Inferring the Topology and Traffic Load of Parallel Programs Running in a Virtual Machine Environment

Ashish Gupta and Peter A. Dinda

Department of Computer Science, Northwestern University
{ashish,pdinda}@cs.northwestern.edu

Abstract. We are developing a distributed computing environment based on virtual machines featuring application monitoring, network monitoring, and an adaptive virtual network. In this paper, we describe our initial results in monitoring the communication traffic of parallel applications, and inferring its spatial communication properties. The ultimate goal is to be able to exploit such knowledge to maximize the parallel efficiency of the running parallel application by using VM migration, virtual overlay network configuration and network reservation techniques, which are a part of the distributed computing environment. Specifically, we demonstrate that: (1) we can monitor the parallel application network traffic in our layer 2 virtual network system with very low overhead, (2) we can aggregate the monitoring information captured on each host machine to form a global picture of the parallel application's traffic load matrix, and (3) we can infer from the traffic load matrix the application topology. In earlier work, we have demonstrated that we can capture the time dynamics of the applications. We begin here by considering offline traffic monitoring and inference as a proof of concept, testing it with a variety of synthetic and actual workloads. Next, we describe the design and implementation of our online system, the Virtual Topology and Traffic Inference Framework (VTTIF), and evaluate it using a NAS benchmark.

1 Introduction

Virtual machines have the potential to simplify the use of distributed resources in a way unlike any other technology available today, making it possible to run diverse applications with high performance after only minimal or no programmer and administrator effort. Network and host bottlenecks, difficult placement decisions, and firewall obstacles are routinely encountered, making effective use of distributed resources an obstacle to innovative science. Such problems, and the human effort needed to work around them, limit the development, deployment, and scalability of distributed parallel applications.

We have presented a detailed case for virtual machine-based distributed and parallel computing [4], and we are now developing a system, Virtuoso, which has the following model:

Effort sponsored by the National Science Foundation under Grants ANI-0093221, ACI-0112891, ANI-0301108, EIA-0130869, 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).

D. Feitelson, L. Rudolph, and U. Schwiegelshohn (Eds.): JSSPP 2004, LNCS 3277, pp. 125–143, 2005.

- The user receives what appears to be a new computer or computers on his network at very low cost. The user can install, use, and customize the operating system, environment, and applications with full administrative control.
- The user chooses where to execute the virtual machines. Checkpointing and migration is handled efficiently through Virtuoso. The user can delegate these decisions to the system.
- A service provider need only install the VM management software to support a diverse set of users and applications.
- Monitoring, adaptation, resource reservation, and other services are retrofitted to existing applications at the VM level with no modification of the application code, resulting in broad application of these technologies with minimal application programmer involvement.

An important element of our system is a layer 2 virtual network, VNET, which we initially developed to create the "networking illusion" needed for the first element of the model. It can "move" a set of virtual machines in a WAN environment to the user's local layer 2 domain. We are now expanding VNET into a tool that supports arbitrary overlay topologies and routing rules, passive application and network monitoring, adaptation (based on VM migration and topology/routing changes), and resource reservation. VNET is described in detail in a previous paper [12].

This paper reports on one of our first steps toward achieving the last element of the model. The question we address in particular is: can we monitor, with low overhead and no application or operating system modifications, the communication traffic of a parallel application running in a set of virtual machines interconnected with a virtual network, and compute from it the traffic load matrix and application communication topology? Our initial results demonstrate that this is possible. We are integrating the online implementation of our ideas, VTTIF (Virtual Topology and Traffic Inference Framework), into the evolving VNET system.

We consider here Bulk-Synchronous Parallel [6] (BSP) style applications. Specifically, we consider parallel programs whose execution alternates between one or more computing phases and one or more communication phases, including metaphases. We are testing whether our results hold for more general applications. In earlier work, we have demonstrated that the network traffic of compiler-parallelized BSP applications, when measured using techniques similar to those used here, exhibits clear time dynamical structure (periodicity with harmonics) [3]. Our results here show that we can quickly and efficiently recover its spatial structure, its topology and traffic load, as well.

The ultimate motivation behind recovering the spatial and temporal properties of a parallel application running in a virtual environment is to be able to maximize the parallel efficiency of the running application by migrating its VMs, changing the topology and routing rules of the communication network, and taking advantage of underlying network reservations on the application's behalf.

A parallel program may employ various communication patterns for its execution. A communication pattern consists of a list of all the message exchanges of a representative processor during a communication phase. The result of each processor executing