that this adaptation problem is NP-hard. Further, we characterize the adaptation problem's hardness of approximation and show that it is NP-hard to approximate within a factor of $m^{1/2-\delta}$ for any $\delta > 0$, where m is the number of edges in the virtual overlay graph. We then present greedy adaptation algorithms followed by an evaluation that shows that the greedy strategy works well in practice.

1. Introduction

Over the past decade, wide-area distributed computing has emerged as a powerful computing paradigm [1–3]. However, developing applications for such environments has remained a challenge, primarily due to the issues involved in designing automatic, dynamic and run-time adaptation schemes. Any application running in a distributed environment needs to adapt to available computational and network resources to optimize its performance. Despite many efforts [4, 5], until recently, all adaptation in distributed applications had remained application specific and dependent on direct involvement of the developer or user. Such custom adaptation involving the user or developer is extremely difficult due to the dynamic nature of application demands and resource availability.

Since then it has been argued that OS-level virtual machines (VMs) [6] provide a very flexible and powerful abstraction to perform wide-area distributed computing [7–9]. In particular, we have previously shown that virtual execution environments consisting of virtual machines tied together via virtual networks provide an opportunity to dynamically optimize, at run-time, the performance of existing, *unmodified* distributed applications running on existing, unmodified operating systems without any user or programmer intervention [10, 11]. The relative virtualization overhead has been previously shown to be less than 5% [7, 12], deeming this abstraction feasible.

Such virtual execution environments [11] provide an ideal platform for inferring application resource demands and measuring available computational and network resources. Further, they also make available application independent adaptation mechanisms such as VM migration, overlay network topology and routing changes and resource reservations. However, the key to success is an efficient algorithm to drive these adaptation mechanisms as guided by the measured and inferred data. To gain a better understanding of the adaptation problem and to devise efficient algorithms it is important to formalize and characterize the adaptation problem.

In this paper we provide a rigorous formalization of

the adaptation problem that occurs in virtual execution environments. We characterize its computational complexity and hardness of approximation. We also present a few greedy algorithms that perform well in practice. The contributions of this work include:

(i) Formalizing a real adaptation problem that occurs in virtual execution environments. To the best of our knowledge there currently exists no theoretical analysis for real problems that involve both mapping and routing aspects. The formalization is abstract and generic enough to allow other adaptation problems in many different contexts, such as hardware chip design [13], to possibly map onto it.

(ii) Proving the problem to be NP-hard thereby necessitating the search for approximate solutions.

(iii) Characterizing the problem's hardness of approximation by proving that it is NP-hard to approximate within a factor of $m^{1/2-\delta}$ for any $\delta > 0$, where m is the number of edges in the virtual overlay graph. This shows that the adaptation problem is hard to approximate as well.

(iv) Devising four different variations of greedy algorithms as solutions that work well in practice.

(v) Comparing and contrasting the algorithms with a view to gaining a better understanding of the problem.

2. Related Work

Over the last decade there has been a great deal of interest in wide area distributed computing, primarily due to the substantial increase in commodity computer and network performance. GLobal Object Based Environment, GLOBE, is a wide area distributed system that provides a convenient programming abstraction and full transparency [1]. The computational grid refers to the abstraction of a single unified computing resource that harnesses computational resources geographically distributed under different administrative domains and connected via wide area networks [2]. Legion is an object-based meta-system [3] that provides the software infrastructure for a system of heterogeneous, geographically distributed high-performance computers to interact seamlessly.

An application running in any distributed computing environment must adapt to the (dynamically changing) available computational and networking resources to achieve stable high performance. Over the years there have been numerous attempts at adaptation in different settings such as load balancing in networks of shared processors [14], solutions to workflow problems, component placement problems and support for heavyweight applications in computational grids [5], distributed mobile applications [15], automated runtime tuning systems [4], adaptation, load balancing and fault tolerance in message passing and parallel processing systems spread over heterogeneous resources [16, 17], and extensions to commercial standards such as CORBA [18]. Despite these efforts adaptation and control mechanisms are not common in today's distributed computing environments as most of the approaches are very application-specific and require considerable user or developer effort.

Recently, interest in using OS-level virtual machines as the abstraction for distributed computing has been growing [7, 19]. These build upon operating-system level virtual machines, of which there are essentially two kinds, fully virtualized virtual machine monitors [6] and paravirtualized technology [20]. The former provides a full virtualization, of at least a subset, of the underlying hardware while the latter provides a software interface to virtual machines that is similar but not identical to that of the underlying hardware. Virtuoso [21], PlanetLab [22], SODA [8] and Terra [9] are all virtual execution environments for distributed computing. Though this work has been done in the context of Virtuoso, it should be noted that the adaptation problem generalizes to other scenarios as well. Virtuoso is described in Section 3.

In a previous work [23] we laid the groundwork for the characterization of the adaptation problem in virtual execution environments. This work directly builds upon it by further generalizing the adaptation to include all the pieces involved in such a system. We show that not only is it hard to find an efficient solution but that it is hard to approximate it within a factor of $m^{1/2-\delta}$ for any $\delta > 0$, where m is the number of edges in the virtual overlay graph. In response we have come up with efficient greedy algorithms that work well in practice.

We formulate the generic adaptation problem in virtual environments with the objective of maximizing the sum of the residual bottleneck bandwidths over all the mapped paths. Numerous cases of related work exist in optimizing network flows. The two closest problems to our formulation are the Edge Disjoint Path Problem (EDPP) and the Unsplittable-flow problem (UFP). EDPP appears in Karp's original list of NP-complete problems [24] and has been extensively studied. A comprehensive background on these problems is available elsewhere [25].

One of the motivations for formulating UFP is to address the problem of allocating bandwidth for traffic with different bandwidth requirements in heterogeneous networks [25]. The UFP has been shown to be MAX SNP-hard [26]. Hence numerous prior works

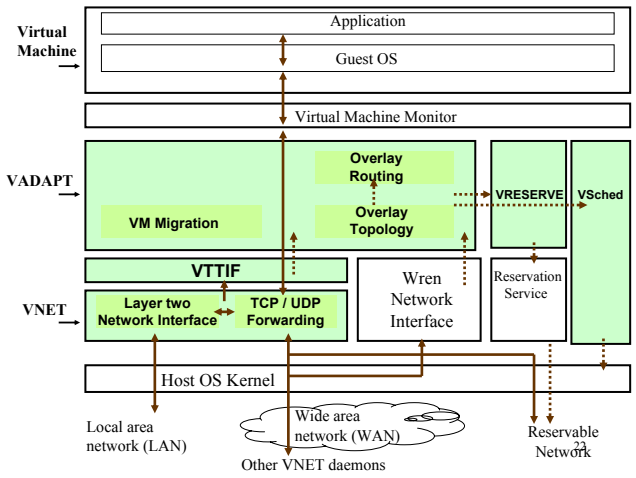


Fig. 1. Virtuoso system architecture.

have explored approximation algorithms for UFP [25–28]. An LP-based algorithm provided a $O(\sqrt{m})$ approximate algorithm, where m is the number of edges in the graph [27]. This was followed by a simpler combinatorial algorithm with the same approximation guarantee [29].

The main difference between our problem and UFP is that the profits in UFP associated with each source-sink pair are predetermined and static, while in our problem the profits depend on the particular solution (path between each source-sink pair) to the instance of the problem at hand. Further, our problem has an additional mapping component that is absent in network flow problems typically investigated. Hence our adaptation problem also has a strong connection to parallel task graph mapping problems [30]. To the best of our knowledge no prior theoretical works exists that includes both the mapping and network flow components.

3. Dynamic adaptation in virtual execution environments

Virtuoso is our middleware system for virtual machine based wide-area distributed computing [11, 21]. For a user, it very closely emulates the existing process of buying, configuring, and using a computer or a collection of computers from a web site [31]. Instead of a physical computer, the user now receives a reference to the virtual machine which she can then use to start, stop, reset, and clone the machine. Since such a virtual machine could be hosted on any foreign network, the nature of the network presence that the virtual machine gets depends solely on the policies of the remote site. Figure 1 illustrates our system architecture. We next describe the different pieces of Virtuoso that are salient to this work.

VNET [21] is a simple and efficient Ethernet layer virtual network tool that interconnects all the VMs of a user and creates the illusion that they are located on the user's local area network (LAN) by bridging the foreign LAN to a Proxy on the user's network. VNET makes available application independent adaptation mechanisms that can be used to automatically and dynamically optimize at run-time the performance of applications running inside of a user's VMs [11].

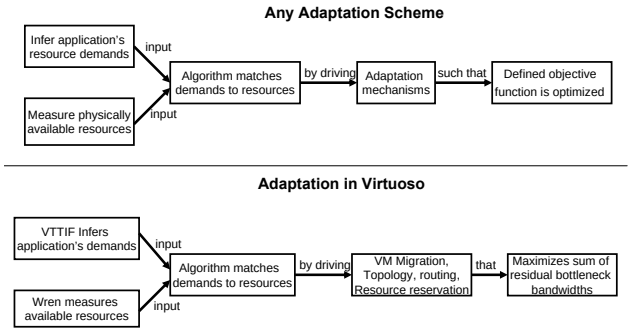
It has been previously shown that VNET is ideally placed to monitor the resource demands of the VMs. The VTTIF (Virtual Topology and Traffic Inference Framework) component of Virtuoso, integrated with VNET, achieves this [32]. Wren is a passive network measurement tool developed at the College of William and Mary [33], that we have integrated with Virtuoso. It can use the naturally occurring traffic of existing, unmodified applications running inside of the VMs to measure the characteristics of the underlying physical network [34]. VRESERVE [35] and VSched [36] are our network and CPU reservation systems respectively.

Such virtual execution environments provide an ideal platform to build an automatic, dynamic and run-time adaptation scheme that works for *unmodified* applications running on *unmodified* operating systems. In simple terms a successful adaptation scheme will involve an efficient algorithm that matches the application's inferred resource (network and computation) demands to the measured available resources using adaptation mechanisms at hand such that some defined metric is optimized. However, what is missing is a rigorous formalization of the adaptation problem, characterization of its hardness and the hardness of its approximability, solutions that work well in practice and the characterization of the same. This work, in part, serves this purpose. Figure 2 illustrates how these pieces fit in together.

Inferring application resource demands: This involves measuring the computational and network demands of applications running inside the virtual machines. In previous work it has been shown how VTTIF successfully accomplishes this in the context of Virtuoso [32].

Measuring available resources: This involves monitoring the underlying network and inferring its topology, bandwidth and latency characteristics, and also measuring the availability of computational resources. Again, it has been previously shown that this can be achieved with very little overhead in the context of virtual environments [34].

Adaptation mechanisms at hand: Virtual execution environments make available the following adaptation



Virtuoso: A virtual execution environment
VNET: Virtual NETtworking component that makes available adaptation mechanisms
VTTIF: Virtual Topology and Traffic Inference Framework that infers application demands
Wren: Watching Resources from edge of the network measurement tool

Fig. 2. Adaptation scheme in virtual execution environments using Virtuoso as an example.

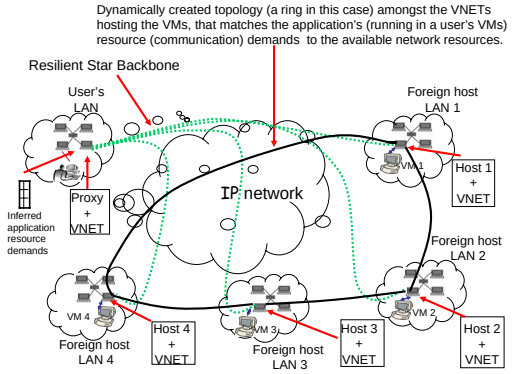


Fig. 3. As the application progresses Virtuoso adapts its overlay topology to match that of the application communication as inferred by VTTIF leading to a significant improvement in application performance, without any participation from the user.

mechanisms: VM migration, virtual network topology and routing changes, CPU and network resource reservation. These have been previously described in the context of Virtuoso [11, 35, 36].

Measure of performance: For the purposes of this work, we are attempting to maximize the application's throughput. We claim that optimizing the defined metric will achieve our goal. At this point in time it is not known if a single optimization scheme will work effectively for a range of distributed applications.

Adaptation algorithm: Finally we need an efficient adaptation algorithm that will tie all these individual pieces together.

Figure 3 illustrates a simplified version of a typical adaptation scenario in Virtuoso wherein a heuristic drives application independent adaptation mechanisms (in this case overlay topology and routing changes), while leveraging inferred application resource demands and measured resource information.

4. Adaptation problem formulation

VNET monitors the underlying network and provides a directed VNET topology graph, $G = (H, E)$, where H are VNET nodes (hosts running VNET daemons and capable of supporting one or more VMs) and E are the possible VNET links. Note that this may not be a complete graph as many links may not be possible due to particular network management and security policies at different network sites. Wren [33] (integrated with VNET [34]) provides estimates for the available bandwidth and latencies over each link in the VNET topology graph. These estimates are described by a bandwidth capacity function, $\text{bw} : E \rightarrow \mathbb{R}$, and a latency function, $\text{lat} : E \rightarrow \mathbb{R}$.

In addition, VNET is also in a position to collect information regarding the space capacity (in bytes) and compute capacity made available by each host, described by a host compute capacity function, $\text{compute} : H \rightarrow \mathbb{R}$ and a host space capacity function, $\text{size} : H \rightarrow \mathbb{R}$. The set of virtual machines participating in the application is denoted by the set VM. The size and compute capacity demands made by every VM can also be estimated and denoted by a VM compute demand function, $\text{vm\_compute} : VM \rightarrow \mathbb{R}$ and a VM space demand function, $\text{vm\_size} : VM \rightarrow \mathbb{R}$, respectively. We are also given an initial mapping of virtual machines to hosts, M, which is a set of 3-tuples, $M_i = (\text{vm}_i, h_i, y_i)$, $i = 1, 2 \ldots n$, where $\text{vm}_i \in \text{VM}$ is the virtual machine in question, $h_i \in H$ is the host that it is currently mapped onto and $y_i \in \{0, 1\}$ specifies whether the current mapping of VM to host can be changed or not. A value of 0 implies that the current mapping can be changed and a value of 1 means that the current mapping should be maintained.

The bandwidth and compute rate estimates do not implicitly imply reservation, they are random variables that follow a normal distribution with a mean of the estimated value. As mentioned previously Virtuoso provides for network and CPU reservations, in which case the estimates are exactly the resources we get as we can reserve the same. Hence for each edge in E, we define a function $\text{nw\_reserve} : E \rightarrow \{0, 1\}$. If the value associated with the edge is 0 then we cannot reserve the link and the actual bandwidth has a normal distribution with a mean of $\text{bw}(E)$ and a variance $\sigma^2_{bw(E)}$, else the link is reservable and the actual bandwidth is $\text{bw}(E)$. Similarly for each host we define a function $\text{cpu\_reserve} : H \rightarrow \{0, 1\}$, where a value of 0 means that the compute capacity made available by the host is not reservable and the actual value has a normal distribution with a mean of $\text{compute}(H)$ and a variance $\sigma^2_{\text{compute}(H)}$.

VTTIF infers the application communication topology in order to generate the traffic requirements of

the application, A, which is a set of 4-tuples, $A_i = (s_i, d_i, b_i, l_i)$, $i = 1, 2 \ldots m$, where s_i is the source VM, d_i is the destination VM, b_i is the bandwidth demand between the source destination pair and l_i is the latency demand between the source destination pair.

It should be noted that there is always a cost involved with all the measurements and adaptation mechanisms. Because the overheads of VNET, VTTIF and Wren have been shown to be negligible [34] we do not include them in our formalization. However, the cost of migrating a virtual machine is dependent on the size of the virtual machine, the network characteristics between the corresponding hosts and the specific migration scheme used. These estimates are described by a migration function, migrate: VM x H x H $\rightarrow \mathbb{R}^+$, that provides an estimate in terms of the time required to migrate a virtual machine from one host to another. There is more than one way to take into account the cost of migration, one being to keep the costs of migration for each of the VMs below a certain threshold. Online migration of virtual machines is receiving a lot of interest in the research community [37–39]. As the migration times are being continually driven down the relevance of our work will continue to increase.

The goal then is to find an adaptation algorithm that uses the measured and inferred data to drive the adaptation mechanisms at hand in order to improve application throughput. In other words we wish to find

(i) a mapping from VMs to hosts, vmap : $VM \rightarrow H$, meeting the size and compute capacity demands of the VMs within the host constraints and leveraging CPU reservations where available. Further, the new mapping should also reflect the mapping constraints provided.

(ii) a routing, $R : A \rightarrow P$, where P is the set of all paths in the graph $G = (H, E)$, i.e. for every 4-tuple, $A_i = (s_i, d_i, b_i, l_i)$, allocate a path, $p\big(\text{vmap}(s_i), \text{vmap}(d_i)\big)$, over the overlay graph, G, meeting the application demands while satisfying the bandwidth and latency constraints of the network and leveraging network reservations where available.

Once all the mappings and paths have been decided, each VNET edge will have a residual capacity, rc_e, which is the bandwidth remaining unused on that edge, in that direction

$$\text{rc}_e = \text{bw}_e - \sum_{e \in R(A_i)} b_i$$

For each mapped path, $R(A_i)$, we can also define its bottleneck residual capacity

$$\text{brc}\big(R(A_i)\big) = \min_{e \in R(A_i)} \big\{\text{rc}_e\big\}$$

and its total latency

$$\text{tl}\big(R(A_i)\big) = \sum_{e \in R(A_i)} \big(lat_e\big)$$

It should be noted that the residual capacity can be spoken of at two levels, at the level of VNET edges and at the level of paths between communicating VMs. The various objective functions that could be defined would fall into one of two classes, an edge-level or a path-level objective function.

(i) Edge-level: a composite function, f, that is a function of, g, a function of the migration costs of all the VMs and h, a function of the total latency over all the edges for each routing and k, a function of the residual bottleneck bandwidths over all the edges in the VNET graph.

(ii) Path-level: a composite function, f, that is a function of, g, a function of the migration costs of all the VMs and h, a function of the total latency over all the edges for each routing and k, a function of the residual bottleneck bandwidths over all the paths in the routing.

Problem 1 (Generic Adaptation Problem In Virtual Execution Environments (GAPVEE))
INPUT:

- A directed graph $G = (H, E)$
- A function bw : $E \rightarrow \mathbb{R}$
- A function lat : $E \rightarrow \mathbb{R}$
- A function compute : $H \rightarrow \mathbb{R}$
- A function size : $H \rightarrow \mathbb{R}$
- A set, VM $= (\text{vm}_1, \text{vm}_2 \ldots \text{vm}_n)$, $n \in \mathbb{N}$
- A function vm_compute : VM $\rightarrow \mathbb{R}$
- A function vm_size : VM $\rightarrow \mathbb{R}$
- A function migrate : $(VM, H, H) \rightarrow \mathbb{R}$
- A function nw_reserve : $E \rightarrow \{0, 1\}$
- A function cpu_reserve : $H \rightarrow \{0, 1\}$
- A set of ordered 4-tuples $A = \{(s_i, d_i, b_i, l_i) \mid s_i, d_i \in \text{VM};\ b_i, l_i \in \mathbb{R};\ i = 1, \ldots, m\}$
- A set of ordered 3-tuples $M = \{(\text{vm}_i, h_i, y_i) \mid \text{vm}_i \in \text{VM};\ h_i \in H;\ y_i \in \{0, 1\};\ i = 1, \ldots, n\}$

OUTPUT: vmap : $VM \rightarrow H$ and $R : A \rightarrow P$ such that

- $\sum_{\text{vmap(vm)}=h} \big(\text{vm\_compute(vm)}\big) \leq \text{compute(h)},\ \forall\ h \in H$
- $\sum_{\text{vmap(vm)}=h} \big(\text{vm\_size(vm)}\big) \leq \text{size}(h),\ \forall\ h \in H$
- $h_i = \text{vmap}(\text{vm}_i) \quad \forall\, M_i = (\text{vm}_i, h_i) \in M$ if $y_i = 1$
- $\text{rc}_e \geq 0,\ \forall e \in E$
- $\big(\sum_{e \in R(A_i)} \text{lat}_e\big) \leq l_i,\ \forall e \in E$
- For some functions f, g, h and k the function $f(g(migrate), h(\text{lat}), k(\text{rc}_e))$ is optimized

It should be noted that for this most generic incarnation we have not specified any particular objective function. The intent of providing this formulation is to provide an abstract description of all the components of the adaptation problem. We next take a significant piece of this generic problem and analyze and characterize it in great detail.

Mapping and routing are the two main components of our adaptation problem. With a view to better understand these two components we define a simpler version wherein we drop the size, compute and latency constraints. We also neglect the cost of migration, which is reasonable as recently migration costs as low as a few seconds have been reported [38]. It should be noted that if the migration is conducted online then the downtime is virtually zero [39]. We also assume that all the links are reservable and that the compute capacity made available is reserved as well.

The specific objective function we choose belongs to the second category mentioned above wherein we consider residual bandwidths of the various paths in the routing. The objective is to maximize the sum of residual bottleneck bandwidths over each mapped path. The intuition behind this objective function is to leave the most room for the application to increase its throughput.

Problem 2 (Mapping and Routing Problem In Virtual Execution Environments (MARPVEE))
INPUT:

- A directed graph $G = (H, E)$
- A function bw : $E \rightarrow \mathbb{R}$
- A set, $\text{VM} = (\text{vm}_1, \text{vm}_2 \ldots \text{vm}_n)$, $n \in \mathbb{N}$
- A set of ordered 3-tuples $A = \{(s_i, d_i, b_i) \mid s_i, d_i \in \text{VM};\ b_i;\ i = 1, \ldots, m\}$
- A set of ordered 3-tuples $M = \{(\text{vm}_i, h_i, y_i) \mid \text{vm}_i \in \text{VM};\ h_i \in H;\ y_i \in \{0,1\};\ i = 1, \ldots, n\}$

OUTPUT: $\text{vmap} : VM \rightarrow H$ and $R : A \rightarrow P$ such that

- $h_i = \text{vmap}(\text{vm}_i) \quad \forall\, M_i = (\text{vm}_i, h_i) \in M$ if $y_i = 1$
- $\text{rc}_e \geq 0,\ \forall e \in E$
- $\sum_{i=1}^{m} \left(\min_{e \in R(A_i)} \{\text{rc}_e\}\right)$, where $\text{rc}_e = (\text{bw}_e - \sum_{e \in R(A_i)} b_i)$, is maximized

From now on when we refer to the adaptation problem we will be referring to MARPVEE.

5. Computational complexity of the adaptation problem

We first formulate the decision version of the adaptation problem.

Problem 3 (Mapping and Routing Problem In Virtual Execution Environments (MARPVEED))
INPUT:

- A directed graph $G = (H, E)$
- A function bw : $E \rightarrow \mathbb{R}$
- A set, $\text{VM} = (\text{vm}_1, \text{vm}_2 \ldots \text{vm}_n)$, $n \in \mathbb{N}$
- A set of ordered 3-tuples $A = \{(s_i, d_i, b_i) \mid s_i, d_i \in \text{VM};\ b_i;\ i = 1, \ldots, m\}$
- A set of ordered pairs $M = \{(\text{vm}_i, h_i) \mid \text{vm}_i \in \text{VM}, h_i \in H;\ i = 1, 2 \ldots r, r \leq n\}$
- $\alpha \in \mathbb{R}$

OUTPUT:

- YES, if there exists a mapping $\text{vmap} : VM \rightarrow H$ and a routing $R : A \rightarrow P$ such that
  - $h_i = \text{vmap}(\text{vm}_i),\ \forall M_i = (\text{vm}_i, h_i) \in M$
  - $\text{rc}_e \geq 0,\ \forall e \in E$
  - $\sum_{i=1}^{m} \left(brc(R(A_i))\right) \geq \alpha$
- NO, otherwise

To establish the hardness of the problem, we consider a further special case of the problem wherein all the VM to host mappings are constrained by the set of 3-tuples M, leaving us only with the routing problem.

Since the mappings are pre-defined, we can formulate the problem in terms of only the hosts and exclude all VMs. Also, as the latency demands have been dropped, the application 4-tuple reduces to 3-tuple, $A_i = (s_i, d_i, b_i),\ s_i, d_i \in H,\ b_i \in \mathbb{R},\ i = 1, 2 \ldots m$. Notice that now $s_i, d_i \in H$ as VM to host mappings are fixed and VMs are synonymous with the hosts that they are mapped to.

This further constrained version of the adaptation problem with only the routing component is defined as follows.

Problem 4 (Routing Problem In Virtual Execution Environments (RPVEE))
INPUT:

- A directed graph $G = (H, E)$
- A function bw : $E \rightarrow \mathbb{R}$
- A set of ordered 3-tuples $A = \{(s_i, d_i, b_i) \mid s_i, d_i \in H;\ b_i \in \mathbb{R};\ i = 1, \ldots, m\}$

OUTPUT: $R : A \rightarrow P$ such that

- $\text{rc}_e \geq 0,\ \forall e \in E,$
- $\sum_{i=1}^{m} \left(brc(R(A_i))\right)$ is maximized

Further, The decision version of RPVEE can be formulated as follows.

Problem 5 (Decision version of Routing Problem In Virtual Execution Environments (RPVEED))
INPUT:

- A directed graph $G = (H, E)$
- A function bw : $E \rightarrow \mathbb{R}$
- A set of ordered 3-tuples $A = \{(s_i, d_i, b_i) \mid s_i, d_i \in H;\ b_i \in \mathbb{R};\ i = 1, \ldots, m\}$
- $\alpha \in \mathbb{R}$

OUTPUT:

- YES, if there exists a routing $R : A \rightarrow P$ such that
  - $\text{rc}_e \geq 0,\ \forall e \in E;$
  - $\sum_{i=1}^{m} \left(brc(R(A_i))\right) \geq \alpha$
- NO, otherwise

For the proofs of hardness we will reduce the Edge Disjoint Path Problem to the Routing Problem in Virtual Execution Environments. The edge disjoint problem has been shown to be NP-complete [24] and NP-hard to approximate within a factor of $m^{1/2-\delta}$ [26].

The edge disjoint path problem can be formulated as follows.

Problem 6 (The Edge Disjoint Path Problem (EDPP))
INPUT:

- A graph $G=(H,E)$, $|H|=p$, $|E|=q$
- A set of ordered 2-tuples $S=\{(s_i,d_i) \mid s_i,d_i \in H;\ i=1,\ldots,k\}$

OUTPUT:

- The maximum numbers of pairs $(s_i,d_i) \in S$ that can be connected via edge disjoint paths from s_i to d_i in $G=(H,E)$

Further, the decision version of the edge disjoint path problem can be stated as follows.

Problem 7 (Decision version of Edge Disjoint Path Problem (EDPPD))
INPUT:

- A directed graph $G=(H,E)$, $|H|=p$, $|E|=q$
- A set of ordered 2-tuples $S=\{(s_i,d_i) \mid s_i,d_i \in H;\ i=1,\ldots,k\}$

OUTPUT:

- YES, if $\forall (s_i,d_i) \in S$ there exist edge disjoint paths from s_i to d_i in $G=(H,E)$
- NO, otherwise

5.1. *Reduction of the Edge Disjoint Path Problem to the Routing Problem in Virtual Execution Environments*

Given an instance $I=\{S,G=(H,E)\}$ of EDPPD or EDPP we reduce it to an instance $R(I)$ of RPVEE or the instance $R_D(I)$ or RPVEED as follows. Construct a complete directed graph $G'=(H,E')$ where $\text{bw}((u,v))=1+\varepsilon$ for $\varepsilon<1$ if $(u,v) \in E$ and $\text{bw}((u,v))=1$ if $(u,v) \notin E$. Further for all $(s_i,t_i) \in S$, let $(s_i,d_i,1) \in A$ (see Figure 4) to get the instance $R(I)$ for RPVEE. Let $\alpha = k{\cdot}\varepsilon$ to get the instance $R_D(I)$ for RPVEED. The reductions are trivially accomplished in $O(n^2)$ time.

Theorem 1 *MARPVEED is NP-complete.*

Proof Given an instance $I=\{S,G=(V,E)\}$ of EDPPD, construct the instance $R_D(I)$ of RPVEED as described earlier. We now claim that (a) a YES instance of EDPPD yields a YES instance of RPVEED; and (b) a NO instance of EDPPD yields a NO instance of RPVEED;

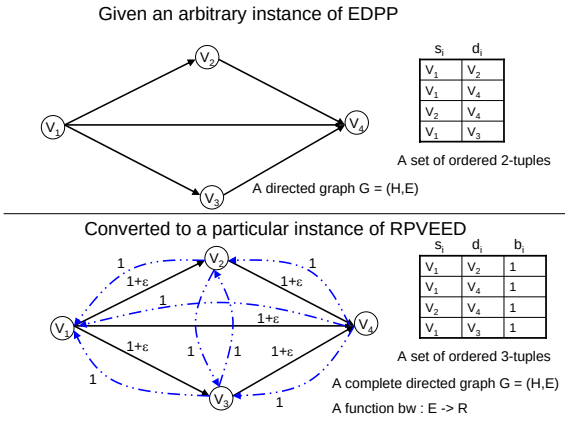


Fig. 4. Reducing EDPPD to RPVEED. The edge weights are bandwidths as specified by the function bw.

The proof for (a) is by construction. Given a YES instance of EDPPD, we know that there exists a set of k edge disjoint paths in G for each of the k (s_i,d_i) tuples in S. Construct the routing R for RPVEED as follows. For every $A_i=(s_i,d_i,1) \in A$, let $R(A_i)$ be the edge disjoint path for the corresponding (s_i,d_i) pair in the EDPPD instance. For every edge e included in the routing, $\text{bw}(e)=1+\varepsilon$. Further, since the routing consists of edge disjoint paths, each edge is assigned to at most one route. Therefore, $\text{rc}_e=(\text{bw}_e-\sum_{e\in R(A_j)} b_j)=\varepsilon$ for all edges $e \in R(A_i) \forall i$. Thus, $\sum_{i=1}^{k}\left(\min_{e\in R(A_i)}\{\text{rc}_e\}\right)=k{\cdot}\varepsilon=\alpha$. Hence, the corresponding instance of RPVEED is a YES instance.

The proof for (b) is by contradiction. Suppose a NO instance of EDPPD yields a YES instance of RPVEED. We will use the YES instance of RPVEED to construct a YES instance of EDPPD. Since the weight of every edge in G' is at most $1+\varepsilon$ and $b_i=1 \forall i$, an edge could belong to at most one route. This implies that all the routes in R are disjoint. Further, since the bottleneck residual capacity for each route $(\min_{e\in R(A_i)}\{\text{rc}_e\})$ could at most be ε and the total residual capacity is at least $\alpha=k{\cdot}\varepsilon$, the residual capacity of each route should be exactly ε. This implies that the bandwidth of each edge in the route is $1+\varepsilon$. Therefore, all the edges included in the routing exist in the graph G and the routes constitute edge disjoint paths in G, thus yielding a YES instance of EDPPD. Hence, the contradiction.

Since RPVEED is a special case of MARPVEED, the NP-completeness of RPVEED immediately implies that MARPVEED is NP-complete. □

6. Hardness of Approximation

A natural way to cope with NP-completeness is to seek approximate solutions instead of exact solutions. An algorithm with approximation ratio C computes, for every problem instance, a solution whose cost is within a factor C of the optimum. In this section, we investigate the approximability of MARPVEE. We show that unless P=NP, there does not exist a polynomial approximation algorithm with an approximation ratio better than $m^{1/2-\delta}$ for any $\delta > 0$.

We again use the edge disjoint problem for the purposes of our reduction. It has been previously shown that the problem is NP-hard to approximate within $m^{1/2-\delta}$ [26]. We will prove an essentially matching hardness result on the optimization version of the routing problem RPVEE and then use that result to prove the same bounds for MARPVEE.

6.1. *Hardness of approximation of RPVEE*

For establishing the hardness of approximation for RPVEE, we reduce an instance I of EDPP to instance $R(I)$ of RPVEE as described earlier in Section 5.1.

Lemma 1 *If the value of the optimal solution to an instance I of EDPP is k^* then the value of optimal solution to the instance $R(I)$ of RPVEE is $k^* \cdot \varepsilon$.*

Proof Let the value of optimal solution to $R(I)$ be OPT. If there are k^* edge disjoint paths in I the corresponding routes for each of those paths in $R(I)$ will have a bottleneck residual capacity of ε. Therefore, OPT $\geq k^* \cdot \varepsilon$.

Note that for any route in $R(I)$, the bottleneck residual capacity is either 0 or ε. Therefore the total bottleneck residual capacity is a factor of ε. Let OPT $= z \cdot \varepsilon$. We then need to show that $z \leq k^*$. Since a route with a bottleneck residual capacity of ε consists of only the edges in the input graph to I and no two routes share a common edge, there are at least z disjoint paths in I. Since the value of optimal solution to I is k^*, $z \leq k^*$. Hence, we are done. □

Theorem 2 *For any $\delta > 0$, it is not possible to approximate RPVEE within a factor of $m^{1/2-\delta}$ unless P=NP.*

Proof We will prove this by contradiction. Let us assume that there exists a polynomial time approximation algorithm A for RPVEE that achieves an approximation guarantee of factor $m^{1/2-\delta}$. Using Lemma 1, algorithm A in conjunction with the reduction R yields a polynomial time $m^{1/2-\delta}$-approximation algorithm for EDPP which is not possible unless P=NP [26]. □

6.2. *Hardness of approximation of MARPVEE*

We use the inapproximability result obtained above for RPVEE to state the inapproximability result for MAPRVEE with the same bounds. The proof is by contradiction and follows very closely the proof for Theorem 2.

Corollary 1 *For any $\delta > 0$, it is NP-hard to approximate MARPVEE within $m^{1/2-\delta}$ unless P=NP.*

7. Greedy adaptation algorithms

The adaptation problem is not only NP-complete, but is also hard to approximate. We have devised two greedy algorithms for mapping VMs to hosts. One finds all the mappings in a single pass, while the other takes two passes over the input data. We have also adapted Dijkstra's shortest path algorithm [40] that now finds the widest path for an unsplittable network flow. Since MARPVEE involves both, mapping and routing network flows we can first apply the mapping algorithm (either one) followed by the routing algorithm, thus first determining all the VM to host mappings which is then followed by computing the routing. Alternatively we can interleave the two wherein we find a mapping for a pair of communicating VMs immediately followed by finding a path for it over the network, before we map any other VM. The work in this section directly builds upon our previous work [34], but for the sake of completeness we present the entire analysis.

7.1. *Greedy algorithm for mapping VMs to Hosts*

As stated above we have two versions of the algorithm. Algorithm 1 makes a single pass over the input data while Algorithm 2 makes two passes. In both, VMs are mapped onto physical hosts and the input to the algorithm is the application communication behavior as captured by VTTIF and available bandwidth between each pair of VNET daemons, as reported by Wren, both expressed as adjacency lists.

7.2. *A greedy heuristic mapping communicating VMs to paths*

We use a greedy heuristic algorithm (Algorithm 3) to determine a path for each pair of communicating VMs. As above we use VTTIF and Wren outputs expressed as adjacency lists as inputs.

Algorithm 1 Greedy One-pass Mapping (GreedyMapOne)

Order the VM adjacency list by decreasing traffic intensity
Order the VNET daemon adjacency list by decreasing throughput
while There is are unmapped VMs **do**
  if both the VMs for a communicating pair are not mapped **then**
    Map them to the first pair of hosts which currently have no VMs mapped onto them
  else
    Map the VM to a VNET daemon such that the throughput estimate between the VM and its already mapped counterpart is maximum
  end if
end while
Compute the difference between the current mapping and the new mapping and issue VM migration instructions to achieve the new mapping.

Algorithm 2 Greedy Two-pass Mapping (GreedyMapTwo)

Order the VM adjacency list by decreasing traffic intensity
Order the VNET daemon adjacency list by decreasing throughput
/* First pass */
while There is a pair of VMs neither of which has been mapped **do**
  Locate the first pair of communicating VMs such that neither of them have been mapped
  Map them to the first pair of hosts which currently have no VMs mapped onto them
end while
/* Second pass */
while There is an unmapped VMs **do**
  Locate a VM that have not been mapped
  Map the VM to a VNET daemon such that the throughput estimate between the VM and its already mapped counterpart is maximum.
end while
Compute the difference between the current mapping and the new mapping and issue VM migration instructions to achieve the new mapping.

7.3. *Adapted Dijkstra's algorithm*

We use a modified version of Dijkstra's algorithm [40] to select a path for each 3-tuple that has the maximum bottleneck bandwidth. This is the "select widest" approach.

We adapt Dijkstra's algorithm for single source shortest path to find the maximum bottleneck bandwidth between each VNET daemon and to find for each 3-tuple $A(s_i, d_i, c_i)$, the widest path $p(i, j)$ with respect to the residual capacity.

Algorithm 3 Greedy Routing (GreedyRouting)

Order the set A of VM to VM communication demands in descending order of communication intensity (VTTIF traffic matrix entry)
while There is are unmapped 3-tuple in A **do**
  Map it to the widest path possible, using an adapted version of Dijkstra's algorithm described later
  Adjust residual capacities in the network adjacency list to reflect the mapping
end while

Dijkstra's algorithm solves the single-source shortest paths problem on a weighted, directed graph $G = (H, E)$. We have created a modified Dijkstra's algorithm that solves the single-source widest paths problem on a weighted directed graph $G = (H, E)$ with a weight function $c : E \rightarrow \mathbb{R}$ which is the available bandwidth in our case.

As in Dijkstra's algorithm we maintain a set U of vertices whose final widest-path weights from source u have already been determined. That is, for all vertices $v \in U$, we have $b[v] = \gamma(u, v)$, where $\gamma(u, v)$ is the widest path value from source u to vertex v. The algorithm repeatedly selects the vertex $w \in H - U$ with the largest widest-path estimate, inserts w into U and relaxes (we slightly modify the original Relax algorithm) all edges leaving w. Just as in the implementation of Dijkstra's algorithm, we maintain a priority queue Q that contains all the vertices in $H - U$, keyed by their b values. This implementation too assumes that graph G is represented by adjacency lists.

Similar to Dijkstra's algorithm we initialize the widest path estimates and the predecessors by the procedure described in Algorithm 4.

Algorithm 4 Initialize(G,u)

1: **for** each *vertex* $v \in H[G]$ **do**
2: {
    $b[v] \leftarrow 0$
    $\pi[v] \leftarrow NIL$
  }
3: **end for**
4: $b[u] \leftarrow \infty$

The modified process of relaxing an edge (w, v) consists of testing whether the bottleneck bandwidth decreases for a path from source u to vertex v by going through w, if it does, then we update $b[v]$ and $\pi[v]$. This

procedure is described in Algorithm 5

Algorithm 5 ModifiedRelax(w,v,c)

1: **if** $b[v] < min(b[w], c(w,v))$ **then**
2: {
$b[v] \leftarrow min(b[w], c(w,v))$
$\pi[v] \leftarrow w$
}
3: **end if**

We can very easily see the correctness of ModifiedRelax. After relaxing an edge (w,v), we have $b[v] \geq min(b[w], c(w,v))$. As, if $b[v] \leq min(b[w], c(w,v))$, then we would set $b[v]$ to $min(b[w], c(w,v))$ and hence the invariant holds. Further, if $b[v] \geq min(b[w], c(w,v))$ initially, then we do nothing and the invariant still holds.

Algorithm 6 is the adapted version of Dijkstra's algorithm to find the widest path for a single tuple.

Algorithm 6 AdaptedDijkstra(G,c,u)

1: $Initialize(G,u)$
2: $U \leftarrow \emptyset$
3: $Q \leftarrow H[G]$
4: **while** $Q \neq \emptyset$ **do** {loop invariant: $\forall\ v \in U$, $b(v) = \gamma(u,v)$}
5: {
$w \leftarrow ExtractMax(Q)$
$U \leftarrow U \cup w$
6: **for** each *vertex* $v \in Adj[w]$ **do**
7: {
$ModifiedRelax(w,v,c)$
}
8: **end for**
}
9: **end while**

7.4. *Correctness of adapted Dijkstra's algorithm*

Similar to the proof of correctness for Dijkstra's shortest paths algorithm, we can prove that the adapted Dijkstra's algorithm is correct by proving by induction on the size of set U that the invariant, $\forall v \in U$, $b[v] = \gamma(u,v)$, always holds.

Base case: Initially $U = \emptyset$ and the invariant is trivially true.

Inductive step: We assume the invariant to be true for $|U| = i$.

Proof: Assuming the truth of the invariant for $|U| = i$, we need to show that it holds for $|U| = i+1$ as well.

Let v be the $(i+1)^{th}$ vertex extracted from Q and placed in U and let p be the path from u to v with weight $b[v]$. Let w be the vertex just before v in p. Since only those paths to vertices in Q are considered that use vertices from U, $w \in U$ hence by the inductive step we have $b[w] = \gamma(u,w)$.

Next, we can prove that p is the widest path from u to v by contradiction. Let us assume that p is not the widest path and instead p^* is the widest path from u to v. Since this path connects a vertex in U to a vertex in $H-U$, there must be a first edge, $(x,y) \in p^*$ where $x \in U$ and $y \in H-U$. Hence the path p^* can now be represented as $p_1.(x,y).p_2$. By the inductive hypothesis $b[x] = \gamma(u,x)$ and since p^* is the widest path, it follows that $p_1.(x,y)$ must be the widest path from w to y, as if there had been a path with higher bottleneck bandwidth, that would have contradicted the optimality of p^*. When the edge x was placed in U, the edge (x,y) was relaxed and hence $b[y] = \gamma(u,y)$. Since v was the $(i+1)^{th}$ vertex chosen from Q while y was still in Q, it implies that $b[v] \geq b[y]$. Since we do not have any negative edge weights and $\gamma(s,v)$ is the bottleneck bandwidth on p^*, that combined with the previous expression gives us bottleneck bandwidth of $p^* \leq b[v]$ which is the bottleneck bandwidth of path p. This contradicts our first assumption that path p^* is wider than path p.

Since we have proved that the invariant holds for the base case and that the truth of the invariant for $|U| = i$ implies the truth of the invariant for $|U| = i+1$, we have proved the correctness of the adapted Dijkstra's algorithm using mathematical induction.

7.5. *Complexity of adapted Dijkstra's algorithm*

Similar to Dijkstra, it can be shown that the running time of the adapted Dijkstra's algorithm is $O(H^2 + E)$. This bound can be reduced by a faster implementation of the priority queue Q.

8. Evaluation of the greedy algorithms

We evaluated two different combinations of our greedy algorithms. We first state the two combinations and next compare the combinations with each other for different problem instances.

GreedyMapOne followed by GreedyRouting: In this, we first run the two pass mapping algorithm to compute all the VM to host mappings and follow that by running the greedy routing algorithm to map

the tuples to paths in the network.

GreedyMapTwo followed by GreedyRouting: In this, we first run the two pass mapping algorithm to compute all the VM to host mapping and follow that by running the greedy routing algorithm to map the tuples to paths in the network.

GreedyMapOne interleaved with GreedyRouting: In this, for each application 3-tuple, we first use the one-pass mapping algorithm and find a mapping from VM to hosts and then immediately we map it greedily to a path in the network. We then repeat the same for each of the remaining application three tuples.

GreedyMapTwo interleaved with GreedyRouting: In this, for each application 3-tuple, we first use the two-pass mapping algorithm and find a mapping from VM to hosts and then immediately we map it greedily to a path in the network. We then repeat the same for each of the remaining application three tuples.

We have previously [34] presented a detailed evaluation of a simulated annealing heuristic and compared it to a greedy heuristic. In search heuristics such as simulated annealing the objective function and constraints can be readily changed without having to change the optimization system. It is also more amenable to multi-objective optimization. We found that the simulated annealing heuristic took a long time to complete as compared to the greedy approach, however, producing better results in certain cases.In this work we try to conduct a more detailed study of the different variations of the greedy strategy.

We implemented an evaluator that was used to calculate the residual bandwidth for multiple test cases. We evaluated the four algorithm variations in three different settings, a real world scenario, randomly generated topologies and smaller topologies created by hand. The table in Figure 5 summarizes our findings.

8.1. *Randomly generated topologies*

We used BRITE [41] to generate network topologies. BRITE was chosen because of its ability to annotate topology maps with bandwidth capacity for each link, this was necessary as the algorithms were designed to maximize the sum of the bottleneck residual capacities. Similarly we generated VM communication topologies using a random generator which we developed. These VM topologies were generated to match the data collected by our VTTIF aggregation tool. We studied a large number of cases with different topology maps to determine whether one algorithm version was superior. The results demonstrated that no variation out-performed the others in any of the cases. However, qualitative reasoning about the algorithms shows that, at least for simple cases, the 2-pass variation is susceptible to clustering.

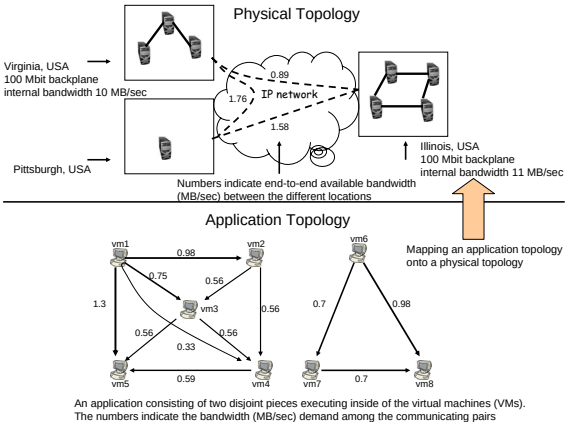


Fig. 6. Experimental setup.

8.2. *Smaller topologies created by hand*

To understand the differences between the 1-pass and 2-pass variations we developed simple test cases by hand demonstrating clustered topologies. The difference arises when the algorithms are faced with clustered topologies. For a simple scenario consider the case of two sites, each with 4 physical machines connected to a high capacity LAN and connected to the other site via a low capacity WAN Internet connection. Now consider two independent VM sets, one with 3 VMs and the other with 2, which have large amounts of communication traffic inside each set but no traffic to the other set. Ideally each VM set would be mapped onto different physical sites, such that the low capacity WAN connection would never be used. All of the algorithm variations are susceptible to incorrectly mapping this scenario, however the 2-pass variation is the most susceptible for these cases.

In order to evaluate the algorithms we created by hand the scenario described above, as well as several other variations on the scenario, and evaluated the performance of each algorithm. The results clearly show that the 2-pass version is the most susceptible to creating an inefficient mapping.

	GreedyMapOne (Algorithm 1)		GreedyMapTwo (Algorithm 2)	
	Followed by GreedyRouting	Interleaved with GreedyRouting	Followed by GreedyRouting	Interleaved with GreedyRouting
Real world example	94	94	34	34
Random BRITE topology	237	237	237	237.12
Clustered topology	56	56	38	38

Fig. 5. Example results from our four different algorithm variations. The values represent the objective function being maximized, the sum of residual bottleneck bandwidths over all the mapped paths in MB/s.

8.3. *A real world scenario*

We were also able to analyze the algorithms on real world topology data. Figure 6 illustrates our experimental setup. Using bandwidth measurements and VM traffic aggregations previously collected we were able to evaluate the algorithms on actual scenarios. The bandwidth data was collected from physical machines hosted at CMU, College of William and Mary and Northwestern University that had been used in earlier experiments. The VM traffic aggregations was collected by running several benchmarking tools on actual VMs. The results of this evaluation again show that the 1-pass algorithm is clearly superior when clustered topologies are present.

The table in Figure 5 summarizes our findings. For randomly generated topologies we do not see any differences between the different variations. However for the topology created by hand and for the real world scenario that result in a clustered setting, the 1-pass variation outperforms the 2-pass algorithm. Further, we did not notice in difference between the interleaved and non-interleaved variations.

9. Conclusion

The decade gone by has seen the emergence of a powerful computing paradigm, wide-area distributed computing. An application running in any distributed environment must adapt to available resources. However, until recently, all adaptation attempts had remained application specific requiring direct user involvement. Since then it has been shown that virtual execution environments consisting of virtual machines inter-connected by virtual networks provide opportunities to dynamically optimize, at run-time, the performance of existing, *unmodified* distributed applications running on existing, unmodified operating systems without any user or programmer intervention. Efficient and effective adaptation algorithms in such environments will help realize the full potential of wide-are distributed computing.

We formalized the adaptation problem that arose in such environments. We have shown that the adaptation problem is NP-hard. Further, we have shown that it hard to find efficient approximate solutions for it. In particular we have proven that it is NP-hard to approximate within a factor of $m^{1/2-\delta}$ for any $\delta > 0$, where m is the number of edges in the virtual overlay graph. We presented greedy adaptation algorithms for the mapping and routing components of the problem. We evaluated four different combinations of the algorithms and found them to perform well in practice. We are currently focusing on researching the feasibility of a single optimization metric that would be effective for a range of distributed applications.

References

[1] P. Homburg, M. V. Steen, A. Tanenbaum, An architecture for a wide area distributed system, in: Proceedings of the Seventh ACM SIGOPS European Workshop, 1996, pp. 75–82.

[2] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the grid: Enabling scalable virtual organizations, International Journal of Supercomputer Applications 15 (3).

[3] A. Grimshaw, W. Wulf, the Legion Team, The legion vision of a worldwide virtual computer, Communications of the ACM 40 (1).

[4] C. Tapus, I.-H. Chung, J. Hollingsworth, Active harmony: Towards automated performance tuning, in: Proceedings of the ACM/IEEE Conference on Supercomputing, 2002, pp. 1–11.

[5] T. Kichkaylo, V. Karamcheti, Optimal resource-aware deployment planning for component-based distributed applications, in: Proceedings of HPDC, 2004, pp. 150–159.

[6] VMware Corporation, http://www.vmware.com.

[7] R. Figueiredo, P. A. Dinda, J. Fortes, A case for grid computing on virtual machines, in: Proceedings of ICDCS, 2003.

[8] X. Jiang, D. Xu, Soda: A service-on-demand architecture for application service hosting platforms, in: Proceedings of HPDC, 2003.

[9] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, D. Boneh, Terra: A virtual machine-based platform for trusted computing, in: Proceedings of SOSP, 2003, pp. 193–206.

[10] A. I. Sundararaj, A. Gupta, P. A. Dinda, Dynamic topology adaptation of virtual networks of virtual machines, in: Proceedings of LCR, 2004.

[11] A. I. Sundararaj, A. Gupta, P. A. Dinda, Increasing application performance in virtual environments through run-time inference and adaptation, in: Proceedings of HPDC, 2005.

[12] J. Sugerman, G. Venkitachalan, B.-H. Lim, Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor, in: Proceedings of the USENIX Annual Technical Conference, 2001.

[13] A. Hansson, K. Goossens, A. Radulescu, A unified approach to constrained mapping and routing on network-on-chip

architectures, in: Proceedings of the 3rd IEEE/ACM/IFIP CODES+ISSS, 2005.

[14] M. Harchol-Balter, A. B. Downey, Exploiting process lifetime distributions for dynamic load balancing, in: Proceedings of ACM SIGMETRICS, 1996.

[15] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, K. R. Walker, Agile application-aware adaptation for mobility, in: Proceedings of ACM SOSP, 1997.

[16] B. Siegell, P. Steenkiste, Automatic generation of parallel programs with dynamic load balancing, in: Proceedings of the Third International Symposium on High-Performance Distributed Computing (HPDC), 1994, pp. 166–175.

[17] A. S. Grimshaw, W. T. Strayer, P.Narayan, Dynamic object-oriented parallel processing, IEEE Parallel and Distributed Technology: Systems and Applications (1993) 33–47.

[18] J. A. Zinky, D. E. Bakken, R. E. Schantz, Architectural support for quality of service for CORBA objects, Theory and Practice of Object Systems 3 (1) (1997) 55–73.

[19] R. Figueiredo, P. Dinda, J. Fortes, Resource virtualization renaissance, IEEE Computer Special Issue On Resource Virtualization 38 (5) (2005) 28–31.

[20] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, in: Proceedings of ACM SOSP, 2003, pp. 164–177.

[21] A. I. Sundararaj, P. A. Dinda, Towards virtual networks for virtual machine grid computing, in: Proceedings of the 3rd USENIX Virtual Machine Research and Technology Symposium (VM), 2004.

[22] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, M. Wawrzoniak, Operating system support for planetary-scale network services, in: Proceedings of USENIX NSDI, 2004.

[23] A. I. Sundararaj, M. Sanghi, J. R. Lange, P. A. Dinda, An optimization problem in adaptive virtual environments, ACM SIGMETRICS Performance Evaluation Review 33 (2).

[24] R. Karp, Compexity of Computer Computations, Plenum Press, New York, 1972, Ch. Reducibility among combinatorial problems, pp. 85–103.

[25] J. Kleinberg, Approximation algorithms for disjoint paths problems, Ph.D. thesis, Massachusetts Institute of Technology, EECS Department (1996).

[26] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, M. Yannakakis, Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems, Journal of Computer and System Sciences 67 (3) (2003) 473–496.

[27] A. Baveja, A. Srinivasan, Approximation algorithms for disjoint paths and related routing and packing problems, Mathematics of Operations Research 25 (2) (2000) 255–280.

[28] S. G. Kolliopoulos, C. Stein, Approximating disjoint-path problems using greedy algorithms and packing integer programs, in: Proceedings of the 6th International IPCO Conference on Integer Programming and Combinatorial Optimization, Springer-Verlag, London, UK, 1998, pp. 153–168.

[29] Y. Azar, O. Regev, Strongly polynomial algorithms for the unsplittable flow problem, in: Proceedings of IPCO, 2001.

[30] K. Kwong, A. Ishfaq, Benchmarking and comparison of the task graph scheduling algorithms, Journal of Parallel and Distributed Computing 59 (3) (1999) 381–422.

[31] A. Shoykhet, J. Lange, P. Dinda, Virtuoso: A system for virtual machine marketplaces, Tech. Rep. NWU-CS-04-39, Department of Computer Science, Northwestern University (July 2004).

[32] A. Gupta, P. A. Dinda, Infering the topology and traffic load of parallel programs running in a virtual machine environment, in: Proceedings of JSSPP, 2004.

[33] M. Zangrilli, B. B. Lowekamp, Using passive traces of application traffic in a network monitoring system, in: Proceedings of HPDC, 2004.

[34] A. Gupta, M. Zangrilli, A. I. Sundararaj, A. Huang, P. Dinda, B. Lowekamp, Free network measurment for adaptive virtualized distributed computing, in: Proceedings of IPDPS, 2006.

[35] J. R. Lange, A. I. Sundararaj, P. A. Dinda, Automatic dynamic run-time optical network reservations, in: Proceedings of the HPDC, 2005.

[36] B. Lin, P. A. Dinda, Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling, in: Proceedings of ACM/IEEE SC, 2005.

[37] C. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. Lam, M. Rosenblum, Optimizing the migration of virtual computers, in: Proceedings of OSDI, 2002.

[38] M. Kozuch, M. Satyanarayanan, T. Bressoud, Y. Ke, Efficient state transfer for Internet suspend/resume, Tech. Rep. IRP-TR-02-03, Intel Research Laboratory at Pittburgh (May 2002).

[39] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, Live migration of virtual machines, in: Proceedings of NSDI, 2005.

[40] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, Second Edition, MIT Press and McGraw-Hill, 2001.

[41] A. Medina, A. Lakhina, I. Matta, J. Byers, BRITE: Universal topology generation from a user's perspective, Tech. Rep. BU-CS-TR-2001-003, Computer Science Department, Boston University (April 2001).