Mobitopolo: A Portable Infrastructure to Facilitate Flexible Deployment and Migration of Distributed Applications with Virtual Topologies

Richard Potter
NICT, Tokyo, Japan
potter@nict.go.jp

Akihiro Nakao
University of Tokyo & NICT, Tokyo, Japan
nakao@iii.u-tokyo.ac.jp

ABSTRACT

Hosting network services on virtual machines has become appealing for provisioning resources, saving power and enabling migration in case of service disruption, especially in data centers. Network services implemented widely over the Internet may enjoy similar benefits, because general-purpose hosting infrastructures such as PlanetLab and Amazon EC2 are available to host them. However, such infrastructures are piecemeal and heterogeneous in terms of virtualization technologies, which makes it hard to use them all together to their full potential. To ease this challenge, we implemented *Mobitopolo*, a *portable* infrastructure service to *deploy and migrate* distributed network services *spanning over heterogeneous hosting infrastructures* while *preserving the logical connections between the service components*. To the best of our knowledge, Mobitopolo is the first virtualized execution environment to integrate all these attributes into one system.

Categories and Subject Descriptors

C.2.6 [**Computer Communication Networks**]: Internetworking; C.2.1 [**Computer Communication Networks**]: Network Architecture and Design

General Terms

Design, Experimentation, Measurement, Performance

Keywords

Virtualization, Internet

1. INTRODUCTION

Virtualization has become a viable technology for provisioning resources, reducing power consumption, and migrating applications in case of service disruption, in data centers where a large number of network services are hosted on top of a multitude of servers. Network services such as web proxies and caches distributed across the Internet may also enjoy similar benefits such as resource provisioning and live migration of network services if deployed on top of distributed virtualized execution environments, now that general-purpose hosting infrastructures such as PlanetLab [3] and Amazon EC2 [1] have become available to host them.

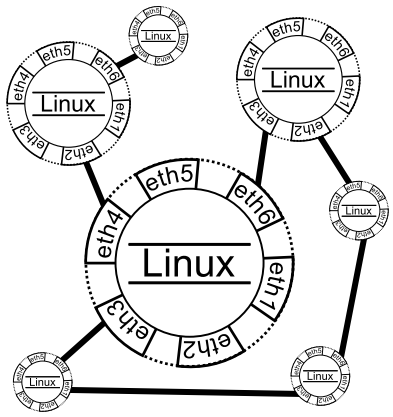

Figure 1: To software inside, Mobitopolo is simply a set of standard Linux environments linked by point-to-point Ethernet connections.

However, such infrastructures are piecemeal and heterogeneous in terms of virtualization technologies, which makes it hard for application developers to use them all together to their full potential, since they have to reconfigure their applications for individual infrastructures.

In this paper, we propose *Mobitopolo*, another layer of infrastructure on top of multiple heterogeneous infrastructures to deploy and migrate network services over the Internet. Mobitopolo is innovative compared to other virtual execution environments in that it is a *portable* infrastructure service to facilitate *deployment and migration* of distributed network services *spanning over heterogeneous hosting infrastructures* while *preserving the logical connections between the service components*. To the best of our knowledge, Mobitopolo is the first virtualized execution environment to integrate all these features into one system.

To increase portability, Mobitopolo is implemented completely in user-mode on Linux, which most hosting infrastructures support. In more detail, Mobitopolo is built on top of SBUML [11, 12, 13], a Linux on Linux virtualization container based on User-Mode Linux [5]. Because SBUML runs strictly on top of normal Linux system calls, it can run inside of almost any Linux host, even if that host Linux is

VISA'09, August 17, 2009, Barcelona, Spain.

itself some type of virtual machine. The portability of Mobitopolo enables snapshot-based deployment and live migration of Linux-based network services across heterogeneous hosting infrastructures.

Mobitopolo keeps the logical connections intact between the service components so that the applications on top of it can use a virtual network topology that stays static, even as services are moved among heterogeneous environments across the world. This feature enables the service components to be migrated inside virtual containers such that connections between them can appear unchanged to the service components. To achieve this feature, point-to-point Ethernet-over-UDP tunnels are implemented in SBUML. The network configuration necessary to connect and reconnect the tunnels is performed outside the virtualization containers of SBUML so that changes are invisible to software running inside, as illustrated in Figure 1.

A key benefit introduced by virtualization technology is that it provides a container complete enough to migrate multiple complex software pieces to another physical machine without breaking internal connections. In a nutshell, Mobitopolo extends this container to comprise multiple Linux machines in different geographical locations that hold connected components of distributed applications. It also makes the components almost free of physical host details, a property that not only enables migration, but also enables many other benefits that apply particularly well to the development and deployment of distributed systems.

The rest of the paper is organized as follows. Section 2 describes the prototype implementation of Mobitopolo. Section 3 explores the benefits a system with Mobitopolo's features can provide using a simple but interesting example. Section 4 discusses performance aspects of the system. Finally, sections for related research and conclusions follow.

2. IMPLEMENTATION

SBUML provides much of Mobitopolo's functionality with its Linux execution environment and snapshot features. The following sections explain the additional implementation effort.

2.1 UDP-Tunnel Extension

SBUML inherits from UML a number of networking interfaces. For our goals, the two with the most potential are *slirp* and *TUN/TAP*. TUN/TAP networking sets up a point-to-point Ethernet tunnel between a UML virtual machine's network device and a *TAP* device on the host. Routing can be set up on the host to send the VM's packets to and from the TAP device. TUN/TAP is the preferred way to provide general network connectivity because of speed. However, for us it is too restrictive because it requires host support for TAP devices and root privileges to set up routing. VServer-based systems such as PlanetLab do not provide such support.

In contrast, UML's slirp interface can be used without root privileges. It gives each VM its own private network and provides NAT functionality to route packets to and from the host's network. Unfortunately, its performance is suboptimal.

With both slirp and TUN/TAP, outside connections are broken when VMs migrate to different subnets. It is possible to use private networks from inside the VMs, however changes to physical topology are no longer transparent to software inside the VMs, which must adjust private network configuration to changes in physical topology. Also, the extra layer slows down performance.

These problems motivated us to make a new UML network device explicitly designed for making private networks based on the existing TUN/TAP driver code. Its device driver code for creating and opening TAP devices was replaced with code for opening UDP socket connections. Once open, TAP devices and socket connections are both represented as file descriptors on the host. Datagrams can be written to and read from these file descriptors in the same way, so little else in the rather complicated TUN/TAP driver code needed to be changed.

2.2 Live Migration Extension

SBUML supports virtual machine snapshots, so it can perform migration by freezing the machine, saving a snapshot, removing the virtual machine, copying the snapshot to another host, and then restoring the snapshot. For this solution, the virtual machine is down for the many minutes necessary to do all five steps. Many virtual machine implementations have *live migration* features to minimize the downtime by iteratively precopying the virtual machine image while the machine is still running. This is a tricky technique because a running VM causes the destination image to contain inconsistent, old data. Nevertheless, some or most of the destination VM image is correct, so subsequent copy operations can copy less and be faster. During the last iteration, the machine only needs to be frozen for a short time while a much smaller portion is transferred. With identical source and destination images, the source machine can be removed and the destination machine started up.

The amount of downtime is determined in part by the speed to compute virtual machine image changes. The technique we chose for SBUML is to modify the *tar* utility to produce archives that contain only block level differences between two directories. This works well because all of a running SBUML VM's state is contained in a single directory on the host machine. It can be done quickly because the sizes of these directories are typically much smaller than that of host main memory. Therefore after one or two precopy iterations, all of the VM is cached in host memory and the final comparison of the few hundred megabytes in main memory can be done in less than half a second. For VMs that are not aggressively changing state during WAN migration, downtime is about one or two seconds. If state is changing during migration, the downtime is determined by how much state is changing and the bandwidth between source and destination hosts.

Most VM migration solutions are for LAN environments where it is not necessary to move the VM's disk state, because the VM can use the same network mounted disk before and after migration [4]. Therefore, fast migration in LAN environments can exclusively focus on how to quickly compute RAM changes and can use very quick techniques that use memory management hardware. In contrast, the special tar archive solution used by SBUML is slower but generalizes to 100% of the machine state, including the disk data necessary for WAN migration.

2.3 Central Management

Although SBUML had existing features to download and demand fetch virtual machines over HTTP, it did not have

features to control running virtual machines remotely. For Mobitopolo, these were added in a layered way.

2.3.1 Remote Host Commands

The lowest layer of Mobitopolo's central management is a command (*sbuml–remotecmd*) to execute a shell command inside an SBUML environment on a remote machine. For parameters, sbuml–remotecmd is given an ssh command that can connect to the remote user account and the directory path to the SBUML installation. To make it comfortable to use this command interactively, the ssh and path information can be saved into an alias file. Negotiating secure connections for each command can take significant time. Fortunately, ssh has a feature to open permanent secure connections that can be used to send future commands more quickly. Sbuml–remotecmd is responsible for automatically opening and reusing these connections.

2.3.2 Remote Guest Commands

Another command (*sbumlguestexec*) executes shell commands inside a virtual machine. Because these shell commands are used to set up networking, it is important that sbumlguestexec itself not rely on networking. Therefore it is implemented by using UML's hostfs feature to poll special files on the host for new shell commands to execute. For remote virtual machines, this command is used in combination with sbuml–remotecmd.

2.3.3 Objects as Proxy Representations

The top layer is a centralized data structure for holding the status of the globally distributed machines and tunnels. We call it the "control tree" because it is implemented as a normal UNIX file directory tree. The top level includes directories for `nodes`, `vms`, `udp-tunnels`, and `udp-halfs`. Inside each of these directories is another directory for each object of that type that is part of the distributed system.

Inside each object directory is a collection of files that store information about the remote object that the directory represents. Relationships between objects are represented as soft file links. For example, `nodes/princeton/` might represent a user account on a PlanetLab server at Princeton, and `vms/testvm/` might represent a virtual machine on Princeton's server. In that case, `vms/testvm/atnode/` will be a soft link to the `nodes/princeton/` directory. Each udp-tunnel object points to the two udp-half objects that implement each end of the tunnel, and these in turn point to the virtual machines that host them.

All objects have a soft link called `methods` that points to a directory of scripts for manipulating that object. This allows the control tree to behave as a simple object oriented language with object persistence. For example, `vms/testvm/methods/do-summary` will give the current status of the virtual machine called testvm. `vms/testvm/atnode/methods/do-summary` will give the status of whatever node it on. The SBUML commands sbuml–remotecmd and sbumlguestexec can recognize control tree objects and use them as shortcut aliases for remote command targets.

The main methods for each object are `do-bringup` and `do-takedown`, which instantiate and destroy the actual remote object. Each `do-bringup` method has the same basic structure for its implementation:

```
- lock object
- check status of the object with do-summary
- if "object is not up" then
   - bring up all objects it depends on
      (e.g. bring up VM before any tunnels)
   - take down any part of the object still existing
   - make a fresh attempt to bring up the object
- recheck status of the object with do-summary
- release lock
```

When deploying complex distributed systems, networks and remote hosts outside of the user's control often cause partial failure. The structure of the `do-bringup` method has proved to be effective for gracefully allowing successful parts of the system to keep running while making attempts to fix problematic parts. The locks allow the system to bring objects up in parallel as much as possible.

As an optimization, if the do-summary method determines an object is up, further calls within the following 120 seconds will assume the object is still up, which saves the time necessary to do the checks. The actual steps to bring up an object varies according to the type of object and usually requires a number of remote commands. The objects for nodes know about some types of hosts, such as PlanetLab nodes, and will take steps to ensure that the required software, including SBUML itself, is installed.

3. AN EXAMPLE DISTRIBUTED SYSTEM

Being able to separate physical host details from the implementation of a distributed application leads to a number of interrelated benefits during both development and use. Mobitopolo allows us to experience these with real systems today. For a simple but sufficiently-complex example application, consider a hypothetical administrator who wishes to to explore the merits of splitting a file server into globally distributed components. The problem the administrator wants to solve is that the file server was not designed for connections with high latency. (For a plausible story line, assume the administrator must appease a stubborn supervisor who is traveling internationally, yet wants file server access to be the same as when working at the home office.) The administrator believes that virtualized infrastructure might offer a solution.

High international latency is caused by distance, so at first the only solution seems to be the use of virtual infrastructure to move the whole file server closer to the user. However, for a server with large contents, this simple solution is likely to become too unwieldy or impractical. This realization inspires another idea, which is to move only the protocol part of the file server, leaving the server's contents mostly unmoved.

3.1 Design of Example System

Starting with this rough idea, the administrator's next task is to find a workable way to implement it. Because Mobitopolo separates internal implementation issues from most physical deployment issues, it enables effective *divide and conquer*, which is one of the most time-tested strategies for simplifying software development. The administrator can temporarily ignore many issues about whatever physical machines will be used, such as performance, routing, IP addresses, free ports, firewalls, location, ownership, kernel features, account permissions, etc. The administrator task is simplified to that of creating an implementation that will work for locally connected Linux machines that can commu-

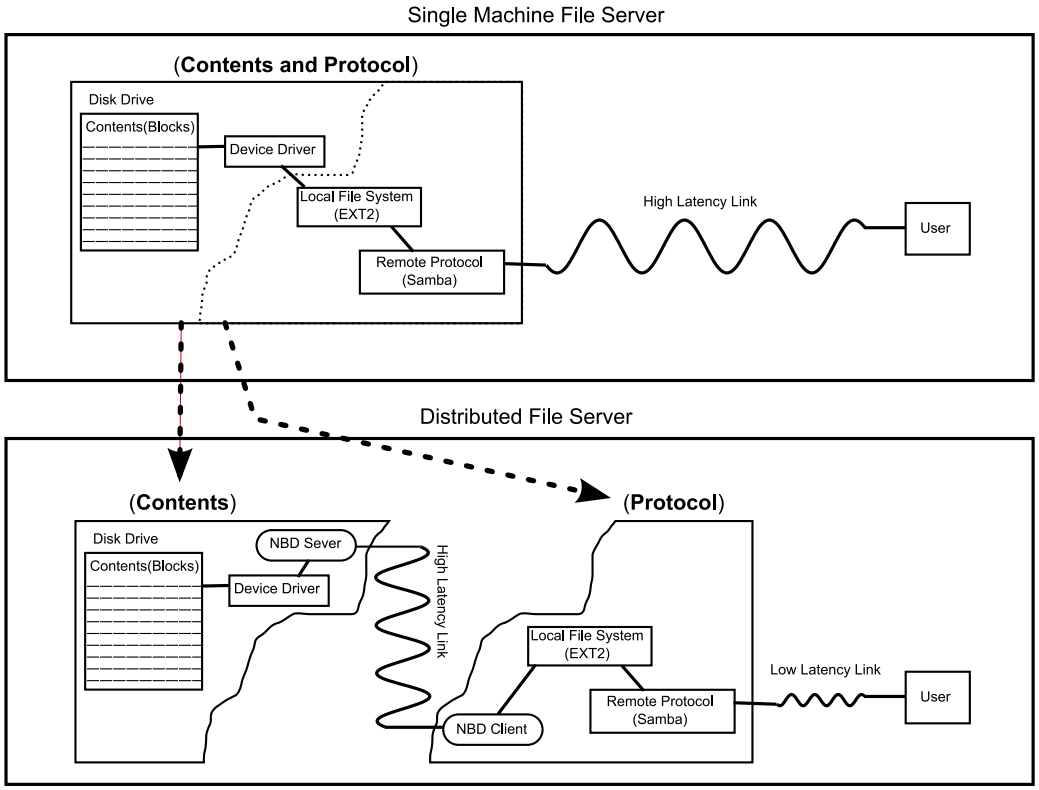

Figure 2: Basic idea for splitting a file server to move the protocol component closer to the user.

nicate freely over Ethernet, though keeping in mind that the connections will be slower and less reliable once the system is deployed globally.

At a high-level, designing for Mobitopolo means splitting the system into components that can function in separate Linux containers. The key challenge here is how to separate the protocol part of the file server from its contents. The top of Figure 2 shows one depiction of how a normal one-machine file server is set up with contents, device driver, local file system, and remote file system protocol all running in one machine. The three links between these components suggest plausible places to implement the separation. The administrator has full root permissions in the Linux machines and can therefore make use of any kernel functionality or install any software that makes implementation easier.

The link between the device driver and the local file system is particularly promising, because the Linux kernel provides a special "Network Block Device" (NBD) driver that can access a block device on a different Linux machine. Therefore, the administrator can achieve the key design goal simply by using a block device in one VM to store the contents and accessing them from a separate VM using NBD. This solution is illustrated in the bottom of Figure 2. No custom software needs to be written, and configuring NBD and Samba is as simple as on a local network.

Figure 3 shows the administrator's complete design at the topology level, which includes the contents VM and the protocol VM shown as the larger icons. It also includes several addition VMs that have been added to address design issues affected by factors external to the distributed system. The first issue is the reliability of Internet connections. Although there is no 100% solution, the administrator decides that some attempt should be made to make the NBD connection more reliable, so a backup router has been added, which appears as the smaller icon closest to the bottom of the figure. If this router can be placed carefully on the physical Internet topology, the system can potentially route around major Internet failures more quickly than the Internet's internal routing. XORP [6], an open source Linux-based routing solution, can be installed inside the all the VMs as a standard way to detect link failure and change routing.

A second issue is providing a way for users of the system to connect to the private network. As explained in Section 2.1, the preferred solution is to use a TAP device from the host. Unfortunately, this solution puts extra restrictions on Mobitopolo. The administrator does not want these restrictions to affect placement of the protocol VM, and so decides to allocate a separate VM for the TAP bridge and to duplicate it for more flexibility and redundancy, and because users at the home office will also want access to the system, adds a third TAP VM for their use. These three TAP VMs appear in Figure 3 as medium sized icons. An extra backup router VM appears in the design so that the remote TAP VMs have a chance for more reliable Internet connectivity, should a suitable location for it be found. The extra VMs do not necessarily have to be deployed, so adding them to the design incurs no cost.

3.2 Implementation of Example System

The first part of implementation is to create the VMs and tunnels that make up the topology itself, which is done through tools provided by Mobitopolo. The other part of

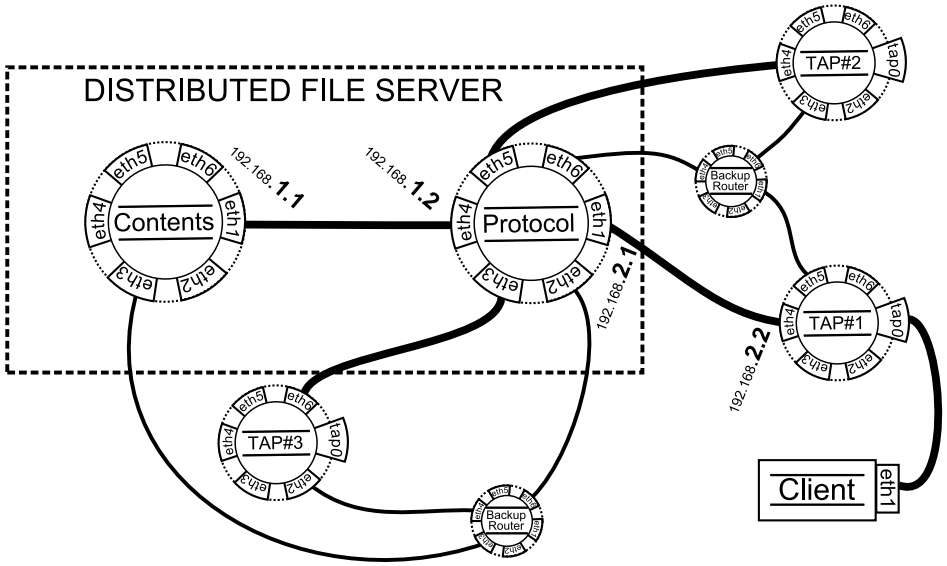

Figure 3: The administrator's final design with 7 virtual machines connected by 10 tunnels. Machines with TAP devices must be placed on hosts that allow some root-level privileges.

implementation is to set up each VM, which is done using familiar, standard Linux tools for installing and setting up software.

When moving from design to implementation, the benefits of divide and conquer still help simplify the focus to the distributed system's internal aspects. Because the physical locations of the VMs do not matter at this point, they can be placed on a normal local machine or wherever is most convenient.

Topology creation and VM setup can be done incrementally and interactively. Changes to the topology are fast because only changes to data structures in the control tree are needed. Actual instantiation of the VMs and tunnels can wait until the administrator is ready to start internal setup of some or all of the VMs. All necessary information is stored in the control tree, so instantiation can proceed automatically.

When virtual machines are added to the topology, an SBUML snapshot must be chosen for initializing it. Snapshots provide easy reuse of VMs that have proven to work on past implementations. For example, when preparing an implementation for testing the design in Figure 3, we at first specified that all VMs be initialized from a snapshot of a VM we had been using to test XORP.

For adding the tunnels, the two VMs that mark its endpoints are specified. Tap devices are added by giving the name of the VM that it will be attached to. After machines, tunnels, and TAPs have been added, all can be brought up and instantiated by issuing one Mobitopolo command.

Once all VMs are running, internal configuration of each VM can begin. The contents and protocol VMs require the most attention. Tools for NBD are copied to the contents and protocol VMs. NBD server is started on the contents VM, and the NBD client is is started on the protocol VM. The file server is set up on the protocol VM, and the NBD device is mounted to a directory exported by the file server. Because configuration is for the virtual network, IP addresses are known and ports can be freely chosen.

For the other five VMs, no extra setup is required, because they already have XORP installed, which enables automatic discovery and setup of routing for all VMs.

Normal Linux tools and techniques can be used for testing. Again, the focus can stay on internal aspects of the distributed system, such as whether NBD works as expected. Once the system is judged to be correctly set up, a single Mobitopolo `save-vms` command can save synchronized snapshots of all seven VMs. The save-vms command also updates each VM object in the control tree with the new corresponding snapshot specification so that when the VMs are taken down and brought up again (perhaps on different infrastructure), all VMs will have software installed and be preconfigured exactly as recorded by the snapshots.

3.3 Testing Performance on the Internet

Although the simple design, reuse of standard components, and local testing make it likely the system will function correctly on the Internet, the performance is not easy for the administrator to predict. On one hand, running the file server's protocol (SMB) over a lower latency link should improve performance. On the other hand, running NBD's protocol over the higher latency link will decrease performance. The hope is that overall performance will be better, because the NBD protocol does not have to handle sharing between multiple clients and can therefore be a less chatty protocol. It should help that NBD will be under the control of a file system that was designed to be efficient when faced with the high latency of disk drives. But to really know and generate convincing evidence for himself and others, the administrator values the hard data that will come from actual deployment.

Now that the topology has been specified with Mobitopolo and all machine configuration has been saved into snapshots, testing is easy. Deployment to remote hosts can be automatic. Utilities exist to send commands to remote hosts and VMs, so centralized automatic test scripts can be written as easily as for VMs on local machines, which is valuable

```
#!/bin/bash
protocolNode="$1"          # The script takes one parameter
oDir=./archive/current/    # The place in control tree for saving statistics
# (1) Remove any running vms:
./control/do-takedown vms/*
# (2) Backup control tree state:
./control/control-tree-backup
# (3) Move protocol vm to given target:
./control/control-tree-move-vm ./vms/protocol "$protocolNode"
# (4) Bring up the two main tunnels, which will bring up the needed three VMs:
./control/do-bringup -bp ./udp-tunnels/192.168.1.0/ ./udp-tunnels/192.168.5.0/
# (5) Mount file share inside backup router VM:
sbumlguestexec ./vms/backup-router-1 mkdir mntdir
sbumlguestexec ./vms/backup-router-1 mount -t smbfs -o \
    username=sbuml,password=sbuml,uid=root //192.168.5.1/sbuml mntdir
# (6) Record performance of tunnels:
./control/control-tree-save-stats -tunnel-performance ./udp-tunnels/192.168.1.0/
./control/control-tree-save-stats -tunnel-performance ./udp-tunnels/192.168.5.0/
# (7) Time copying of 18MBytes of image files:
sbumlguestexec ./vms/backup-router-1 \
    time md5sum mntdir/mnt100m/photos/dandelion/*.jpg  >$oDir/time-results.txt 2>&1
```

Listing 1: Script to deploy three virtual machines, varying the location of the protocol machine specified by a parameter. Network and file server performance are measured.

when the tests are likely to be repeated. This is true for the distributed file server example, because the goal is to compare performance with multiple tests corresponding to various physical deployments.

Assume the supervisor wants to access the file server from Florida and that the home office is in Tokyo. Also assume that PlanetLab provides the only nodes in Florida that the administrator can access, which presents a problem because they do not allow general purpose TAP devices to be installed. At this stage of testing, the administrator decides it is acceptable to simulate client access inside the virtual network using one of the backup router VMs. Having a full Linux kernel, it has all the software needed to mount the Samba share and do performance tests. So for an initial test, only three VMs need to be deployed. The contents VM needs to be deployed in Tokyo, the backup router VM in Florida, and the protocol VM in various locations to test the location's affect on performance. Now to completely automate the test, all that needs to be written is a short script, such as in Listing 1, to set the target node for the protocol VM, bring up the three VMs, and bring up the two tunnels connecting them. Then the remote commands facilities (described in Section 2.3) can be used to send commands to measure performance. Performance is checked by measuring the time necessary for the backup router VM in Florida to compute the md5sum of an 18MB set of files in the file server, which forces the data to be copied across both the NBD and SMB protocols. The output of md5sum is later used to help confirm that the time results are valid.

The penultimate column of Table 1 shows the file copying throughput achieved when placing the protocol VM in 24 different locations. The top line shows that throughput for Tokyo is about 120kbps. The next line shows the ideal case of having the protocol VM on the same local network as the client, which produced 790kbps. The next group of lines are for locations in Eastern North America and average about 400kbps. Locations in Western North America and then Europe follow with throughputs of about 300kbps and 130kbps respectively. (Other table columns will be discussed in Section 4.)

The administrator now has hard evidence that sometimes it is possible for the location of the protocol VM to speed throughput by a factor of six. With this, he can move on to interpret the results in light of his overall goal and alternative solutions. If more tests are desired, the exact same system can be redeployed as many times as necessary.

For example, if colleagues wonder what the packet counts are on the links during file copy, the exact same experiment can be run again with the additional data collection. A quick rerun on a local machine shows that the protocol node receives about the same number of packets from NBD as it transmits to SMB (13222 vs. 13710). Going in the other direction, the SMB link receives 9057 packets, which is much more than the 2410 packets transmitted to the NBD link. In one direction at least, NBD indeed seems to be a less chatty protocol than SMB.

The copy operation took about 19 minutes when the protocol VM was in Tokyo versus about 3 minutes for the Florida location. If each of the extra 6647 SMB packets resulted in a separate round trip delay of about 200ms, up to 22 minutes of delay could result. Therefore the extra packets give a plausible start towards explaining the 16 minute performance difference. More investigation (and perhaps more experiments) are necessary to know for certain.

3.4 Actual Use of Example System

After the distributed system has been tested in local and remote deployments, it is also possible to use Mobitopolo to deploy the exact same system for actual use. At this point, however, the overhead of a user-mode solution like UML/SBUML must be considered. For cases when the overhead is judged to be too large, the system can be transferred to faster environments. Mobitopolo helps keep this option open by supporting standard interfaces that also exist on other virtualization solutions and non-virtualized Linux hosts. Therefore rewriting of software would probably be

Table 1: Comparisons of various deployments. (Th=throughput, La=Latency, H=Host, VM=Virtual Machine)

connection to contents in Tokyo				Host of Protocol VM	connection to client in Florida					(cached)
H-Th (Mbps)	VM-Th (Mbps)	H-La (ms)	VM-La (ms)		H-Th (Mbps)	VM-Th (Mbps)	H-La (ms)	VM-La (ms)	File-Th (Mbps)	File-Th (Mbps)
				collocated with contents:						
5791	117.96	.06	.71	node1-net0.koganei.corelab.jp	2.29	2.13	203.00	204.00	**.12**	.12
				collocated with client:						
1.93	1.18	203.00	204.00	planetlab2.acis.ufl.edu	6367	117.17	.02	.32	**.79**	19.00
				in Eastern North America:						
2.71	1.91	177.00	178.00	planetlab5.csres.utexas.edu	8.55	7.90	28.70	29.00	.49	.91
6.23	2.32	185.00	180.00	ec2-67...amazonaws.com	8.03	7.82	28.90	29.35	.43	.83
3.12	2.44	155.00	156.00	planetlab2.utdallas.edu	8.06	7.37	43.70	44.95	.36	.60
2.47	2.15	188.00	188.00	planetlab2.isi.jhu.edu	8.78	7.74	48.20	48.55	.32	.54
2.50	1.96	188.50	188.50	planetlab4.cnds.jhu.edu	8.55	7.26	48.40	49.10	.32	.53
2.38	1.63	188.00	189.00	planetlab3.cnds.jhu.edu	8.71	7.48	49.30	49.55	.31	.53
2.41	2.24	186.00	193.00	planet1.pitts...intel-research.net	1.12	0.61	54.90	55.30	.30	.46
				in Western North America:						
4.21	3.81	116.00	185.50	planlab2.cs.caltech.edu	8.64	7.28	56.40	56.70	.33	.48
4.08	2.83	120.00	121.00	planetlab-2.calpoly-netlab.net	7.34	6.21	60.30	60.65	.32	.45
3.40	3.16	140.00	141.00	planetlab7.flux.utah.edu	7.00	6.73	64.65	65.00	.31	.42
3.16	2.95	140.00	141.00	planetlab6.flux.utah.edu	6.67	5.99	64.70	65.05	.27	.37
0.79	0.67	243.00	244.00	planetlab4.postel.org	5.48	5.12	83.40	83.80	.21	.30
				in Europe:						
1.62	1.14	284.00	292.00	planetlab4.lublin.rd.tp.pl	3.07	2.55	155.00	156.00	.14	.16
1.63	1.03	289.00	290.00	plebt2.essex.ac.uk	2.90	2.76	163.00	163.00	.14	.16
1.55	1.34	294.00	296.00	planetlab-node1.it-sudparis.eu	2.70	2.65	171.00	171.00	.13	.15
1.50	1.41	297.00	298.00	node1pl.p...telecom-lille1.eu	2.73	2.59	173.50	174.00	.13	.15
1.50	0.49	294.00	295.00	plane-lab-pb2.uni-paderborn.de	2.66	2.34	176.00	177.00	.13	.15
1.40	1.31	309.00	310.00	planetlab2.it.uc3m.es	2.45	0.99	191.50	192.00	.12	.13
1.36	1.01	323.00	324.00	planetlab3.upc.es	2.27	2.15	205.00	206.00	.11	.12
0.28	0.27	304.00	301.00	planetlab1.mwrl.net	0.24	0.29	135.00	136.50	.07	.08
				elsewhere:						
10.20	7.95	.12	1.07	planetlab1.koganei.wide.ad.jp	2.09	2.15	203.00	203.00	.12	.13
0.57	0.34	333.00	334.00	planetlab1.tau.ac.il	1.97	1.89	216.00	216.00	.10	.11

unnecessary, although some reconfiguration and restarting of software would be required.

For cases when performance is acceptable, the flexibility of Mobitopolo offers many benefits. First, by divide and conquer, it is now possible to concentrate on just the remaining deployment issues. Beyond divide and conquer, flexible deployment can help in at least three other ways.

One is to make repeated deployments effortless so that newly discovered and acquired virtual infrastructure can be confirmed to perform as promised. A second way is to allow deployment as late as possible. Everything internally has been started up and debugged, so deployment only requires copying VM images, starting the VMs, and configuring the outside interfaces of the UDP tunnels, all of which can be done automatically. When the cost of virtual infrastructure is an issue, just-in-time deployment can save money. When performance is an issue, just-in-time deployment can allow time for more deployment options to be compared, from which the best can be chosen at the last minute.

A third way that the flexibility can be used is for *redeployment* in order to make a system better meet changing user needs. A typical example common in data centers today is to adjust trade-offs between energy consumption and performance by using live migration to consolidate virtual machines onto fewer physical servers when demand for the system is low. Other researchers have proposed this idea for routing components [14]. Mobitopolo generalizes this idea to wide area live migration and to distributed systems with legacy software that was not originally designed with this capability in mind.

Changing user needs can go beyond the performance/energy trade-off. For our file server example, assume users at the home office are upset that the sever will be moved around the world and make their access *slower*. One compromise solution could be for the server to be in Tokyo during Japan's daylight hours and migrated to North America after the home office closes each day. A test of this idea is shown in Figure 4, which compares the file copy throughput experienced by simulated users in Tokyo and North America as the protocol VM is first migrated from a CoreLab [10] node in Tokyo, to an Amazon EC2 [1] node in Virginia, and finally to a PlanetLab [3] node in Florida. Performance for the Tokyo users drops from about 150Kbps to 3Kbps, so such a transfer should wait until after users in Tokyo have stopped working for the evening. The supervisor in Florida enjoys an increase from 3Kbps to about 25Kbps.

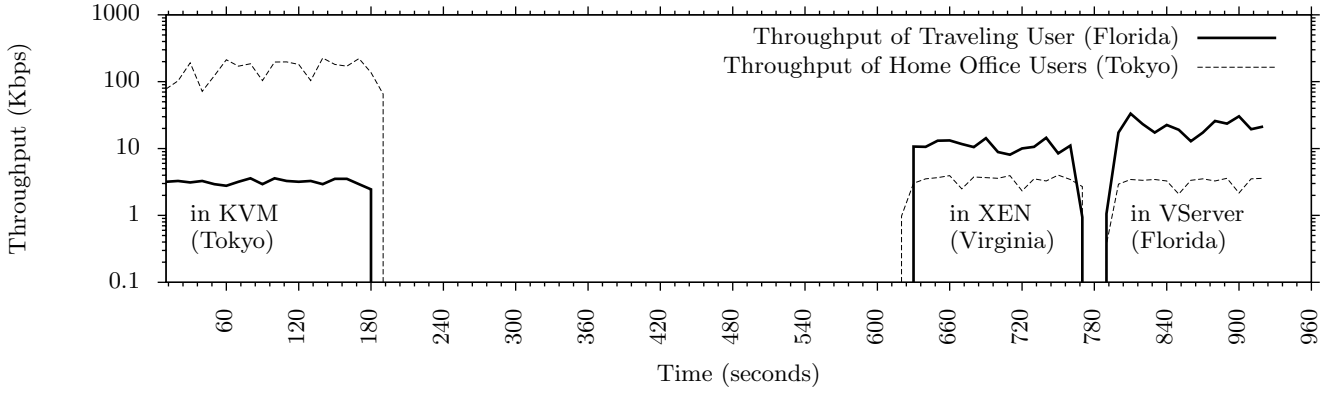

Figure 4: Changes in file copy throughput as the protocol VM is migrated from Tokyo, to Virginia, and then to Florida across three different virtualization technologies.

The graph represents a particularly challenging case of migration, one with high simulated load from Tokyo users, which kept dirtying the VM's RAM and making precopying ineffective. Therefore, the server ended up being down for the 7.5 minutes required to copy the protocol VM's 200MB state over the Internet from Tokyo to Virginia. Remote file copy operations experienced I/O errors during this time. Throttling the Tokyo users' bandwidth during migration could lessen this problem. However, the protocol VM is less busy in Virginia because of lower network bandwidth, and so the migration to Florida happened with about 26 seconds of downtime, which is fast enough for the file operations to proceed without error. This gives our hypothetical administrator hard evidence about what is demonstratively possible, from which alternatives and trade-offs can be more realistically judged.

4. DISCUSSION

In terms of ease of deployment, we believe Mobitopolo has been a success. The most difficult part of implementing and deploying a distributed system such as in the previous section is the effort of configuring software in each VM. Fortunately, even with many deployments, the software configuration only had to be done once.

Distribution to PlanetLab and CoreLab nodes is fully automatic, including requests to PlanetLab's API to add nodes to slices, wait the 20 or so minutes for them to appear, and then install SBUML and supporting software. Once the node is initialized, snapshots for the VM images can be deployed quickly because their sizes are relatively small.

Each snapshot is of a VM with 150MB of memory and a 16GB file system, of which 13GB is filled with software. In the file system are directories with compilations of the XORP and CLICK software suites, which together take up more than 1.3GB of disk space. So many gigabytes of state would make deployment across the Internet slow and would possibly limit which physical infrastructure would be able to host the machines. Fortunately, all that is necessary is to clone the VM, and all the VM's unused file system contents can be left on standby on an HTTP server. Disk information is copied on reference, so it stays with the VM once accessed. Therefore, by cloning and then exercising the software, a good approximation of the system's working set of disk storage can be collected into the VM and its snapshots [12]. This reduces the backup router VMs and TAP VMs to about 23MB. The protocol VM snapshot is 26MB. The contents VM, which will be deployed locally, is larger (1.2GB) because it is loaded with test data. For the smaller snapshots, going from 13GB to 26MB or less reduces the state that must travel over the Internet by a factor greater than 500.

Deployment of the 26MB protocol VM to a PlanetLab node 25km away from our Tokyo lab takes about 52 seconds. Once the snapshot has been distributed, the machine can be brought up again in 30 seconds. The same VM can be deployed freshly to Florida in 125 seconds, and once there brought up again in 20 seconds. After the three VMs used to create the table in Table 1 are brought up, the two tunnels can be brought up in about 16 seconds. We have some optimizations planned to make these times faster, however so far these deployment times have been satisfactory for our current use.

Downtime for migration, however, is more critical. For that we implemented a special case tunnel configuration optimization that skips some time consuming tests, which makes it possible, when enough bandwidth exists between source and destination, to do migration and keep network downtime under 10 seconds. Reducing this time further and finding graceful ways to deal with difficult cases such as illustrated in Figure 4 is a priority for our research.

The new UDP tunnel device for SBUML was successful in making changes in physical deployment transparent to software inside. Active TCP connections are able to survive saving to snapshots and redeployment as well as migrations. Performance can be seen in the "Th" and "La" columns of Table 1. The tunnel added about .35ms of the latency of the host. For Internet connections, the tunnel delivered measured throughput between 33% to approximately 100% of measured host throughput. Over half of these tunnel connections delivered greater than 85% of host throughput.

For connections between virtual machines on the same host, throughput maxed out at about 117Mbps, so the current solution clearly has significant overhead. To make the system more useful for experiments on fast local networks, we plan to explore how much of this overhead can be removed. Previous research [9] has demonstrated techniques that improve UML's throughput by a factor of three, and we expect some of these techniques will improve Mobitopolo.

5. RELATED WORK

Other projects have taken advantage of UML's user-mode characteristics for networking research. The VINI research project [2] used UML to host XORP routing software inside PlanetLab. It used another of UML's many networking interfaces called uml_switch to create a number of Ethernet devices. Uml_switch was integrated with Click network forwarding software to provide a data plane that could do forwarding independently of UML. Although this architecture is probably faster for network forwarding applications, it is probably slower for connections to software hosted inside UML, because of extra hops through the Click and uml_switch processes. VIOLIN [7] also connects UML virtual machines hosted on PlanetLab with UDP tunnels. However it uses intermediary virtual switches and virtual router components between the virtual machines rather than the direct connections used in Mobitopolo. Another distinction between Mobitopolo and these two systems is that Mobitopolo uses SBUML's snapshot extensions to UML and can therefore distribute small cloned prebooted VMs, as well as support migration. A later version of VIOLIN [8] integrates snapshots, however the virtualization was changed to Xen, and thus lost the portability of UML-based solutions.

6. FUTURE WORK AND CONCLUSIONS

In Mobitopolo, we provide standard Linux functionality and networking interfaces for the construction of networking experiments and general purpose distributed applications. We make it possible for these to run as user-mode software on standard Linux hosts, and therefore users can take advantage of the unstandardized mix of commercial and research oriented virtualized infrastructure becoming available globally on the public Internet and private research networks. Physical topology is transparent to software running inside Mobitopolo, which makes it easier to change physical deployment or deploy multiple times, because no changes or reconfiguration of the software is necessary. This makes its snapshot functionality more powerful, because all configuration can be done in advance and saved into snapshots, even if physical deployment details are yet unknown. Live migration across WAN is supported, so that components of distributed systems can be moved about the world with minimal disruption to the systems' continuous operations. We know of no other user-mode system that offers completely automatic deployment of preconfigured distributed Linux topologies and live migration of components across wide-area Internet connections, all transparent to software running inside the Linux components.

7. ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for thoughtful suggestions that improved the paper and to acknowledge the role of the Japan Science and Technology Agency (JST) and the Information-Technology Promotion Agency (IPA) in providing funding for development of the legacy SBUML features used in this work.

8. REFERENCES

[1] Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2.

[2] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In VINI veritas: realistic and controlled network experimentation. In *In Proc. of SIGCOMM*, pages 3–14, 2006.

[3] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, 2003.

[4] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proc. 2nd USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2005)*, 2005.

[5] J. Dike. *User Mode Linux*. Prentice Hall Ptr, 1st edition, April 2006.

[6] M. Handley, E. Kohler, A. Ghosh, O. Hodson, and P. Radoslavov. Designing extensible IP router software. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 189–202, Berkeley, CA, USA, 2005. USENIX Association.

[7] X. Jiang and D. Xu. Violin: Virtual internetworking on overlay infrastructure. In *Proc. of the 2nd Intl. Symp. on Parallel and Distributed Processing and Applications*. Springer, 2004.

[8] A. Kangarlou, D. Xu, P. Ruth, and P. Eugster. Taking snapshots of virtual networked environments. In *Proceedings of the 3rd international workshop on Virtualization technology in distributed computing*. ACM New York, NY, USA, 2007.

[9] Y. Koh, C. Pu, S. Bhatia, and C. Consel. Efficient packet processing in user-level OSes: A study of UML. *Local Computer Networks, Annual IEEE Conference on*, 0:63–70, 2006.

[10] A. Nakao, R. Ozaki, and Y. Nishida. CoreLab: An emerging network testbed employing hosted virtual machine monitor. In *Proc. of ROADS'08*, 2008.

[11] R. Potter. One-click distribution of preconfigured Linux runtime state. In *Virtual Machine Research and Technology Symposium*, 2004.

[12] R. Potter and K. Kato. SBUML: Multiple snapshots of Linux runtime state. *JSSST Computer Software (to appear)*, 2009.

[13] O. Sato, R. Potter, M. Yamamoto, and M. Hagiya. UML scrapbook and realization of snapshot programming environment. In K. Futatsugi, F. Mizoguchi, and N. Yonezaki, editors, *ISSS*, volume 3233 of *Lecture Notes in Computer Science*, pages 281–295. Springer, 2003.

[14] Y. Wang, E. Keller, B. Biskeborn, J. E. van der Merwe, and J. Rexford. Virtual routers on the move: live router migration as a network-management primitive. In *SIGCOMM*, pages 231–242. ACM, 2008.