An Optimization Problem in Adaptive Virtual Environments

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

ABSTRACT

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 and inference and application-independent adaptation mechanisms, efficient adaptation algorithms are key to the success of such an effort. In previous work we have described our measurement and inference framework, explained our adaptation mechanisms, and proposed simple heuristics as adaptation algorithms. Though we were successful in improving performance as compared to the case with no adaptation, none of our algorithms were characterized by theoretically proven bounds. In this paper, we formalize the adaptation problem, show that it is NP-hard and propose research directions for coming up with an efficient solution.

1. INTRODUCTION

Virtual machines greatly simplify wide-area distributed computing by lowering the abstraction to benefit both resource users and providers [1, 4]. We have been developing a middleware system, Virtuoso, for virtual machine grid computing. Virtuoso, for a user, very closely emulates the existing process of buying, configuring, and using a computer or a collection of computers from a web site. Instead of a physical computer, the user receives a reference to the virtual machine which he can then use to start, stop, reset, and clone the machine.

The nature of the network presence that the virtual machine gets depends solely on the policies of the remote site. To deal with this network problem we developed VNET [8], a simple layer two virtual network tool. VNET is ideally placed to monitor the resource demands of the VMs. The VTTIF (Virtual Topology and Traffic Inference Framework) component of Virtuoso achieves this [2]. In addition, the execution environment can use the naturally occurring traffic of existing, unmodified applications running inside of the VMs to measure the characteristics of the underlying physical network.

2. DYNAMIC ADAPTATION PROBLEM IN VIRTUAL EXECUTION ENVIRONMENTS

Any application running in a distributed environment must adapt to the continuously changing network and computing resources. Despite many efforts, adaptation has remained very application specific.

A virtual execution environment, such as Virtuoso [1], provides 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. However, a number of challenges must first be met. Figure 1 illustrates a simplified adaptation scenario wherein a greedy heuristic drives overlay topology and routing changes, leveraging data inferred by VTTIF.

Measurement and inference: This involves (a) measuring the traffic load and topology of applications running inside the virtual machines, (b) monitoring the underlying network and inferring its topology, bandwidth and latency characteristics, and (c) measuring host and VM characteristics such as their size, compute capacities and demands. In previous work [2, 3] we have shown how to successfully accomplish these tasks.

Adaptation mechanisms: A wide variety of adaptation mechanisms are possible in the context of virtual execution environments, such as (a) VM migration, (b) overlay topology and routing changes, and (c) network and CPU resource reservation where possible. These have been described previously [9, 6, 7].

Adaptation algorithm: Most importantly, we need an efficient algorithm to drive the adaptation mechanisms while guided by the measured and inferred data. Note that there may be a variety of reasonable adaptation goals in terms of latency, throughput, congestion, workload etc. For this paper we formulate the problem with the goal of maximizing application throughput.

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. VNET also 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 also collects 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, size

[1] http://virtuoso.cs.northwestern.edu

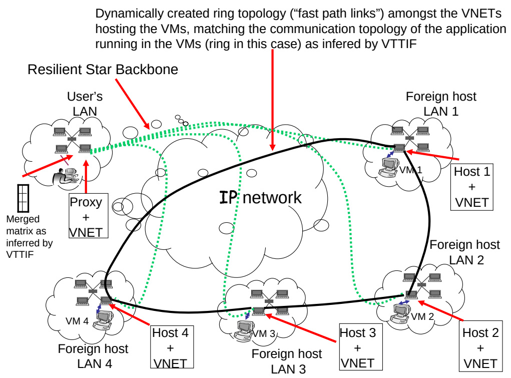

Dynamically created ring topology (“fast path links”) amongst the VNETs hosting the VMs, matching the communication topology of the application running in the VMs (ring in this case) as infered by VTTIF
Resilient Star Backbone
User's LAN
Foreign host LAN 1
VM 1
Host 1 + VNET
IP network
Proxy + VNET
Merged matrix as inferred by VTTIF
Foreign host LAN 2
VM 4
VM 3
VM 2
Host 4 + VNET
Host 3 + VNET
Host 2 + VNET
Foreign host LAN 4
Foreign host LAN 3

Figure 1: As the application progresses VNET 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.

: $H \rightarrow \mathbb{R}$. 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 are also 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.

VTTIF infers the application communication topology in order to generate the traffic requirements of the application, $\mathcal{A}$, 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.

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

- a mapping from VMs to hosts, $\text{vmap} : VM \rightarrow H$, meeting the size and compute capacity demands of the VMs within the host constraints. Further, we may also be given a set of constraint mappings from VMs to hosts that have to be maintained at all times, represented by a set of ordered pairs $M_i = (\text{vm}_i, h_i)$, $\text{vm}_i \in \text{VM}$, $h_i \in H$.

- a routing, $R : \mathcal{A} \rightarrow \mathcal{P}$, where $\mathcal{P}$ 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.

Once all the mappings and paths have been decided, each VNET edge will have a residual capacity, rc_e, which is the bandwidth remaining unutilized 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 define its bottleneck bandwidth, $\text{bb}\big(R(A_i)\big) = \min_{e \in R(A_i)} \left\{\text{rc}_e\right\}$ and its total latency, $\text{tl}\big(R(A_i)\big) = \sum_{e \in R(A_i)} \big(late_e\big)$

The aim of the adaptation algorithm 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 performance within the current configuration thereby increasing application throughput.

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

INPUT:

- A directed graph $G = (H, E)$
- A function $\text{bw} : E \rightarrow \mathbb{R}$
- A function $\text{lat} : E \rightarrow \mathbb{R}$
- A function $\text{compute} : H \rightarrow \mathbb{R}$
- A function $\text{size} : H \rightarrow \mathbb{R}$
- A set, $\text{VM} = (\text{vm}_1, \text{vm}_2 \ldots \text{vm}_n)$, $n \in \mathbb{N}$
- A function $\text{vm\_compute} : \text{VM} \rightarrow \mathbb{R}$
- A function $\text{vm\_size} : \text{VM} \rightarrow \mathbb{R}$
- A set of ordered 4-tuples
  $\mathcal{A} = \{(s_i, d_i, b_i, l_i) \mid s_i, d_i \in \text{VM};\ b_i, l_i \in \mathbb{R};\ i = 1, \ldots, m\}$
- A set of ordered pairs
  $\mathcal{M} = \{(\text{vm}_i, h_i) \mid \text{vm}_i \in \text{VM}, h_i \in H;\ i = 1, 2 \ldots r, r \leq n\}$

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

- $\sum_{\text{vmap(vm)}=h} \big(\text{vm\_compute(vm)}\big) \leq \text{compute(h)}, \forall\ h \in H$
- $\sum_{\text{vmap(vm)}=h} \big(\text{vm\_size(vm)}\big) \leq \text{size}(h), \forall\ h \in H$
- $h_i = \text{vmap}(\text{vm}_i), \forall M_i = (\text{vm}_i, h_i) \in \mathcal{M}$
- $\big(bw_e - \sum_{e \in R(A_i)} b_i\big) \geq 0, \forall e \in E$
- $\big(\sum_{e \in R(A_i)} \text{late}\big) \leq l_i, \forall e \in E$
- $\sum_{i=1}^{m} \big(\min_{e \in R(A_i)} \left\{\text{rc}_e\right\}\big)$, where $\text{rc}_e = (\text{bw}_e - \sum_{e \in R(A_i)} b_i)$, is maximized

3. A SPECIAL CASE OF THE ADAPTATION PROBLEM

The generic adaptation problem seeks a mapping, vmap, from VMs to hosts and a routing, R, of VM traffic over the overlay network, G. To establish the hardness of the problem, we consider a special case of the problem wherein all the VM to host mappings are constrained by the ordered pairs $\mathcal{M}$ and latency demands are dropped, 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.

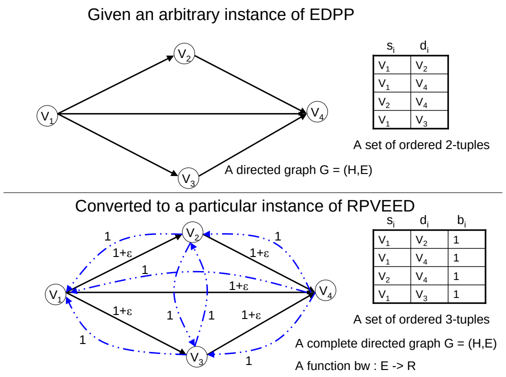
Problem 2 (Routing Problem In Virtual Execution Environments (RPVEE))

INPUT:

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

OUTPUT: $R : \mathcal{A} \rightarrow \mathcal{P}$ such that

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


Given an arbitrary instance of EDPP
V_1
V_2
V_3
V_4
A directed graph G = (H,E)
A set of ordered 2-tuples
Converted to a particular instance of RPVEED
1
1+ε
1
1+ε
1
1+ε
1+ε
1
1
1+ε
1
1
A set of ordered 3-tuples
A complete directed graph G = (H,E)
A function bw : E -> R

s_i	d_i
V_1	V_2
V_1	V_4
V_2	V_4
V_1	V_3

s_i	d_i	b_i
V_1	V_2	1
V_1	V_4	1
V_2	V_4	1
V_1	V_3	1

Figure 2: Reducing EDPP to RPVEED. The edge weights are bandwidths as specified by the function bw.

4. ANALYSIS

Theorem 1. *RPVEE is NP-hard.*

The NP-hardness for the problem is established by reduction from the Edge Disjoint Path Problem (EDPP) which is shown to be NP-complete in [5]. In the interest of space we provide only a brief sketch of the reduction here.[2]

Problem 3 (Edge Disjoint Path Problem (EDPP))

INPUT:

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

OUTPUT:

- Yes, if and only if $\forall (s_i, d_i) \in \mathcal{S}$ their exist edge disjoint paths from s_i to d_i in $G = (H, E)$
- No, otherwise

For reducing EDPP to an instance of the decision version of RPVEE (RPVEED), construct a complete graph $G' = (V, E')$ where $\mathrm{bw}((u, v)) = 1 + \varepsilon$ if $(u, v) \in E$ and $\mathrm{bw}((u, v)) = 1$ if $(u, v) \notin E$. Further for all $(s_i, d_i) \in \mathcal{S}$, let $(s_i, d_i, 1) \in \mathcal{A}$. Figure 2 illustrates this reduction. Note that there exists edge disjoint paths for the EDPP if and only if the sum of bottleneck bandwidths in the instance of RPVEED is $k{\cdot}\varepsilon$.

Since GAPVEE is a special case of RPVEE, the following theorem immediately follows.

Theorem 2. *GAPVEE is NP-hard.*

5. STATUS

We have previously developed a variety of heuristics to drive the adaptation mechanisms [9]. Though they were successful in improving performance relative to the naive approach (with no adaptation), we believe there is significant scope for improvement. Therefore, as a first step, we have formalized the adaptation problem and given preliminary results of its hardness.

Even though the problem in this most generic incarnation is computationally hard, special cases of the problem amenable to efficient solutions will be of significant interest as we have a working system wherein various adaptation algorithms can be deployed and studied conveniently.

There are a number of well studied variants of our problem such as routing un-splittable flows. We believe that the rich collection of approximation algorithms already available for them can be adapted to our specific problem. Accordingly, we have begun work in that direction.

6. CONCLUSION

We formalize the adaptation problem that arises in virtual execution environments consisting of virtual machines interconnected by virtual networks. Such a platform provides 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. We show that the adaptation problem is NP-hard and propose potential research directions for coming up with efficient solutions to certain interesting special cases.

7. REFERENCES

[1] R. Figueiredo, P. A. Dinda, and J. Fortes. A case for grid computing on virtual machines. In *Proceedings of ICDCS*, May 2003.

[2] A. Gupta and P. A. Dinda. Infering the topology and traffic load of parallel programs running in a virtual machine environment. In *Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Program Processing(JSSPP)*, June 2004.

[3] A. Gupta, M. Zangrilli, A. I. Sundararaj, P. Dinda, and B. Lowekamp. Free network measurment for adaptive virtualized distributed computing. Technical Report NWU-CS-05-13, June 2005.

[4] X. Jiang and D. Xu. Soda: A service-on-demand architecture for application service hosting platforms. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 174–183, June 2003.

[5] R. Karp. *Compexity of Computer Computations*, chapter Reducibility among combinatorial problems, pages 85–103. Plenum Press, New York, 1972.

[6] J. R. Lange, A. I. Sundararaj, and P. A. Dinda. Automatic dynamic run-time optical network reservations. In *Proceedings of the 14th International Symposium on High Performance Distributed Computing (HPDC)*, July 2005.

[7] B. Lin and P. A. Dinda. Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling. In *Proceedings of ACM/IEEE SC 2005 (Supercomputing)*, 2005.

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

[9] A. I. Sundararaj, A. Gupta, and P. A. Dinda. Increasing application performance in virtual environments through run-time inference and adaptation. In *Proceedings of the 14th International Symposium on High Performance Distributed Computing (HPDC)*, July 2005.

[2] Complete proofs at: http://virtuoso.cs.northwestern.edu/