Peer-to-Peer Virtualized Services

David Bailey and Kevin Vella
University of Malta
Msida, Malta
Email: david@davidbailey.info, kevin.vella@um.edu.mt

***Abstract*—This paper describes the design and operation of a peer-to-peer framework for providing, locating and consuming distributed services that are encapsulated within virtual machines. We believe that the decentralized nature of peer-to-peer networks acting in tandem with techniques such as live virtual machine migration and replication facilitate scalable and on-demand provision of services. Furthermore, the use of virtual machines eases the deployment of a wide range of legacy systems that may subsequently be exposed through the framework. To illustrate the feasibility of running distributed services within virtual machines, several computational benchmarks are executed on a compute cluster running our framework, and their performance characteristics are evaluated. While I/O-intensive benchmarks suffer a penalty due to virtualization-related limitations in the prevailing I/O architecture, the performance of processor-bound benchmarks is virtually unaffected. Thus, the combination of peer-to-peer technology and virtualization merits serious consideration as a scalable and ubiquitous basis for distributed services. A view of some challenges and opportunities that emerge in the design of such frameworks is also offered.**

***Keywords*-Virtualization; distributed systems; peer-to-peer computing; service-oriented computing; cloud computing.**

I. Introduction

This paper describes a framework that enables the dynamic provision, discovery, consumption and management of software services hosted within distributed virtual machines. The framework, Xenos [1][2], uses a decentralised peer-to-peer overlay network for advertising and locating service instances and factories. It also leverages techniques such as live virtual machine migration and replication to enhance operational agility and ease of management, and to lay the foundations for deploying fault-tolerant services. The primary objective is to shift the focus away from managing physical or virtual machines to managing software services.

In recent years, data centre operations have experienced a shift in focus away from managing physical machines to managing virtual machines. Renewed exploration of this well-trodden path is arguably driven by virtualization's mantra of enhanced operational agility and ease of management, increased resource utilisation, improved fault isolation and reliability, and simplified integration of multiple legacy systems. Virtualization is also permeating the cluster and grid computing communities, and we believe it will feature at the heart of future desktop computers and possibly even advance a rethink of general purpose operating system architecture.

The performance hit commonly associated with virtualization has been partly addressed on commodity computers by recent modifications to the x86 architecture [3], with both AMD and Intel announcing specifications for integrating IOMMUs (Input/Output Memory Management Units) with upcoming architectures. While this largely resolves the issue of computational slow-down and simplifies hypervisor design, virtualized I/O performance will remain mostly below par until I/O devices are capable of holding direct and concurrent conversations with several virtual machines on the same host. This generally requires I/O devices to be aware of each individual virtual machine's memory regions and demultiplex transfers accordingly. We assume that this capability or a similar enabler will be commonplace in coming years, and that the commoditization of larger multi-core processors will reduce the frequency of expensive world-switches as different virtual machines are mapped to cores over space rather than time.

The paper is organized as follows. Section II provides an overview of related work, and Section III briefly describes the key topics that underpin this research. Section IV details the proposed framework and the implemented prototype, while Section V presents an evaluation of the framework. Finally, Section VI exposes a number of issues for future investigation, and an overview of this work's contribution can be found in Section VII.

II. Related Work

The ideas presented here are influenced by the Xenoservers project [4], initiated by the creators of the Xen hypervisor. Xenoservers was designed to "build a public infrastructure for wide-area distributed computing" by hosting services within Xen virtual machines. The authors argue that current solutions for providing access to online resources, such as data storage space or an application-specific server, is not flexible enough and is often based on a timeline of months or years, which might not always accommodate certain users. The Xenoserver infrastructure would allow for users to purchase temporary resources for immediate use and for a small time period, for instance a group of players wanting to host a game server for a few hours or even minutes. A global infrastructure can also

aid in exploiting locality by running code on a network location that is close to the entities that it uses, such as data and services, to improve performance. In order to allow untrusted sources to submit their own applications, execution environments need to be isolated; the authors propose to use the Xen hypervisor [5] to provide these isolated and secure environments in the form of virtual machines, which also allows for a high degree of flexibility as users have a wide array of operating system and application environments to choose from. Xenosearch [6] locates Xenoservers using the Pastry peer-to-peer overlay network. A Xenoservers implementation is not generally available, hence our decision to build and conduct experiments with Xenos.

WOW [7] is a "distributed system that combines virtual machine, overlay networking and peer-to-peer techniques to create scalable wide-area networks of virtual workstations for high-throughput computing". Applications and services in the system are provided in virtual machines, which must also contain a virtual network component that is used to register the machine on a peer-to-peer overlay network when the machine boots up. This peer-to-peer overlay network is used to create virtual links between the virtual machines, which are self-organizing and maintain IP connectivity between machines even if a virtual machine migrates across the network. The authors do not provide a mechanism which allows for searching of other services registered on the peer-to-peer network; this is where our approach differs in that we intend to use a peer-to-peer overlay network to advertise the services running within the virtual machines rather than to set up a virtual network to enable communication between virtual machines.

SP2A [8] is a service-oriented peer-to-peer architecture which enables peer-to-peer resource sharing in grid environments, but is not concerned with the uses of virtualization in distributed computing architectures, which is one of our main interests. Several publications have focused on the use of peer-to-peer overlay networks to implement distributed resource indexing and discovery schemes in grids [9][10][11]. Wadge [12] investigates the use of peer groups to provide services in a grid, as well as transferring service code from one node to another for increased fault-tolerance and availability.

The dynamic provisioning of services is a relatively young area of research, and commercial products such as Amazon Elastic Compute Cloud (EC2) have only appeared in the past few years. Virtualization and hardware advancements have had a major impact on the structure of these datacenters, which typically rely on tried-and-tested setups and favour the traditional client-server approach to locating and consuming services. We believe that exploiting the advantages of peer-to-peer networks is the next step in achieving a truly distributed, scalable and resilient services platform.

III. Background

A. Virtualization

In computing, virtualization can be broadly defined as the software abstraction of a set of resources, which enables the sharing of these resources in parallel by higher-level systems. While the actual definition and mechanisms used varies depending on the type of virtualization in question, the concept always remains the same; that of efficiently, securely and transparently multiplexing a set of resources in a manner which allows for higher-level systems to use these resources and allowing them to assume that they are using the real resources instead of the abstraction provided by the mechanism. We are mostly interested in hardware-level virtualization, where the virtualization layer sits on top of the hardware and virtualizes the hardware devices, allowing multiple operating systems to execute within the virtual machine environments presented by the layer. Hardware resources such as the processor, memory and I/O are managed by the virtualization layer and shared by the executing operating systems, although the latter might have no knowledge of the underlying virtualization layer. This layer is often called a virtual machine monitor or a hypervisor.

One of the techniques used in achieving full hardware virtulization is paravirtualization, where the hypervisor provides virtual machines that are not exact copies of the underlying hardware architecture. This implies that the operating system executing in a virtual machine provided by the hypervisor is aware that it is running inside a virtualized environment, and has to issue calls to the hypervisor for certain operations. Legacy operating systems therefore need to be ported in order to run on the hypervisor. Perhaps the most successful paravirtualized hypervisor that has gained widespread use in the industry is the Xen hypervisor, on which a number of commercial solutions are based, such as Citrix XenServer, Oracle VM and Sun xVM, as well as heavily influencing the design of Microsoft's Hyper-V hypervisor. Xen supports existing application binary interfaces, meaning it can support applications written for the x86 architecture without the need for modification; and it exposes certain physical resources directly to guest operating systems, allowing for better performance. The Xen hypervisor aims at supporting legacy operating systems (with minimal porting effort) and existing applications, while leveraging the benefits of a paravirtualized approach, such as high performance and stronger isolation.

B. High-Performance Computing and Grids

Mergen *et al.* [13] argue that hypervisors offer a new opportunity for high performance computing (HPC) environments to circumvent the limitations imposed by legacy operating systems. Modern hypervisors not only support legacy systems, but they can also simultaneously execute

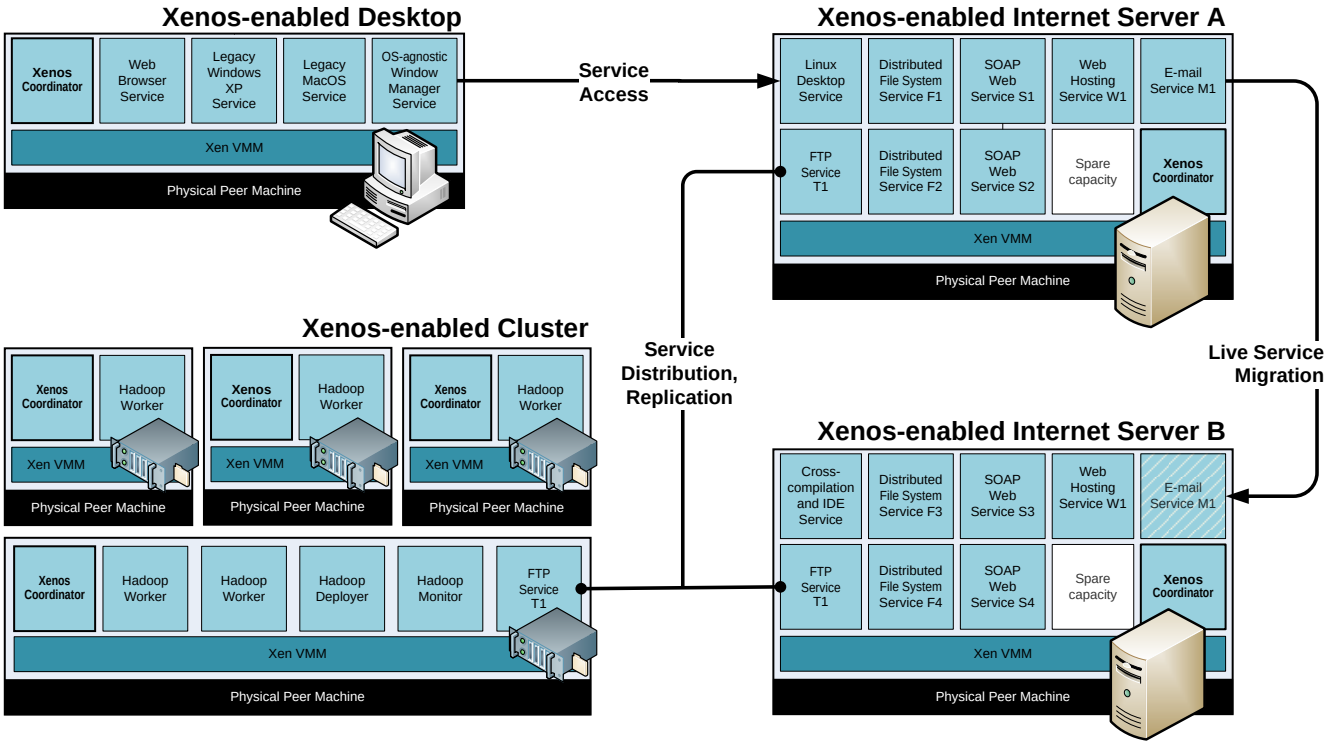

Figure 1. A selection of computing platforms running the Xenos framework and hosting several interacting services.

specialized execution environments in which HPC applications are run; this allows for legacy HPC software and other non-critical operating system services, such as file systems, to be run within the legacy operating system, while the HPC application can exploit the hypervisor directly and implement any optimization opportunities, such as the use of super-pages. These specialized execution environments are also known as *library OS*, since they typically contain only the required software stacks for the application(s) that will be executing within them. Thibault *et al.* [14] implement lightweight Xen domains based on the Mini-OS paravirtualized kernel, which is included with the Xen hypervisor as a demonstration of how to implement a basic guest kernel for Xen – it is able to use the Xen network, block and console mechanisms, supports non-preemptive threads and only one virtual memory address space. A similar approach is taken by Anderson *et al.* [15], although the focus is on security and reliability; the authors argue that partitioning critical services and applications into domains with tight restrictions improves trustworthiness. Falzon [16] implements a lightweight Xen domain to explicitly execute a thread scheduler that supports multiple processors and offers several scheduling policies. This allows for the creation and evaluation of different schedulers that have direct access to the virtualized hardware, and can for instance control the number of kernel threads mapped on a particular virtual CPU, or disable timers and interrupts in the domain.

A number of publications have focused on the use of virtualization and virtual machines in grid computing environments in response to a number of significant issues such as security, administration costs and resource control. Figueiredo *et al.* [17] present a number of tests that show overheads to be minimal under the VMware Workstation hypervisor. The authors also present an architecture for dynamically instantiated virtual machines based on a user's request, where a virtual machine is distributed across three logical entities: image servers that hold static virtual machine states, computation servers that can dynamically instantiate and execute images, and data servers which store user application data. Keahey *et al.* [18] propose a similar architecture, providing execution environments for grid users called Dynamic Virtual Environments (DVEs), as well as implementing DVEs using different technologies such as UNIX accounts, and operating-system and hardware-level virtual machines such as VServer sandboxes and VMware respectively. The different implementations were analyzed to determine their viability for use in a grid infrastructure, and while they provided sufficient in terms of applications without heavy I/O loads, the authors believe that all had shortcomings in Quality of Service (QoS) functionality, and some technologies such as VMware did not expose enough of their functionality for direct use in the grid. Santhanam *et al.* [19] experiment with different sandbox configurations, deployed using the Xen hypevisor, and concluded that jobs with heavy I/O loads take a performance hit when running inside a virtual machine sandbox, although they advocate the use of virtual machines in grid environments where applications often tolerate delays on the order of minutes,

and if the user wishes to benefit from the advantages obtained by using virtual environments.

IV. The Xenos Framework

Xenos is built on top of Xen, a virtualization platform that has gained traction as a stable and mature virtualization solution, but any hypervisor with the appropriate hooks and programming interfaces will suffice in principle, including a hypothetical ROM-based hypervisor. The JXTA framework is currently used to maintain a peer-to-peer overlay network for service advertisement, discovery and, optionally, transport. However, we feel that a more specialized or expressive latter generation peer-to-peer framework would better fit our requirements. The Hadoop map-reduce framework, described in more detail in Section V, is used as a benchmarking tool to evaluate the framework, but it is not an intrinsic part of the Xenos framework itself.

A. Physiology

Figure 1 illustrates a scenario with different hardware platforms running Xenos and a variety of services, which may be any software application that can be encapsulated within a Xen virtual machine. A commodity cluster, typically used for high-performance computing applications, offers users the ability to dynamically create computation services, such as Hadoop map-reduce nodes, while also using other services such as the Hadoop deployer and Hadoop monitor to easily deploy these services on the network, and monitor them for fault-tolerance and load-balancing. Xenos also runs on desktop machines, with the user utilizing several services such as a file system for personal data storage, and a legacy operating system service offering traditional applications. The user interface that the user interacts with is itself a virtualized service, possibly forming part of a distributed operating system made up of several services running on the Xenos framework. If, for instance, the user is transferring files between the file system and the Hadoop cluster, it would be possible for the instance of the file system containing the required files to be migrated (physically moved) to the cluster, thus improving the performance when transferring data to the cluster or retrieving results. Finally, another platform supporting Xenos is a traditional server in a datacentre, where services such as web, FTP and email servers, and web services are executed as virtual machines, and are used by clients or by other services across the Xenos cloud.

From the perspective of the user, the platform provides two major features: the ability to search for services, obtain information about them and make use of them, and the ability to control these services by creating new service instances, migrate running services, manage existing ones and monitor their use. System administrators are responsible for setting up and managing the infrastructure on which Xenos is hosted, providing services packaged in virtual machines, and configuring these services to appear on the Xenos peer-to-peer network. Optionally, users or administrators can also develop custom services that participate on the same peer-to-peer network as the other hosts and services and act as an additional feature to the platform. These services join the peer-to-peer network provided by Xenos and complement the existing features of our framework, or act as support services for a user's existing services. These can include fault-tolerance and load-balancing monitors, which trigger migration and replication of services as required, introspection services that provide useful information about domains, and management services that use JXTA groups to effectively manage a user's services.

B. Architecture

Figure 2 illustrates the architecture of a single physical machine in the framework. Each Xenos-enabled physical machine runs the Xen hypervisor using a paravirtualized Linux kernel in Domain 0, which is a privileged domain capable of controlling the guest domains (virtual machines) that will host services on the same physical machine. The Xenos Coordinator is a Java application that executes in Domain 0 whose primary function is to incorporate the physical machine into Xenos' peer-to-peer overlay network and advertise services running on that physical machine, through the JXTA library. Services running within guest domains do not normally join the overlay network directly, but are registered with the coordinator in Domain 0, which acts as a 'notice board' for all local services. Administrators configure these services through text-based configuration files that are picked up by the Coordinator on startup. It also provides utilities for controlling these domains by making use of the Xen Management API, and other utilities used by other components of the system itself or directly by administrators, such as file management routines and ID generators for quick configuration of hosts and services.

The Xenos API is an XML-RPC programming interface available for users and services to interact with, and is the primary channel through which services are discovered and managed. Users can search for services and/or hosts on the peer-to-peer network by passing in search parameters to the API, which then returns results describing the services or hosts. Services may also be controlled and monitored remotely by passing in identifiers for the services to be acted upon. Migration and replication of services can also be triggered through the API, which implements file transfer and copying features that are required for this functionality. The Xenos API also features an implementation of XML-RPC over JXTA protocols, which enables hosts on the peer-to-peer network to issue XML-RPC calls to each other without requiring a TCP/IP socket connection, but rather use the built-in socket functionality in JXTA. Service delivery itself may be accomplished without the involvement of Xenos, and is not restricted to any particular network protocol or

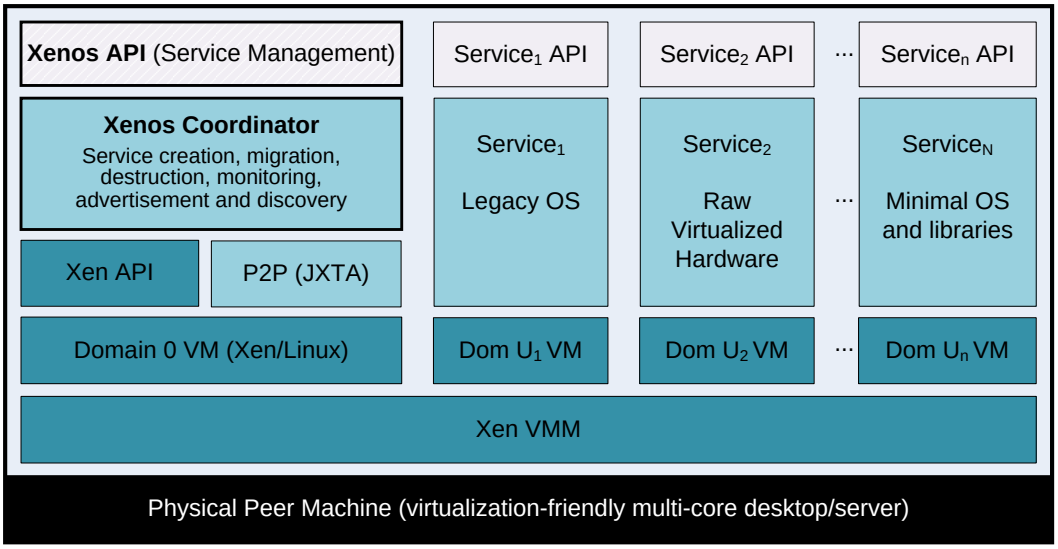

Figure 2. A Xenos-enabled physical machine.

address space. However, the direct use of network protocols beneath layer three (for example, Ethernet) would oblige communicating services to share a physical network or a physical machine.

In order to accommodate multiple instances of the same service and service migration, each service type has a template associated with it that enables the automatic configuration of new service instances and their Xen domains, as illustrated in Figure 3. When replicating a service or creating a new service instance, a new copy of the relevant template is used. Service templates will automatically replicate on other Xenos hosts as required so that service instances can be spawned anywhere on the Xenos cloud. Migration of service instances makes use of Xen's virtual machine migration mechanism with a slight modification to transfer virtual machine disk images along with the virtual machine configuration. Our current implementation inherits a Xen restriction limiting live virtual machine migration to the local area network, though this may be overcome as discussed in Section VI.

C. Design Benefits

The architectural design discussed above leads to several benefits over similar platforms. The use of a peer-to-peer overlay network enables a decentralized approach to registering and discovering services, in contrast with the centralized approach often used within existing web services platforms, such as Universal Description Discovery and Integration (UDDI). By having the Xenos API available on every host on the platform instead of a main server (and possibly some backup servers), users of the platform can make the applications that interact with the API more fault-tolerant by initially searching for a number of Xenos hosts and storing them locally as a backup list. If the host being used by the applications becomes unavailable, another host can be picked from the backup list and communication attempted with it. This can also lead to implementing a load-balancing approach to issuing API calls, so that the workload is spread over multiple hosts instead of a single one.

JXTA provides a grouping facility, where services or peers can be organized into groups that are created by the user. Our framework allows administrators to specify which groups a service should join initially; this can be used, for instance, to group together services that offer the same functionality, or to group together services that belong to the same user. Services can form part of multiple groups, and are always part of the net peer group, which is the global group maintained by JXTA. Additionally, users who build their own applications that form part of the peer-to-peer network can create new groups on the fly and assign services to them. For instance, a custom built load-balancer could create a group and automatically monitor all the services that join it; this scoping can help reduce the amount of messaging going on in the network, since the load-balancer would only need to broadcast into its created group instead of the net peer group.

Existing commercial cloud solutions, such as Amazon EC2, often provide computing instances that are fixed and feature large amounts of memory, processor resources and storage space, which are not always necessary when dealing with lightweight or specialized services. We have already discussed the benefits of running certain services inside specialized execution environments in Section III; the majority of publications that we review have used Xen as the hypervisor, as it is based on paravirtualization, which performance significantly better that other virtualization techniques and allows for modifications or development of custom operating systems for running specific services. Our platform allows administrators to create services that are based within lightweight Xen domains, and assign to them

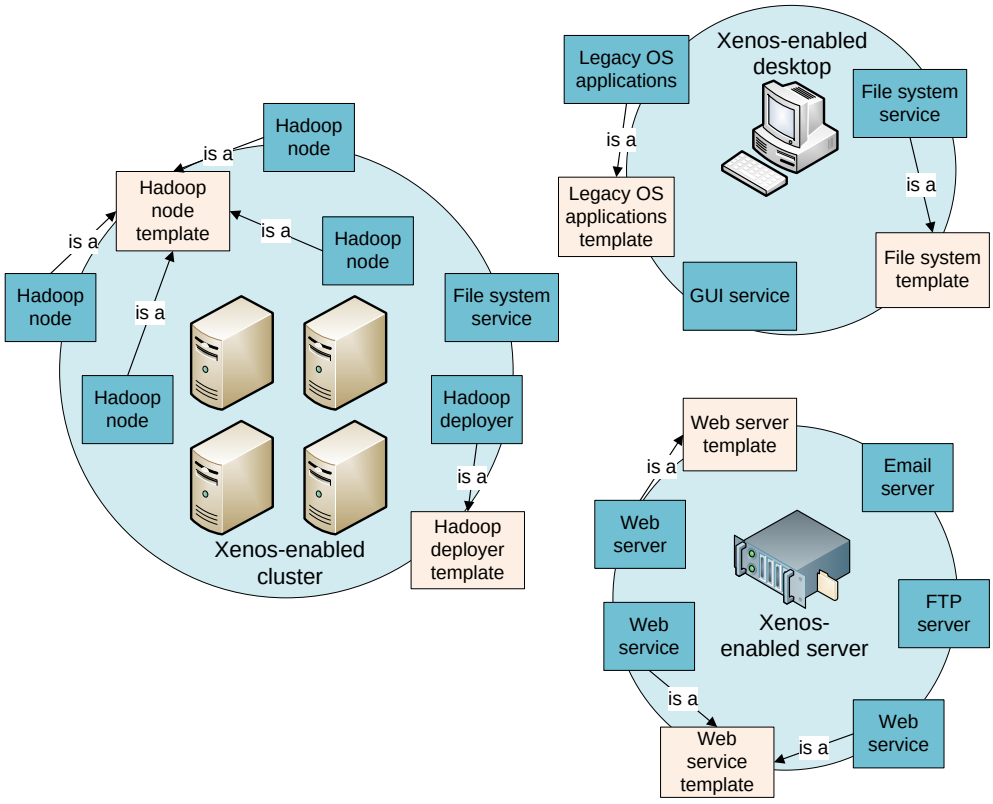

Figure 3. The relationship between Xenos templates and services.

resources as needed. This is beneficial for certain services that, for instance, would not require large amounts of storage space but benefit from multiple virtual CPUs and large amounts of memory due to their computation being mostly processor bound. Conversely, certain services might deal with data storage and processing, and thus require large amount of storage space but can do with a single virtual CPU and a small amount of memory. Although our current prototype does not have the ability for users to upload their own virtual machine images and configurations, this is trivial to add to our existing infrastructure, and would be a powerful feature that gives users even more flexibility.

V. Hadoop Case Study and Performance Analysis

A series of preliminary tests were conducted in order to assess the viability of our approach. The test cases all involve deploying multiple instances of a Hadoop map-reduce wrapper service using a separate distributed coordination service. We aim to explore three principal avenues, namely (1) the automatic and dynamic deployment of the Hadoop service to Xenos hosts and the migration of the master Hadoop node from a failing physical machine; (2) the performance of file I/O within virtual machines, which is crucial for services with large-volume data processing requirements (this is particularly relevant since Xenos requires virtual machine images to exist in files rather than as physical disk partitions); and (3) the performance of a series of virtualized Hadoop map-reduce processing jobs.

A similar evaluation of running the Hadoop map-reduce framework within a virtualized cluster is carried out by Ibrahim *et al.* [20]. They argue that a virtual machine-based Hadoop cluster can offer compensating benefits that overshadow the potential performance hit, such as improved resource utilization, reliability, ease of management and deployment, and the ability to customize the guest operating systems that host Hadoop to increase performance without disrupting the cluster's configuration.

A. Map-Reduce and Hadoop

In our experiments we used the Hadoop Distributed File System (HDFS) and MapReduce components of the Apache Hadoop framework. The map-reduce programming model, introduced by Dean *et al.* [21], is aimed at processing large amounts of data in a distributed fashion on clusters. HDFS is a distributed file system suitable for storing large data sets for applications with heavy data processing, such as typical map-reduce jobs. The Hadoop map-reduce implementation involves a master node that runs a single *JobTracker*, which accepts jobs submitted by the user, schedules the job across worker nodes by assigning map or reduce tasks to them, monitors these tasks and re-executes failed ones. Each worker (or slave) node runs a single *TaskTracker*, which is responsible for executing the tasks assigned to it by the job

tracker on the master node.

B. Deploying Hadoop Services

When setting up a computing cluster with the aim of running the Hadoop map-reduce framework, each node needs to be configured with specific settings, such as the hostname, SSH certificates and the hosts that it has access to, and the HDFS and map-reduce settings that are common throughout the cluster. When setting up a non-virtualized environment, administrators typically configure a single node, and then manually clone the hard disk to all the other nodes, resulting in an identical installation across the cluster, which would then require node-specific settings on each machine. Setting up Hadoop on a more general cluster can be done by setting up an installation on single node, and then distributing the installation to the other cluster nodes, typically via shell scripts and *rsync*. Another alternative is to use existing deployment frameworks that manage the configuration, deployment and coordination of services such as Hadoop, and do much of the work.

One of the issues that we identify with deploying any sort of service on an existing cluster environment is the potential to disrupt the configuration or execution of other services when configuring the new one. If one were to use a virtualized cluster, services could be supplied within pre-packaged virtual machines that would not interfere with other services running within their own virtual machines, since the hypervisor guarantees isolation between them. The configuration of the physical node would therefore never need to be modified when adding new services; of course, the initial setup of the virtualized cluster still needs to be done manually by administrators cloning an initial setup to all the cluster nodes, but this is inevitable. One can always set up a physical cluster with a single service in mind, which would not require frequent re-configuration, but this often leads to wasted resources that virtual machines could fully exploit if the cluster were to be virtualized.

We can identify several other benefits in using a virtualized cluster for Hadoop services. Since services would be packaged within their own virtual machine, we can easily modify the installation and configuration of the operating system running within the virtual machine to accommodate Hadoop map-reduce and the HDFS and tweak its performance, without having to modify the configuration of the operating system running on the physical node, which is Domain 0 in the case of Xen. Since the master nodes are potential single-points-of-failure both in the HDFS and Hadoop map-reduce, the master node can also be packaged inside a virtual machine, which can be checkpointed regularly, thus saving the whole virtual machine state, or migrated to another physical host if the current host is malfunctioning or needs to be shut down.

If we opt for a virtualized cluster on which to deploy Hadoop, we are still faced with the task of deploying the virtual machine containing the Hadoop map-reduce workers on the nodes of the cluster. Deployments methods similar to the ones when running a non-virtualized cluster can be used, such as setting up shell scripts to transfer virtual machine images and then issuing remote commands on the nodes to start the virtual machines. However, a more appropriate solution would be to use an existing platform, which can deploy virtual machine images to the cluster's nodes, and allows users to administer these images remotely, typically from the same node that acts as the map-reduce master. The Xenos framework that we have implemented is a perfect candidate on which to build a Hadoop deployer that allows users and administrators to provide their own Hadoop installation as a service within a domain, register this service with the Xenos coordinator, and then use the framework's replication, migration and service control features to deploy these services on the virtualized cluster. This requires a small application to be developed that oversees this task, since by itself the framework has no capabilities of deploying services automatically, but simply provides the mechanisms that allow this. We have therefore developed a Java application that uses the JXTA framework to connect to the Xenos network, and use the Xenos API to deploy the Hadoop service supplied by the administrator. Although we have tailored this application for the Hadoop map-reduce and HDFS service, we feel that it can be generalized rather easily to support any service that is registered within our system; in fact, only a small portion of the application is Hadoop-specific, as the rest simply deals with services that are defined by the user in a separate configuration file. This would provide users with a service deployer with which they can deploy their services on the Xenos framework.

C. Evaluation Platform and Results

Our evaluation platform consists of a thirteen-host commodity cluster, connected over a 1 Gigabit Ethernet connection through a D-Link DGS-1224T switch. Each physical host in the cluster runs an Intel Core 2 Duo E7200 CPU, with 3MB of L2 cache and clocked at 2.53GHz, 2GB of DDR2 RAM, and a 500GB SATA2 hard disk. All the hosts were configured with Xen and the Xenos framework. One of the hosts, which we refer to as the master host, was configured with a template of the Hadoop slave service as well as an instance of the Hadoop master node, from where we issue commands to deploy services and execute tests; however, it was also configured not to accept service instances of the Hadoop slave service, meaning that we have 12 hosts on which to instantiate Hadoop slave services. The master host was also set as a JXTA rendezvous server, and all the Xenos hosts configured to use it. All physical hosts were assigned fixed IP addresses, and a DHCP server was configured on the master host to allocate addresses to spawned domains.

In all of our tests except where stated, the domain that we use as the Hadoop slave template which is replicated to

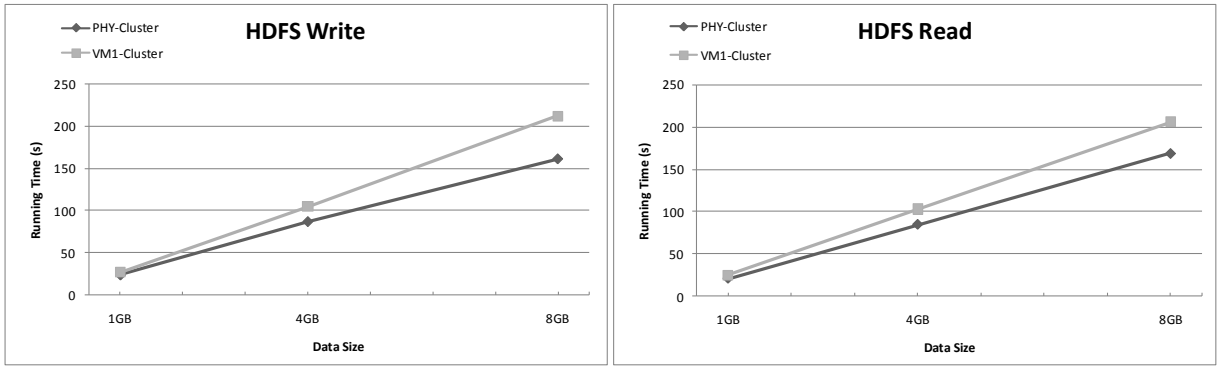

Figure 4. PHY-Cluster *vs* VM1-Cluster with varying data sizes.

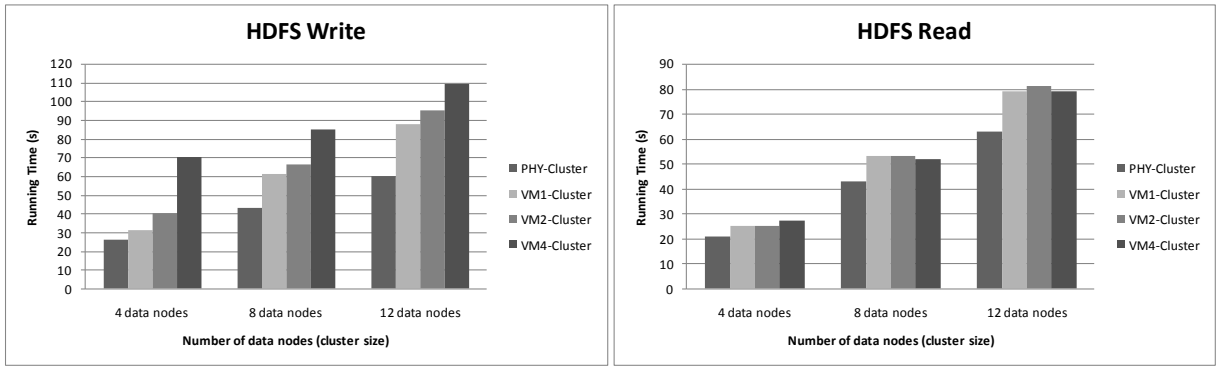

Figure 5. PHY-Cluster *vs* VM clusters with varying data nodes (cluster size) and virtual machines per physical machine.

all the hosts is configured with a 10GB disk image, 1GB swap image, and the vmlinuz-2.6.24-27-xen kernel. Default settings for the domain are 1 virtual CPU (VCPU), 384MB of RAM, and set to use DHCP to obtain addresses. Domain 0 is set to use 512MB of memory, leaving the rest to be allocated to domains, and has no restrictions on the physical CPUs it can use. For the Hadoop tests run on the native, non-virtualized Linux distribution, the same cluster and same Linux installation that is used as Domain 0 is used, but without booting into Xen so that the operating system runs natively. In all our tests, the HDFS replication factor is set to 2, and we do not use rack awareness since our network only has one switch. Each task tracker is set to execute a maximum of 2 map-tasks and 2 reduce-tasks at any given time. No optimizations to Hadoop or any other software component were made to suit this particular cluster.

1) Replication and Migration of Hadoop Service Templates and Services: The unoptimized replication process took around 45 minutes to deploy a template and a single slave service instance to each of the twelve cluster hosts, which included a network transfer of 132GB as well as another 132GB in local data copying; this translates to a network throughput of around 40MB/s and a disk throughput of around 25MBs/s. Since the process mostly involves transferring domain files over the network and copying them locally, its performance depends on the hardware platform that the services are being deployed on, as well as the size of the domains that contain the service. Other operations performed during replication, such as issuing Xenos API calls and updating local configuration files are typically sub-second operations that do not affect the overall performance. Further optimizations such as using LAN broadcasts on the cluster to transfer the service to the hosts can be implemented to minimize the time required for deployment.

In order to test the migration of the Hadoop master instance, a Hadoop Wordcount benchmark was initiated on the master, using a single slave instance deployed on each cluster. About half-way through the job, we issue a migration request from a small application in the master host to the Xenos API in the same host, instructing Xenos to migrate the master instance to another host on the cluster, which is automatically located through a peer-to-peer search. This causes the master to be paused while its files are moved and its state migrated, inevitably causing some map tasks on the slave services to fail, since they are not able to report back to the master. Once the master has migrated, it is un-paused and resumes executing the job, and the failed map tasks are re-executed by Hadoop itself. The job finishes successfully, although as expected it takes more time than if it were not migrated, due to the re-execution of failed map tasks. The migration itself takes less than 5 minutes, which is practically the time needed to transfer the domain files and the memory from the original host to the target host, and to resolve the new address of the host. Once again,

the operations issued by Xenos are lightweight and have a negligible affect on the overall duration.

2) HDFS Performance: To evaluate the performance of the HDFS on which Hadoop map-reduce relies, we designed a series of tests to measure its performance when reading and writing data in both a physical and virtualized cluster setup. Our main objective is to determine the performance penalty suffered by I/O operations in virtualized environments, using different data set sizes, cluster configurations and number of HDFS datanodes. All read and write operations were issued using the *get* and *put* operations provided by Hadoop, which allows reading from the HDFS to the local filesystem and writing from the local filesystem to the HDFS respectively. The data written into the HDFS in all tests is a large text file automatically generated by scripts beforehand. In all results, PHY-Cluster refers to a Hadoop cluster on native Linux, while VM1-Cluster, VM2-Cluster and VM4-Cluster refer to Xenos-enabled virtualized clusters with one, two and four virtualized Hadoop slave services deployed per physical host respectively.

We first evaluate the performance of the HDFS when reading and writing different data sizes (1GB, 4GB and 8GB) under a physical and a virtualized environment with only one service instance per host (VM1-Cluster). 12 cluster hosts are used in all these tests, resulting in 12 datanodes being made available to the Hadoop master. As shown in Figure 4, PHY-Cluster performs better than VM1-Cluster in both reading and writing, which is expected due to the overheads typical in virtual machines. While the performance gap is marginal for the 1GB data set, which translates to around 85MB per datanode, the gap increases with bigger data sets that involve more data per node.

Another evaluation carried out for HDFS is to identify whether the number of virtualized service instances on each physical host affects read and write performance. For each test, we read and write 256MB for each datanode, as in the previous test, meaning 1GB, 2GB and 3GB for 4, 8 and 12 datanodes respectively. As shown in Figure 5, PHY-Cluster once again outperforms all the virtualized setups as expected. However, we note an interesting difference between reading and writing on virtualized datanodes; when writing, the performance gap grows significantly larger as the number of datanodes increases, but remains stable when reading. Ibrahim *et al.* [20] also make this observation in one of their tests, indicating that the write performance gap increased markedly but it increased only slightly when reading.

3) Hadoop Benchmarks: One of the possible benefits of running Hadoop jobs within virtual machines is increasing the amount of computation nodes thus using the physical processing resources available more efficiently. To evaluate this, we execute several benchmark jobs that are provided as examples by Hadoop. In all of the evaluations presented below, we execute Hadoop jobs on the physical (native) cluster and three other virtualized cluster setups, as shown in Table I. We use 12 physical hosts throughout all tests, but since the number of service instances (VMs) per host changes, we have a different amount of Hadoop nodes available in certain setups. We again refer to these setups as PHY-Cluster, VM1-Cluster, VM2-Cluster and VM4-Cluster. Note that since the domain of each service instance is set to use 1 VCPU, when deploying four domains on each physical host, the total number of VCPUs on the host is larger than the number of physical CPU cores available, which can have a detrimental effect on the performance of the domains on the host, due to CPU contention. The best VCPU to CPU core ratio is in the VM2-Cluster case, where each VCPU is mapped to a CPU core, as shown in Table I.

The Wordcount benchmark counts the occurrence of each word in a given text file, and outputs each word and its associated count to a file on the HDFS. Each mapper takes a line as input, tokenizes it and outputs a list of words with the initial count of each, which is 1. The reducer sums the counts of each word and outputs the word and its associated count. We execute the benchmark varying the input data size, using 1GB and 8GB data files. As shown in Figure 6, the performance of the VM2-Cluster and VM4-Cluster is better than the VM1-Cluster, indicating that the extra computation nodes being made available are providing a performance benefit. However, the performance of PHY-Cluster is still better than all the virtualized clusters. In the Wordcount benchmark, Ibrahim *et al.* [20] achieved better performance on their virtualized clusters with 2 and 4 VMs per physical host than their physical (native) cluster; however each host in their evaluations was equipped with 8 cores, and their VCPU to CPU core ratio is always less than 1.

The Sort benchmark sorts an input file containing <key,value> pairs and outputs the sorted data to the file system. The input data is usually generated using the RandomWriter sample application provided with Hadoop, which can be used to write large data sets to the HDFS, consisting of sequence files. The mappers reads each record and outputs a <key, record> pair, sorting them in the process, and the reducer simply outputs all the pairs unchanged. We execute the benchmark varying the input data size, using 1GB and 8GB data files. As shown in Figure 7, increasing the number of computation nodes does not result in a performance benefit for the VM2-Cluster and VM4-Cluster; their performance actually degrades significantly. During the tests we observed that while the mappers started executing at a quick rate, once the reducers started executing, the whole job execution slowed down considerably; this was also observed by Ibrahim *et al.* [20] in their evaluation of the Sort benchmark. The authors argue that this can be attributed to the large amount of data transferred from the mappers to the reducers when they start, causing the virtualized Hadoop nodes on the same physical host to compete for I/O resources. The performance of the HDFS is also a factor,

	PHY-Cluster	VM1-Cluster	VM2-Cluster	VM4-Cluster
Services (VMs)	-	1 VM/host	2 VM/host	4 VM/host
Hadoop nodes	12	12	24	48
VCPU : CPU core ratio	-	1:2	1:1	2:1

Table I
THE PHYSICAL AND VIRTUALIZED CLUSTERS SETUPS ON WHICH HADOOP JOBS ARE EXECUTED.

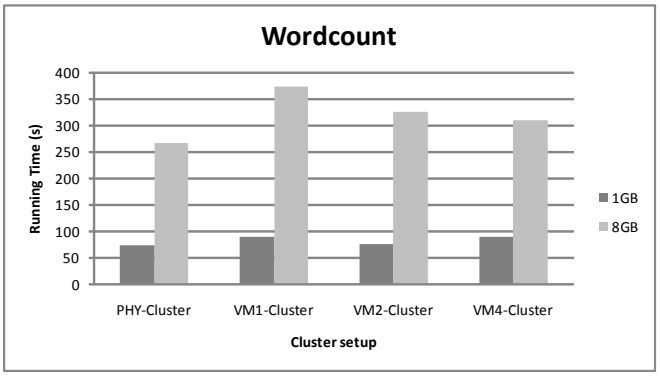

Figure 6. Wordcount execution on PHY-Cluster and VM clusters with varying data input size and virtual machines per physical machine.

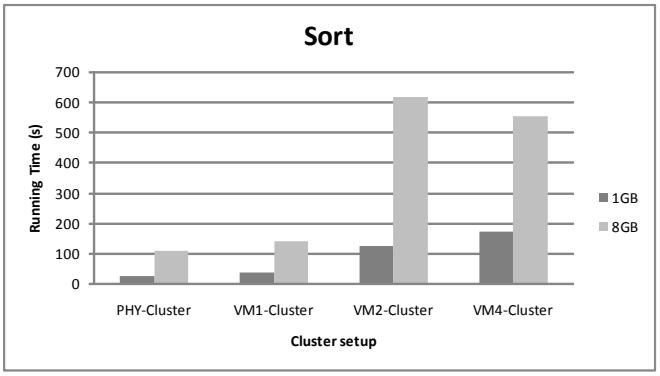

Figure 7. Sort execution on PHY-Cluster and VM clusters with varying data input size and virtual machines per physical machine.

since large amounts of data are being read and written at the same time.

The PiEstimator application included with Hadoop uses a quasi-Monte Carlo method to estimate the value of Pi. The mappers generate points in a unit square, and then count the points inside and outside of the inscribed circle of the square. The reducers accumulate these points from the mappers, and estimate the value of Pi based on the ratio of inside to outside points. The job takes as input the number of mappers to start, and the number of points to generate for each mapper; 120 mappers and 10,000 points were used in all tests. As shown in Figure 8, the performance of VM2-Cluster and VM4-Cluster shows a decisive improvement over VM1-Cluster, since more processing nodes are available, and very little I/O operations are done on the HDFS. In order to verify whether the ratio of VCPUs to physical cores has an effect on performance, we setup a VM1-Cluster with each VM assigned 2 VCPUs instead of 1; this resulted in the same amount of Hadoop nodes available, but each node has an extra VCPU compared to the standard VM1-Cluster. We ran the PiEstimator tests on this cluster and noticed a considerable performance improvement, although not as high as the VM2-Cluster and VM4-Cluster. Interestingly enough, we performed a test on PHY-Cluster where we restricted the Linux kernel to use only a single core, expecting to see a performance degradation when compared with a dual-core setup. However there was no degradation; this could be a deficiency with this particular Hadoop job or Hadoop itself, although it does not explain how VM1-Cluster with two VCPUs achieved better performance than with a single VCPU. It would be an interesting exercise to perform more tests, varying parameters such as the number of map-tasks and reduce-tasks allowed per node, to identify the reasons for this observation.

D. Summary of Results

Using the Hadoop map-reduce framework and the HDFS as a test case, we evaluated the Xenos framework in terms of the functionality and features that it provides, and whether

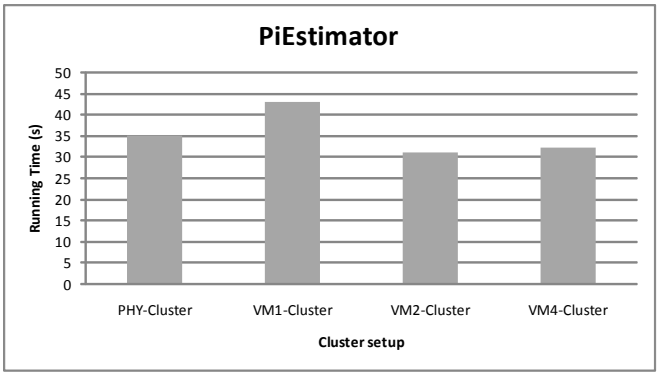

Figure 8. PiEstimator execution on PHY-Cluster and VM clusters with varying virtual machines per physical machine.

the use of virtualization introduces a performance penalty that might not make it feasible to use such a framework for certain data intensive applications such as Hadoop. Using the functionality exposed by the Xenos API, we successfully developed an application that automatically deploys Hadoop map-reduce services over a cluster, allowing the user to specify the number of cluster hosts to use, and the number of services per host. We also successfully migrated the Hadoop master node from its original host to another; the node resumed without issues and eventually the job was completed.

As expected, reading and writing operations on the HDFS when run on a virtualized cluster suffers a performance penalty when compared to a physical (native) cluster setup. For small data transfers and cluster setups, the gap is negligible, but increases steadily when involving large data sets or a large number of cluster nodes. While we acknowledge that these I/O performance penalties can be a barrier when adopting virtualization, a significant amount of ongoing work and research is aiding in reducing this cost and increasing hardware support for virtual I/O.

Increasing the number of computation nodes by adding more virtualized service instances per physical host benefits certain Hadoop jobs that are processor bound, since more efficient use of the physical processing resources is being made. However, jobs that are more I/O bound and that deal with large data sets tend to suffer a performance hit due to the performance degradation of the HDFS. For this reason, an interesting experiment would be to separate the HDFS from the service instances, which become computation nodes that execute the Hadoop tasks but use a non-virtualized HDFS, and evaluate whether any performance benefits are obtained.

To summarize, we have shown that any negative performance effects arising from using Xenos are related to the deficiencies in current virtualized I/O systems, and not due to the overhead imposed by Xenos, which is kept to a minimum. A peer-to-peer virtualized services platform similar to Xenos allows for rapid deployment of services, with the additional benefit that it does not require applications to be modified. We have also shown that leveraging the superior search capabilities of peer-to-peer networks and virtualization features such as migration allows for a more scalable and resilient approach to dynamic service provisioning. Once a platform like Xenos is in place, we can focus on managing services instead of managing physical machines.

VI. Topics For Further Investigation

Xenos can fill the role of a test-bed to facilitate experimentation with a variety of emerging issues in distributed virtualized services, some of which are briefly discussed here.

A. A Library of Essential Services

The core functionality provided by the Xenos framework can be further extended and abstracted away through additional services. Examples include service wrappers for load-balancing and fault-tolerance (virtual machine checkpointing is invisible to the service(s) hosted within), virtual machine pooling and replication, service deployers such as the Hadoop deployer discussed previously, platform emulators, legacy services supporting a range of operating systems, and a Xenos-UDDI adapter that can be used to search for Xenos services via UDDI. Xenos does not impose a single method for actual service delivery, thus web services, Sun RPC, and even services using raw Ethernet may be advertised.

B. Seamless Wide-Area Service Migration

The issue of live virtual machine migration over WANs has been addressed by several authors and a number of prototypes are available. Travostino *et al.* [22] approach the problem of preserving TCP connections by creating dynamic IP tunnels and assigning a fixed IP address to each virtual machine, which communicates with clients via a virtual gateway interface that is set up by Xen. After migration, a virtual machine retains its address, and the IP tunnels are configured accordingly to preserve network routes – this is completely transparent to TCP or any other higher level protocol. Bradford *et al.* [23] combine the IP tunneling approach with

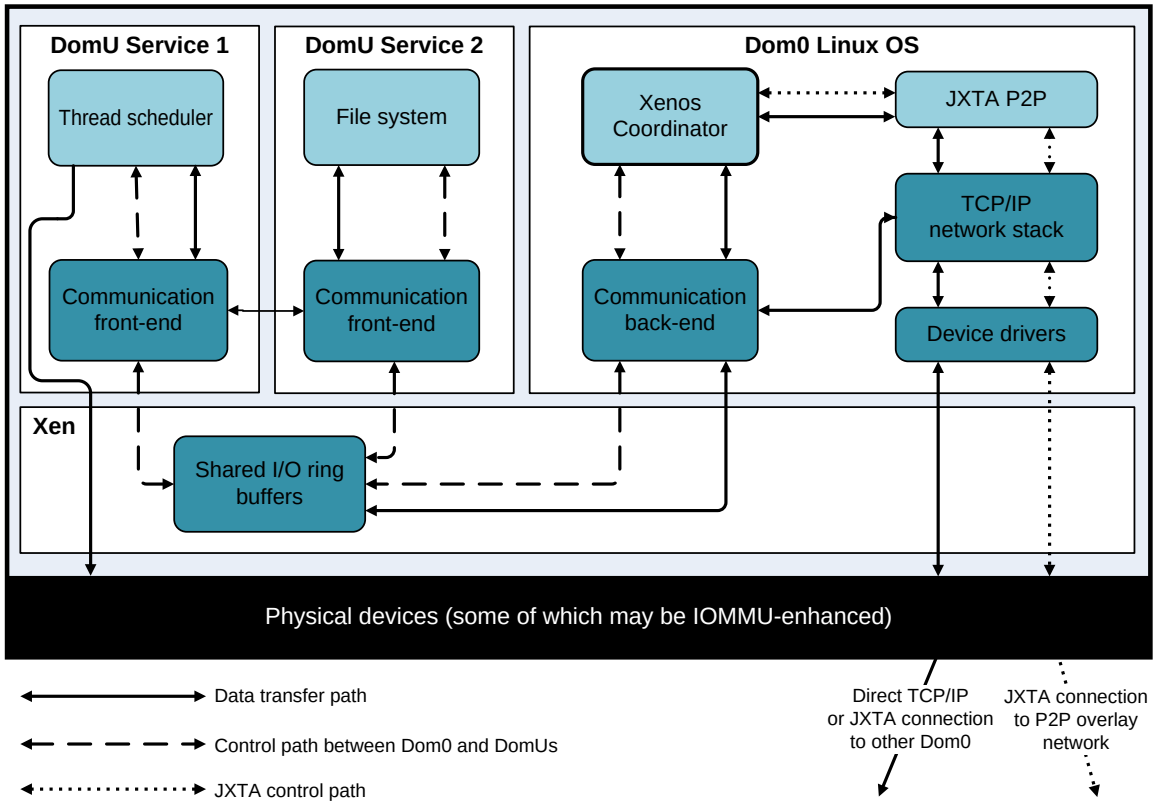

Figure 9. An alternate architecture allowing for various transport methods.

Dynamic DNS to address the problem of preserving network connections. More importantly, the authors also implement a pre-copy approach for transferring the disk image attached to a virtual machine, using a mechanism similar to that used by Xen when live migrating the state of a virtual machine. This greatly minimizes downtime even if the actual migration takes long due to poor network performance. Harney *et al.* [24] suggest using the mobility features in the IPv6 protocol to preserve network communication sessions, an approach that is viable in the long-term.

C. Alternative Transport Methods For Service Delivery

Applications featuring fine grained concurrency spanning across virtual and physical machines stand to gain from inter-virtual machine communication path optimizations such as shared memory communication for services residing on the same physical machine, and hypervisor-bypass network communication for distributed services. In both instances, the secure initialization of each communication path would be delegated to Xenos, allowing the data to move directly between the participating virtual machines and virtualization-enabled I/O devices. In some cases, an I/O could be permanently and exclusively bound to a specific service for low-latency dedicated access. Another enhancement is to allow services to piggy-back their communication over the JXTA protocol, which would allow communication between services that cannot reach one another outside of the peer-to-peer network. Figure 9 illustrates this concept.

D. Security, Authentication and Service Provisioning

A number of underlying mechanisms could be inherited from the Xen hypervisor and the JXTA peer-to-peer framework or their respective alternatives. To our benefit, JXTA provides several security and authentication features, as discussed by Yeager *et al.* [25]; these include TLS (Transport Layer Security), and support for centralized and distributed certification authorities. Xen provides a basis for automated accounting and billing services that track service consumption as well as physical resource use. However, Xenos should at least provide unified and distributed user, service and hierarchical service group authentication and permission mechanisms, a non-trivial undertaking in itself.

E. The Operating System-Agnostic Operating System

Software architectures in the vein of Xenos could fit the role of a distributed microkernel in a virtualization-embracing operating system that consists of interacting light-weight services hosted within virtual machines, including a multi-core thread scheduler, file systems (a stripped down Linux kernel), and device drivers. Each operating system service would run within its own light-weight Xen domain and expose itself through Xenos services (reminiscent of system calls). Xenos services would also host legacy operating systems and applications, presented to users through an

operating system-agnostic window manager hosted in a separate virtual machine. Applications with particular resource requirements or requiring isolation, such as computer games or web browsers, may easily be hosted in their own virtual machines, supported by a minimal application-specific kernel or library or even executing on 'bare virtualized metal'. Xen, and virtual machine monitors in general, have been described as "microkernels done right" [26], although others have argued that the drawbacks that muted the adoption of microkernels [27] still apply.

VII. Conclusion

An approach to building distributed middleware where services are hosted within virtual machines interconnected through a peer-to-peer network has been presented through the Xenos framework. Xenos extends well-established solutions for virtualization hypervisors and peer-to-peer overlay networks to deliver the beginnings of a fully decentralized solution for virtualized service hosting, discovery and delivery.

Using the Hadoop map-reduce framework and the HDFS as a test case, it was established that minimal performance overheads are associated with using the Xenos framework itself, and that the overheads introduced through the use of virtual machines are principally linked with the incidence of I/O operations. It is expected that forthcoming hardware support for virtualization will further reduce the gap between virtualized and native I/O performance pinpointed in the results, while simplifying hypervisors. This will further consolidate the virtual machine's position as a viable alternative for hosting both computation- and I/O-intensive tasks.

In practice, Xenos automated to a large degree the deployment of jobs while enabling the seamless migration of live Hadoop nodes. We thus believe that the combination of peer-to-peer technology and virtualization merits serious consideration as a basis for resilient distributed services.

References

[1] D. Bailey and K. Vella, "Towards peer-to-peer virtualized service hosting, discovery and delivery," in *AP2PS '10: Proceesings of the The Second International Conference on Advances in P2P Systems*, 2010, pp. 44–49.

[2] D. Bailey, "Xenos: A service-oriented peer-to-peer framework for paravirtualized domains," Master's thesis, University of Malta, 2010.

[3] P. Willmann, S. Rixner, and A. L. Cox, "Protection strategies for direct access to virtualized I/O devices," in *ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2008, pp. 15–28.

[4] K. A. Fraser, S. M. Hand, T. L. Harris, I. M. Leslie, and I. A. Pratt, "The XenoServer computing infrastructure," University of Cambridge Computer Laboratory, Tech. Rep., 2003.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM, 2003, pp. 164–177.

[6] D. Spence and T. Harris, "XenoSearch: Distributed resource discovery in the XenoServer open platform," in *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 2003, p. 216.

[7] A. Ganguly, A. Agrawal, P. O. Boykin, and R. Figueiredo, "WOW: Self-organizing wide area overlay networks of virtual workstations," in *In Proc. of the 15th International Symposium on High-Performance Distributed Computing (HPDC-15*, 2006, pp. 30–41.

[8] M. Amoretti, F. Zanichelli, and G. Conte, "SP2A: a service-oriented framework for P2P-based grids," in *MGC '05: Proceedings of the 3rd international workshop on Middleware for grid computing*. New York, NY, USA: ACM, 2005, pp. 1–6.

[9] V. March, Y. M. Teo, and X. Wang, "DGRID: a DHT-based resource indexing and discovery scheme for computational grids," in *ACSW '07: Proceedings of the fifth Australasian symposium on ACSW frontiers*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2007, pp. 41–48.

[10] Q. Xia, R. Yang, W. Wang, and D. Yang, "Fully decentralized DHT based approach to grid service discovery using overlay networks," *Computer and Information Technology, International Conference on*, pp. 1140–1145, 2005.

[11] D. Talia, P. Trunfio, J. Zeng, and M. Hgqvist, "A DHT-based peer-to-peer framework for resource discovery in grids," Institute on System Architecture, CoreGRID - Network of Excellence, Tech. Rep. TR-0048, June 2006.

[12] W. Wadge, "Providing a grid-like experience in a P2P environment," Master's thesis, University of Malta, 2007.

[13] M. F. Mergen, V. Uhlig, O. Krieger, and J. Xenidis, "Virtualization for high-performance computing," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 2, pp. 8–11, 2006.

[14] S. Thibault and T. Deegan, "Improving performance by embedding HPC applications in lightweight Xen domains," in *HPCVirt '08: Proceedings of the 2nd workshop on System-level virtualization for high performance computing*. New York, NY, USA: ACM, 2008, pp. 9–15.

[15] M. J. Anderson, M. Moffie, and C. I. Dalton, "Towards trustworthy virtualisation environments: Xen library OS security service infrastructure," Hewlett-Packard Laboratories, Tech. Rep. HPL-2007-69, April 2007. [Online]. Available: http://www.hpl.hp.com/techreports/2007/HPL-2007-69.pdf

[16] K. Falzon, "Thread scheduling within paravirtualised domains," Bachelor of Science (Hons) Dissertation, University of Malta, 2009.

[17] R. J. Figueiredo, P. A. Dinda, and J. A. B. Fortes, "A case for grid computing on virtual machines," in *ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE Computer Society, 2003, p. 550.

[18] K. Keahey, K. Doering, and I. Foster, "From sandbox to playground: Dynamic virtual environments in the grid," in *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 34–42.

[19] S. Santhanam, P. Elango, A. Arpaci-Dusseau, and M. Livny, "Deploying virtual machines as sandboxes for the grid," in *WORLDS'05: Proceedings of the 2nd conference on Real, Large Distributed Systems*. Berkeley, CA, USA: USENIX Association, 2005, pp. 7–12.

[20] S. Ibrahim, H. Jin, L. Lu, L. Qi, S. Wu, and X. Shi, "Evaluating MapReduce on virtual machines: The Hadoop case," in *CloudCom*, 2009, pp. 519–528.

[21] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[22] F. Travostino, P. Daspit, L. Gommans, C. Jog, C. de Laat, J. Mambretti, I. Monga, B. van Oudenaarde, S. Raghunath, and P. Y. Wang, "Seamless live migration of virtual machines over the MAN/WAN," *Future Gener. Comput. Syst.*, vol. 22, no. 8, pp. 901–907, 2006.

[23] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg, "Live wide-area migration of virtual machines including local persistent state," in *VEE '07: Proceedings of the 3rd international conference on Virtual execution environments*. New York, NY, USA: ACM, 2007, pp. 169–179.

[24] E. Harney, S. Goasguen, J. Martin, M. Murphy, and M. Westall, "The efficacy of live virtual machine migrations over the internet," in *VTDC '07: Proceedings of the 3rd international workshop on Virtualization technology in distributed computing*. New York, NY, USA: ACM, 2007, pp. 1–7.

[25] W. Yeager and J. Williams, "Secure peer-to-peer networking: The JXTA example," *IT Professional*, vol. 4, pp. 53–57, 2002.

[26] S. Hand, A. Warfield, K. Fraser, E. Kotsovinos, and D. Magenheimer, "Are virtual machine monitors microkernels done right?" in *HOTOS'05: Proceedings of the 10th conference on Hot Topics in Operating Systems*. Berkeley, CA, USA: USENIX Association, 2005.

[27] G. Heiser, V. Uhlig, and J. LeVasseur, "Are virtual-machine monitors microkernels done right?" *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 1, pp. 95–99, 2006.