Università di Pisa
Dipartimento di Informatica

Technical Report: TR-07-13

VirtuaLinux Design Principles

Marco Aldinucci Massimo Torquati Marco Vanneschi
Dipartimento di Informatica, Università di Pisa, Italy

Manuel Cacitti Alessandro Gervaso
Pierfrancesco Zuccato
Eurotech S.p.A, Italy

June 21, 2007

ADDRESS: largo B. Pontecorvo 3, 56127 Pisa, Italy. TEL: +39 050 2212700 FAX: +39 050 2212726

VirtuaLinux Design Principles[*]

Marco Aldinucci[†] Massimo Torquati[†] Marco Vanneschi[†]
Manuel Cacitti[‡] Alessandro Gervaso[‡]
Pierfrancesco Zuccato[‡]

June 21, 2007

Abstract

VirtuaLinux is a Linux meta-distribution that allows the creation, deployment and administration of both physical and virtualized clusters with no single point of failure. They are avoided by means of a combination of architectural, software and hardware strategies, including the transparent support for disk-less and master-less cluster configuration. VirtuaLinux support the creation and management of virtual clusters in seamless way: VirtuaLinux Virtual Cluster Manager enables the system administrator to create, save, restore Xen-based virtual clusters, and to map and dynamically re-map them onto the nodes of the physical cluster. Master-less, disk-less and virtual clustering relies on the novel VirtuaLinux disk abstraction layer, which enables the fast (almost constant time), space-efficient, dynamic creation of virtual clusters composed of fully independent complete virtual machines. VirtuaLinux has been jointly designed and developed by the Computer Science Dept. (HPC lab.) of the University of Pisa and Eurotech HPC lab., a division of Eurotech S.p.A. VirtuaLinux is a open source software under GPL available at `http://virtualinux.sourceforge.net/`.

[*]VirtuaLinux has been developed at the HPC lab. of Computer Science Dept. - University of Pisa and Eurotech HPC, a division of Eurotech Group. VirtuaLinux project has been supported by the initiatives of the LITBIO Consortium, founded within FIRB 2003 grant by MIUR, Italy.

[†]Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo 3, I-56127 Pisa, Italy. WEB: `http://www.di.unipi.it/groups/architetture/` E-mail: {`aldinuc, torquati, vannesch`}`@di.unipi.it`

[‡]Eurotech S.p.A., Via Fratelli Solari 3/a, I-33020 Amaro (UD), Italy. WEB: `http://www.eurotech.com` E-mail: {`m.cacitti, a.gervaso, p.zuccato`}`@exadron.com`

Contents

1 Introduction

A computer cluster is a group of tightly coupled computers that work together closely so that in many respects they can be viewed as a single computing platform. The computers comprising a cluster (nodes) are commonly connected to each other through one or more fast local area networks. Clusters are usually deployed to improve performance and/or availability over that provided by a single computer, while typically being much more cost-effective than single computers of comparable speed or availability.
The kind of nodes may range over the complete spectrum of computing solutions, from large Symmetric MultiProcessor (SMP) computers to bare uniprocessor desktop PCs. Intermediate solutions, such as dual/quad-core processors, are nowadays popular because they can fit in very compact blades, each of them implementing a complete multi-processor-multi-core computer. As an example, *Eurotech* clusters are currently shipped in a 4U case hosting 8 blades, each of them equipped with two dual-core CPUs (32 cores per case). Larger clusters can be assembled by wiring more than one case [5].

A wide range of solutions is also available for node networking. High-density clusters are commonly equipped with at least a switched Ethernet network for its limited cost and wide support in operating systems. Recently, the Infiniband network has also become increasingly popular due its speed-cost trade-off [10]. As an example, Eurotech clusters are currently shipped with one switched fast Ethernet, two switched Giga-Ethernets, and one 10 Gbits/s Infiniband NICs per node (Infiniband switch is not included in the 4U case). Since the nodes of a cluster implement complete computers, each node is usually equipped with all standard devices, such as RAM memory, disk, USB ports, and character I/O devices (keyboard, video, mouse). High-density clusters typically include a terminal concentrator (KVM) to enable the control of all nodes from a single console. Also, they include a hard disk per blade and are possibly attached to an external Storage Area Network (SAN). A disk mounted on a blade is typically installed with the node Operating System (OS) and might be used, either locally or globally, to store users data. A SAN is typically used to store user data because of its shared nature. A blade for high-density clusters can be typically fitted with no more than one disk, which is typically quite small and slow (e.g. 2.5 inches SATA disk, 80-120 MBytes, 5-20 MBytes/s) due to space, power, and cooling limitations. A SAN usually exploits a Redundant Array of Independent Disks (RAID) and offers large storage space, built-in fault-tolerance, and medium-large I/O speed (e.g. 10 TBytes, 400-800 MBytes/s). SANs are usually connected to a cluster via Fiber Channel and/or other Gigabit-speed technology such as Ethernet or Infiniband.

The foremost motivation of clusters popularity comes from their similarity with a network of complete computers. They are built by assembling standard components thus, on the one hand they are significantly cheaper than parallel machines, and on the other hand they can benefit from already existing and tested software. In particular the presence of a private disk space for each node enables the installation of standard (possibly SMP-enabled) OSes that can be managed by non-specialized administrators. Once installed, nodes of a cluster can be used in insulation to run different applications or cooperatively to run parallel applications. Also, a number of tools for the centralized management of nodes are available in all OSes [25]. These tools typically use client-server architecture: one of the nodes of the cluster acts as the master of the cluster, while others depend on it. The master is usually statically determined at the installation time for its hardware (e.g. larger disks) or software (e.g. services configuration).

1.1 Common Flaws of Classical Clusters

Unfortunately, disks mounted on blades, especially within high-density clusters, are statistically the main source of failures because of the strict constraints of size, power and temperature [19, 24]. Moreover, they considerably increase cluster engineering complexity and power requirement, and decrease density. This is particularly true on the master disk, that happens also to be a critical single point of failure for cluster operation since it hosts services involving file sharing, user authentication, Internet gateway, and in the case of disk-less cluster also root file system access, IP management and network boot services. A hardware or software crash/malfunction on the master is simply a catastrophic event for cluster stability. Moreover, cluster management exhibits several critical issues:

- First installation and major OS upgrades are very time consuming, and during this time the cluster should be set offline. Node hot-swapping usually requires cluster reconfiguration.
- Rarely a single configuration or even a single OS can be adapted to supply all user needs. Classic solutions like static cluster partitioning with multiple boots are static and not flexible enough to consolidate several user environments and require an additional configuration effort.
- Since cluster configuration involves the configuration of distinct OS copies in different nodes, any configuration mistake, which may seriously impair cluster stability, is difficult to undo.

We present a coupled hardware-software approach based on open source software aiming at the following goals:

1. Avoiding fragility due to the presence of disks on the blades by removing disks from blades (high-density disk-less cluster) and replacing them with a set of storage volumes. These are abstract disks implemented via an external SAN that is accessed via suitable protocols.
2. Avoiding single point of failure by removing the master from the cluster. Master node features, i.e. the set of services implemented by the master

node, are categorized and made redundant by either active or passive replication in such a way they are, at each moment, cooperatively implemented by the running nodes.

3. Improving management flexibility and configuration error resilience by means of transparent node virtualization. A physical cluster may support one or more virtual clusters (i.e. cluster of virtual nodes) that can be independently managed and this can be done with no impact on the underlying physical cluster configuration and stability. Virtual clusters run a guest OS (either a flavor of Linux or Microsoft Windows) that may differ from the host OS, governing physical cluster activities.

These goals are achieved independently through solutions that have been designed to be coupled, thus to be selectively adopted. A suite of tools, called VirtuaLinux, enables the boot, the installation, the configuration and the maintenance of a cluster exhibiting the previously described features. VirtuaLinux is currently targeted to AMD/Intel x86 64-based nodes, and includes:

- One or more Linux distributions, currently Ubuntu Edgy 6.10 and CentOS 4.4.
- An install facility able to install and configure included distributions according to goals 1-3.
- A recovery facility able to revamp a misconfigured node.
- A toolkit to manage virtual clusters (VVCM) and one or more pre-configured virtual cluster images (currently Ubuntu Edgy 6.10 and CentOS 4.4).

In the following sections we will describe how goals 1-3 are achieved. Features a-d are described in Appendix C.

1.2 No Single Point of Failure

There is no such thing as a perfectly reliable system. Reliability engineering cannot engineer out failure modes that are not anticipated by modeling. For this reason, usually, reliable systems are specified at and designed to some non-zero failure rate (e.g. 99.99% availability). The main engineering approaches toward reliable systems design are (in order of importance):

- eliminating single points of failure ("no single point of failure");
- engineering any remaining single points of failure to whatever level is necessary to reach the system specification;
- adding extra system safety margins to allow for errors in modeling or implementation.

Single point of failure describes any part of the system that can, if it fails, cause an interruption of required target service. This can be as simple as a process failure or as catastrophic as a computer system crash. The present work aims to remove several significant single points of failure in cluster organization at the OS level. This means the target service is cluster functionality at the OS level, not a particular service running on one or more nodes of the cluster. In particular, VirtuaLinux aims to guarantee that in a cluster the following proprieties are ensured:

- if some node crash due to a hardware or software failure, not crashed nodes remain fully functional independently of the identity of the crashed nodes (full symmetry of nodes);
- crashed nodes can be repaired and restarted with no impact on other running nodes. Nodes can be hot-removed or hot-added to the cluster (nodes are hot-swappable).

VirtuaLinux is able to support these properties provided it is running on suitable hardware, that exhibits a sufficient redundancy level of cluster support devices, like power feeds, network connections, routers, and router interconnections. The same kinds of assumptions are made on the internal SAN architecture and the connections among the SAN and the cluster. Note that for mission-critical systems, the mere use of massive redundancy does not make a service reliable because the whole system itself is single point of failure (e.g. both routers are housed in a single rack, allowing a single spilled cup of coffee to take out both routers at once). Mission-critical related problems are not addressed by VirtuaLinux and are outside the scope of the present work.

2 Disk-less Cluster

2.1 Storage Area Network (SAN)

In computing, a storage area network (SAN) is a network designed to attach computer storage devices such as disk array controllers to servers. A SAN consists of a communication infrastructure, which provides physical connections, and a management layer, which organizes the connections, storage elements, and computer systems so that data transfer is secure and robust. SAN are distinguished from other forms of network storage by the low-level access method that they use (block I/O rather than file access). Data traffic on the SAN fabric is very similar to those used for internal disk drives, like ATA and SCSI. On the contrary, in more traditional file storage access methods, like SMB/CIFS or NFS, a server issues a request for an abstract file as a component of a larger file system, managed by an intermediary computer. The intermediary then determines the physical location of the abstract resource, accesses it on one of its internal drives, and sends the complete file across the network.

Sharing storage usually simplifies storage administration and adds flexibility since cables and storage devices do not have to be physically moved to move storage from one server to another. SANs tend to increase storage capacity utilization, since multiple servers can share the same growth reserve, and if compared to disks that a high-density cluster can accommodate, exhibit better performances and reliability since they are realized by arranging high-speed high-quality disks in RAID. SANs also tend to enable more effective disaster recovery processes. A SAN attached storage array can replicate data belonging to many servers to a secondary storage array. This secondary array can be local or, more typically, remote. The goal of disaster recovery is to place copies of data outside the radius of effect of an anticipated threat, the long-distance transport capabilities of SAN protocols.[1]

Note that, in general, SAN storage implements a one-to-one relationship. That is, each device, or Logical Unit Number (LUN) on the SAN is owned by a single computer. In reality, in order to achieve the disk-less cluster goal, it is essential to ensure that many computers can access the same disk abstraction over a network since the SAN should support all nodes of the cluster that are independent computers. In particular, the SAN should store the OS and swap space of each node, in addition to, possibly shared, applications data. Note that node OS kind is not Single-system image (SSI), thus each node requires owning a private read-write copy of several sub-trees of the root file system (e.g. `/var`). While several design alternatives are available to achieve this goal, probably the cleanest one consists in supplying each node with a private partition of the disk abstraction. VirtuaLinux implements a many-to-one abstract disk supporting a flexible partition mechanism by stacking iSCSI (Internet Small Computer System Interface) and EVMS (Enterprise Volume Management System):

- iSCSI is a network protocol standard, that allows the use of the SCSI protocol over TCP/IP networks. It enables many initiators (e.g. nodes) to access (read and write) a single target (e.g. SAN), but it does not ensure any coherency/consistency control in the case that many initiators access in read-write mode to the same partition [13].
- EVMS provides a single, unified system for handling storage management tasks, including the dynamic creation and destruction of volumes, which are EVMS abstractions behaving as disk partitions and enabling the access of volumes of the same target from different nodes [21].

iSCSI, EVMS and their role in VirtuaLinux design are discussed in the next sections.

2.2 Design alternatives: Network-Attached Storage (NAS)

The proposed approach is not the only possible one. Another approach consists in setting up a shared file system abstraction exported by the SAN to nodes of the cluster, i.e. a Network Attached Storage (NAS). SMB/CIFS or NFS are

[1] Demand for this SAN application has increased dramatically after the September 11th attacks in the United States, and increased regulatory requirements associated with Sarbanes-Oxley and similar legislation.

instances of a NAS. Note that file system abstraction (NAS) is a higher-level abstraction w.r.t. disk abstraction (SAN). This impacts both external storage (server) and cluster (client):

- The external storage should be able to implement a quite high-level protocol, and thus should be smart enough. For example, it can be realized with a computer running dedicated software (e.g. OpenFiler), which may become the single point of failure. In addition, the solution is not appropriate with operating system tools accessing directly the block device.
- The nodes of the cluster need a quite deep stack of protocols to access NAS, and thus need a considerable fraction of the OS functionality. These should be set up in the early stages of node boot in order to mount the root file system: in particular, these should be run from initial ramdisk (initrd). This has several disadvantages. First, the complexity to pivot running protocols from the initial ramdisk to the root file system, that may involve deep modifications to the standard OS configuration. Second, the impossibility for the OS to access the disk at the block level, thus independently from the file system, that strongly couples the methodology with the target OS. For example, the OS cannot use a swap partition, but just a swap file.

Overall, the VirtuaLinux design aims to reduce the dependency of disk-less architecture on OS-specific high-level protocols and to expose to the OS a collection of abstract nodes that are similar as possible to a classical cluster.

2.3 VirtuaLinux Storage Architecture

As mentioned above, EVMS provides a single, unified system for handling storage management tasks, including the dynamic creation and destruction of volumes, which are an EVMS abstraction that are seen from the OS as disk devices [6, 21].

The external SAN should hold a distinct copy of the OS for each node. At this end, VirtuaLinux prepares, during installation, one volume per node and a single volume for data shared among nodes. As we shall see later, several other volumes are used to realize virtual cluster abstraction. Volumes are formatted with an OS specific native file system (e.g. ext3), while shared volumes are formatted with a distributed file system that arbitrates concurrent reads and writes from cluster nodes, such as the Oracle Concurrent File System (OCFS2) or the Global File System (GFS).

Volumes are obtained by using the EVMS snapshot facility (see 2.3.1). A snapshot represents a frozen image of a volume of an original source. When a snapshot is created, it looks exactly like the original at that point in time. As changes are made to the original, the snapshot remains the same and looks exactly like the original at the time the snapshot was created. A file on a snapshot is a reference (at the level of disk block) to its original copy, and thus does no consume disk space while the original and its snapshot copy remain identical. A file is really stored in the snapshot, and thus consumes disk space only when either the original or its snapshot copy is modified. Indeed, snapshot creation is quite a fast operation.

The snapshot technique is usually used to build on-line backups of a volume: the accesses to the volume are suspended just for the (short) time of snapshot creation; then the snapshot can be used as on-line backup, which can be kept on-line either indefinitely or just for the time needed to store it on a different storage medium (e.g. tape). Multiple snapshots of the same volume can be used to keep several versions of the volume over time. As we shall see in Sec. 2.3.2, the management of a large number of snapshots requires particular care in current Linux systems.

VirtuaLinux installs an original volume with the selected OS distribution (called the default), and then creates n identical snapshots. Each node of the cluster uses a different snapshot as the root file system. Once snapshots have been made accessible *(activated)*, the content of both original and snapshots can evolve along different paths, as they are independent volumes. However, in the case of cluster management, snapshots have several advantages as compared to independent volumes:

- *Fast creation time.* Assume an n-node cluster is to be installed Since each node of the cluster requires a private disk, n independent volumes should be created at installation time starting from the same initial system distribution (e.g. CentOS system image). These volumes are physically stored in the same SAN due to the disk-less architecture. Creating these volumes by a standard copy loop may be extremely expensive in term of time since a complete Linux distribution should be installed n times.[2] As we shall see in Sec. 5, a similar amount of time should be spent for the creation of each new Virtual Cluster as well. Snapshot usage drastically decreases volume creation time since volume content is not copied but just referenced at the disk block level. Empirical experiences show that a snapshot of 10 GBytes volume is created in a few seconds on a Gig-Ethernet attached SAN.

- *Reduced disk space usage.* In the general case a snapshot requires at least the same amount of space as the original volume. This space is used to store original files in the case they are changed in the original volume after snapshot creation time, or new data stored in the snapshot that was not existing in the original volume at snapshot creation time. However, VirtuaLinux uses snapshots in a particular way: the original volume holds the root file system of the Linux distribution, which does not changes over time (when the original volume changes, the snapshots are reset). Since data in the original volume is immutable to a large degree (OS files), a considerable amount of disk space is saved with respect to full data copy.

[2]Estimated time depends on many factors, such as number of nodes, distribution size, DVD reader speed, SAN throughput. However, it can easily reach several hours even for small cluster configurations due the large number of small files that must be copied.

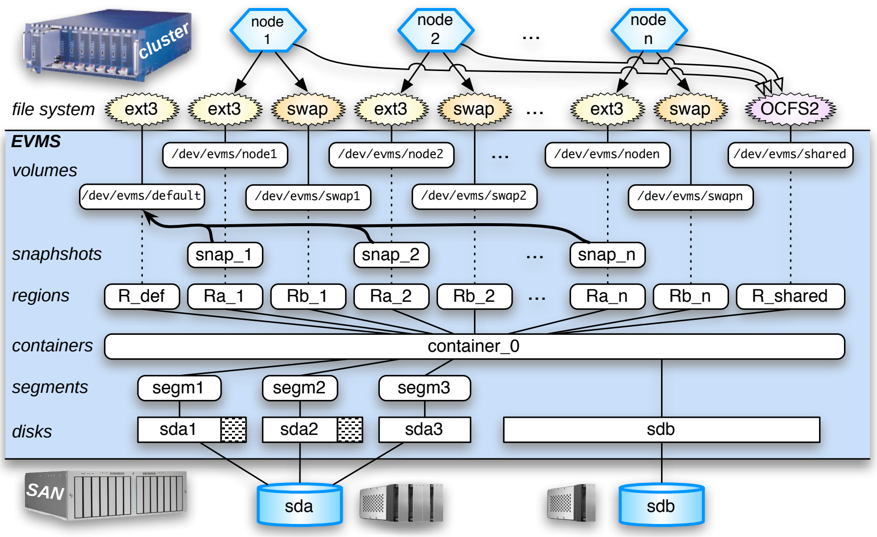

Figure 1: VirtuaLinux storage architecture.

As an example, if a snapshot volume Y (sizeof(Y)=y) is created from an original X (sizeof(X)=x), which stores an amount of z immutable data, then X is able to store an amount x of fresh data, for a total size available from X of almost $x + z$.

- *Device name independence.* EVMS ensures the binding of the raw device name (e.g. /dev/sda1) and logical volume name (e.g. /dev/evms/node1). Avoiding the use of raw device names is particularly important when using iSCSI connected devices since they may appear on different nodes with different names messing up system configuration (this typically happens when a node has an additional device with respect to other nodes, e.g. an external DVD reader).

- *Centralized management.* A snapshot can be reset to a modified version of the original. Data that has been changed in the snapshot is lost. This facility enables the central management of copies, as for example for major system updates that involves all nodes. This facility is not strictly needed for cluster management since all snapshots can be changed, as they are different copies by using classical cluster techniques such as broadcasted remote data distribution [25].

The architectural view of VirtuaLinux disk management is sketched in Fig. 1. Notice that since EVMS is a quite flexible and sophisticated management tool, the same goal can be achieved with different architectural designs, for example by using real volumes instead of snapshots with the EVMS cluster management facility. As discussed above, the VirtuaLinux design exhibits superior features with respect to alternative (and more classical) design options. The full description of EVMS functionality, which is outside the scope of this paper, can be found in [6, 21].

Note that VirtuaLinux uses the snapshot technique to provide a cluster with a number of independent volumes that can be efficiently created from a common template volume (original), whereas snapshots are usually used as transient, short-lived on-line backups. To the best of our knowledge, no other systems exhibit a similar usage of snapshots (and the consequent features). Indeed, in order to correctly exploit a different usage of snapshots, VirtuaLinux slightly extends EVMS snapshot semantics and implementation. This extension, which is described in the following sections, is correct with respect to EVMS snapshot semantics.

2.3.1 Understanding the Snapshot Technique

There are different implementation approaches adopted by vendors to create snapshots, each with its own benefits and drawbacks. The most common are *copy-on-write*, *redirect-on-write*, and *split mirror*. We briefly describe copy-on-write, which is adopted by EVMS; we refer back to the literature for an extensive description [9].

A snapshot of a storage volume is created using the pre-designated space for the snapshot. When the snapshot is first created, only the meta-data about where the original data is stored is copied. No physical copy of the data is made at the time the snapshot is created. Therefore, the creation of the snapshot is almost instantaneous. The snapshot copy then tracks the changing blocks on the original volume as writes to the original volume are performed. The original data that is being written to is copied into the designated storage pool that is set aside for the snapshot before the original data is overwritten.

Before a write is allowed to a block, copy-on-write moves the original data block to the snapshot storage. This keeps the snapshot data consistent with the exact time the snapshot was taken. Read requests to the snapshot volume of the unchanged data blocks are redirected to the original volume, while read requests to data blocks that have been changed are directed to the "copied" blocks in the snapshot. The snapshot contains the meta-data that describes the data blocks that have changed since the snapshot was first created. Note that the original data blocks are copied only once into the snapshot storage when the first write request is received.

In addition to the basic functionality, EVMS snapshots can be managed as real volumes, i.e. data can be added or modified on the snapshot with no impact on the original volume, provided that enough free space has been pre-allocated for the snapshot. Also, they can be activated and deactivated as standard volumes, i.e. mapped and unmapped onto Unix device drivers. However, despite being standard volumes, snapshots have a subtle semantics with respect to activation due to copy-on-write behaviour. In fact, the system cannot write on an inactive snapshot since it is not mapped to any device, thus may lose the correct alignment with its original during the deactivation period. EVMS solves the problem by logically marking a snapshot for reset at deactivation time, and resetting it to the current original status at activation time.

2.3.2 Snapshots as Independent Volumes: an Original Usage

As discussed above, VirtuaLinux uses EVMS snapshots to provide a cluster with a number of independent volumes that can be efficiently created from a common template volume (original). Since snapshots cannot be deactivated without losing snapshot private data, they all should always be kept active in all nodes, even if each node will access only one of them.

Snapshots on Linux OS (either created via EVMS, LVM, or other software) are managed as UNIX devices via the *device mapper* kernel functionality. Although EVMS does not fix any limit on the number of snapshots that can be created or activated, current Linux kernels establish a hardwired limit on the number of snapshots that can be currently active on the same node. This limit comes from the number of pre-allocated memory buffers (in kernel space) that are required for snapshot management. Standard Linux kernels enable no more than a dozen active snapshots at the same time. This indirectly constrains the number of snapshots that can be activated at the same time, and thus the number of nodes that VirtuaLinux can support.

Raising this limit is possible, but requires a non-trivial intervention on the standard Linux kernel code. VirtuaLinux overcomes the limitation with a different approach, which does not require modifications to the kernel code. It leverages on the following facts:

- Since each snapshot is used as private disk, each snapshot is required to be accessible in the corresponding node only. In this way, each node can map onto a device just one snapshot.
- The status of a EVMS snapshot is kept on the permanent storage. This information is also maintained in memory in terms of available snapshot objects. This information is maintained in a lazy consistent way. Status information is read at EVMS initialization time *(evms_activate)*, and committed out at any EVMS command (e.g. create, destroy, activate, deactivate a snapshot). While each snapshot can have just one status for all nodes on the permanent storage, it may have different status on the local memory of nodes (e.g. it can be mapped onto a device on a node, while not appearing on another).
- Snapshot deactivation consists in unmapping a snapshot device from the system, then logically marking it for reset on permanent storage. VirtuaLinux extends EVMS features with the option to disable EVMS snapshot reset-on-activate feature via a special flag in the standard EVMS configuration file. In the presence of this flag, the extended version of EVMS will proceed to unmap the snapshot without marking it for reset.

VirtuaLinux EVMS extension preserves snapshot correctness since the original volume is accessed in read-only mode by all nodes, and thus no snapshot can lose alignment with the original. One exception exists: major system upgrades, that are performed directly on the original copy of the file system trigger the reset of all snapshots.

At the implementation level, the VirtuaLinux EVMS extension requires the patching of EVMS user-space source code (actually just one line of C code). Overall, VirtuaLinux extends EVMS semantics. The extension covers a case in which general conditions that trigger a snapshot reset have been relaxed (avoids reset-on-activate) provided the original volume is not written. The extension ensures snapshot correctness. The described EVMS enables an original usage of the general snapshot technique.

2.4 Cluster Boot Basics

The described architecture has a unique permanent storage: the iSCSI attached SAN. Although booting from it (iBoot [8]) will be possible quite soon, iBoot is not currently supported by the majority of cluster vendors. VirtuaLinux is designed to enable the migration toward iBoot but it currently provides cluster boot through standard Intel PXE (Preboot Execution Environment [11]). The PXE protocol is approximately a combination of DHCP (Dynamic Host Configuration Protocol) and TFTP (Trivial File Transfer Protocol), albeit with some modifications to both. DHCP is used to locate the appropriate boot server or servers, while TFTP is used to download the initial bootstrap program and additional files.

VirtuaLinux uses an initial ram disk image (initrd) as bootstrap program. It includes the kernel of the selected OS as well as all the required software (both kernel modules and applications) to bring up the connection with the SAN via iSCSI in order to mount the root file system, which is stored in an EVMS-managed volume on the SAN itself.

As a result, the cluster can be booted by providing the nodes with a suitable initrd. Since PXE is a client-server protocol, the cluster should be provided with a PXE server functionality to seed initrd images during all stages of the cluster boot. Clusters usually rely on a master-slaves organization: a distinguished node of the cluster (master) provides to other nodes most of the cluster basic services, including DHCP and TFTP. These services enable network boot facility to other disk-less nodes. The master node solution cannot be adopted by VirtuaLinux (goal 2) since the master is clearly a single point of failure of the cluster configuration: any hardware or software problem on the master node may catastrophically disrupt cluster stability. At this end, the VirtuaLinux introduces the meta-master functionality that is supported cooperatively by all nodes of the cluster in such a way that the boot service can be guaranteed also in the case of crash of one or more nodes. Nevertheless, both during cluster boot and install a singular node acting as spark plug of the cluster start-up is unavoidable. To this end, the full booting of the cluster is achieved in three steps:

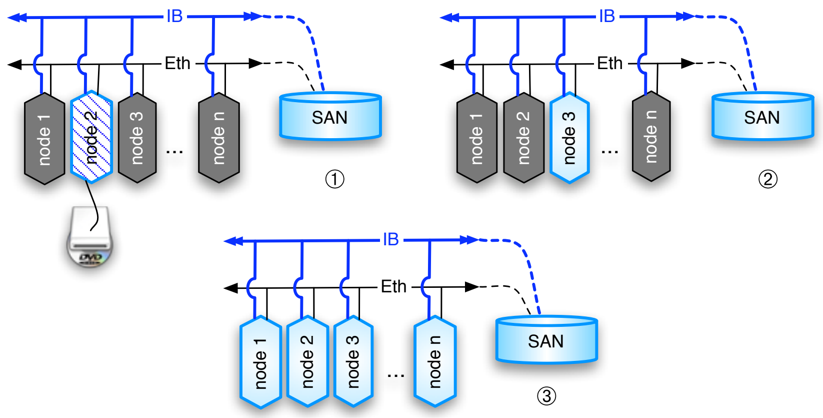

Figure 2: VirtuaLinux boot procedure.

① One of the nodes of the cluster is booted via a DVD reader loaded with VirtuaLinux. The booted OS is equipped with meta-functionality (and tools for SAN preparation and installation, which are described later in this guide).

② All nodes booted after the first inherit from the first some of the meta-master functionality; thus, the first node can be rebooted from the network (just detaching the external USB DVD reader).

③ At the end of the process, all nodes are uniformly configured, and each of them has inherited some of the meta-master functionality, and thus is able to provide at least TFTP service (as well as other above-mentioned services).

The detailed instructions to boot and install a cluster with VirtuaLinux are reported in Sec. C. Notice that the whole procedure, and in particular the live DVD, is largely independent of the installed OS, provided it is a flavor of Linux OS. In order to add yet another distribution to the already existing one it is enough to provide a valid initrd (that should be able to run iSCSI and EVMS) and a valid tarball of the root file system.

3 Master-less Cluster

In the previous section we discussed the importance of exploiting robust services with respect to the cluster boot (e.g. DHCP and TFTP). Since in master-based clusters almost all services are implemented in the master node, the service robustness issue naturally increases along the cluster run steady state. Those services are related to file sharing, time synchronization, user authentication, network routing, etc. VirtuaLinux avoids single point of failure in cluster operation by enforcing fault-tolerance of the master for all services (master-less cluster). Fault-tolerance is enforced by using two classical techniques: active replication and passive replication (primary-backup), targeting the categories of stateless and stateful services, respectively. Two additional categories should be added for completeness, i.e. the category of node-oriented and self-healing services.

- *Stateless services.* Active replication can be used whether the service can be configured to behave as a stateless service. Typically clients locate this kind of service by using broadcasted messages on the local network. A client request is non-deterministically served by the most reactive node.
- *Stateful services.* Passive replication is used when the service exists in a unique copy on the local network because of its authoritative or stateful nature. A client request usually specifies the service unique identifier (e.g. IP, MAC).
- *Node-oriented services.* A fail-stop of the node is catastrophic for this particular service, but has no impact on the usage of services on other nodes. No replication is needed in order to ensure cluster stability. Making these services reliable is outside the scope of VirtuaLinux, which ensures cluster availability at the OS level, not the availability of all services running on nodes.
- *Self-healing services.* These services adopt service-specific methods in order to guarantee fault-tolerance. These methods usually fall into the class of active or passive replication, which are implemented with service-specific protocols. Just services needed to ensure cluster stability are pre-configured in VirtuaLinux.

Since VirtuaLinux targets the uniform configuration of cluster nodes, active replication should be considered the preferred method. Active replication is realized by carefully configuring service behavior. Passive replication is realized by using a heartbeat fault-detector.

Service	Fault-tolerance	Notes
DHCP	active	Pre-defined map between IP and MAC
TFTP	active	all copies provide the same image
NTP	active	Pre-defined external NTPD fallback via GW
IB manager	active	Stateless service
DNS	active	Cache only DSN
LDAP	service-specific	Service-specific master redundancy
IP GW	passive	Heartbeat on 2 nodes with IP takeover (HA)
Mail	node-oriented	Local node and relays via DNS
SSH/SCP	node-oriented	Pre-defined keys
NFS	node-oriented	Pre-defined configuration
SMB/CIFS	node-oriented	Pre-defined configuration

4 Cluster Virtualization

Virtualizing the physical resources of a computing system to achieve improved degrees of sharing and utilization is a well-established concept that goes back decades [7, 20, 23]. Full virtualization of all system resources (including processors, memory and I/O devices) makes it possible to run multiple operating systems (OSes) on a single physical platform. In contrast to a non-virtualized system, in which a single OS is solely in control of all hardware platform resources, a virtualized system includes a new layer of software, called a Virtual Machine Monitor (VMM). The principal role of the VMM is to arbitrate access to the underlying physical host platform resources so that these resources can be shared among multiple OSes that are guests of the VMM. The VMM presents to each guest OS a set of virtual platform interfaces that constitute a Virtual Machine (VM).

By extension, a Virtual Cluster (VC) is a collection of VMs that are running on one or more physical nodes of a cluster and that are wired by one or more virtual private networks. By uniformity with the physical layer, all VMs are homogeneous, i.e. each VM may access a private virtual disk and all VMs of a virtual cluster run the same OS and may access a shared disk space. Different virtual clusters may coexist on the same physical cluster, but no direct relationship exists among them, apart from their concurrent access to the same resources (see Fig. 3). Virtual clusters bring considerable added value to the deployment of a production cluster because they ease a number of management problems, such as:

- *Physical cluster insulation.* Crashes or system instability due to administration mistakes or cursoriness at the virtual layer are not propagated down to the physical layer and have no security or stability impact on the physical layer. The relative usage of physical resources, such as processor, disk, memory, can be dynamically modulated and harmonized among different clusters. Two classes of system administrators are therefore introduced: the physical and virtual cluster administrator. For example, physical cluster management may be restricted to qualified/certified administrator, whereas the management rights of a virtual cluster can be given to virtual cluster owners who can change the configuration and install applications with no impact on underlying physical layer stability.

- *Cluster consolidation.* Virtualization is used to deploy multiple VCs, each exploiting a collection of VMs running an OS and associated services and applications. Therefore, the VMs of different VCs may be targeted to exploit a different OS and applications to meet different user needs. Installing and deploying new VCs, as well as reconfiguring and tuning existing VCs, does not affect the availability of services and applications running on the physical cluster and other VCs.

- *Cluster super scalability.* VCs may enable the efficient usage of the underlying hardware. A couple of cases are worth mentioning. First, the mul-

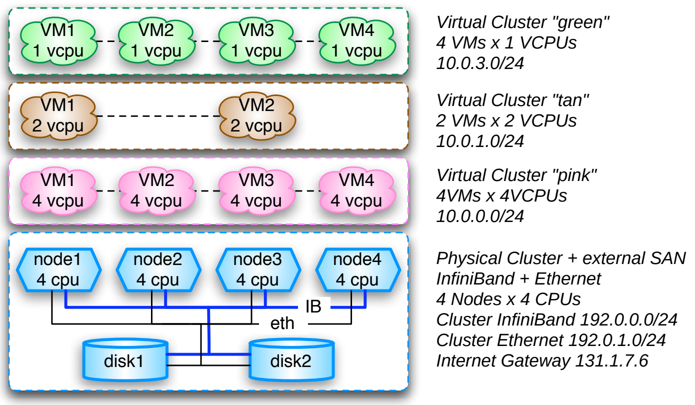

Figure 3: A physical cluster running three Virtual Clusters.

tiplexing of physical resources may induce a natural balancing of physical resource usage. This happens, for example, if a VC runs mostly I/O-bound services whereas another runs mostly CPU-bound applications. Second, applications and OS services can be tested with a parallelism degree that is far larger than the number of nodes physically available on the cluster.

The main drawback of virtualization is overhead, which usually grows with the extent of hardware and software layers that should be virtualized. Virtualization techniques are usually categorized in three main classes, in decreasing order of overhead: emulation, binary translation, and para-virtualization. At one end of the spectrum, a system can be emulated on top of a pre-packaged processor simulator[3], i.e. an interpreter running on a processor of kind A (e.g. x86) that fully simulates the same or another processor of kind B (e.g. PowerPC). A guest OS and related application can be installed and run on top of the emulator. Emulation overhead is usually extremely large; therefore the use of the technique is limited to addressing interoperability tasks.

Para-virtualization is at the other end of the spectrum [3]. Para-virtualization is something of a hack, as it makes the hosted OSes aware that they are in a virtualized environment, and modifies them so they will operate accordingly: some guest OS machine instructions are statically substituted by calls to the VMM. Virtualization mainly involves operations at OS kernel level, such as I/O devices arbitration, memory management, and processes scheduling. In this regard, it is not as much a complete virtualization as it is a cooperative relationship between VMM and guest OSes. Para-virtualization is further discussed in the next section.

[3]Simulation is not, in general, a synonym of emulation. Unlike simulation, which attempts to gather a great deal of run-time information as well as reproducing a program's behavior, emulation attempts to model to various degrees the state of the device being emulated. However, the contrast between the two terms has been blurred by the current usage, especially when referring to CPU emulation.

Binary translation is the middle ground [1]. It translates sequences of instructions from the source to the target instruction set. The translation may be performed both statically and dynamically. In the static binary translation an entire executable file is translated into an executable of the target architecture. This is very difficult to do correctly, since not all the code can be discovered by the translator. For example, some parts of the executable may be reachable only through indirect branches, whose value is only known at runtime. A dynamic binary translator is basically an interpreter equipped with a translated code caching facility. Binary translation differs from simple emulation by eliminating the emulator's main read-decode-execute loop (a major performance bottleneck), paying for this by large overhead during translation time. This overhead is hopefully amortized as translated code sequences are executed multiple times, even if it cannot be fully eliminated. More advanced dynamic translators employ dynamic recompilation: the translated code is instrumented to find out what portions are executed a large number of times, and these portions are optimized aggressively [4]. As a particular case, a binary code for a given architecture can be translated for the same architecture in order to dynamically detect guest OS machine instructions for I/O access and memory, processes scheduling and memory management. These instructions are somehow dynamically translated to trigger a VMM call: this may happen completely in software or with the help of a hardware facility, such as the recently introduced Intel VT-x (a.k.a. Intel Vanderpool) [17] support or AMD SVM (a.k.a. AMD Pacifica), discussed in Appendix A. With respect to para-virtualization, binary translation enables the execution of unmodified executable code, as for example a legacy guest OS such as Microsoft Windows. For this reason the majority of virtualization products currently available, such as QEMU [4], VMware [26], Apple Rosetta [2], use the binary translation technique.

4.1 Para-virtualization

In a traditional VMM the virtual hardware exposed is functionally identical to the underlying machine. Although full virtualization has the obvious benefit of allowing unmodified operating systems to be hosted, it also has a number of drawbacks. This has been particularly true for the prevalent IA-32, or x86, architecture, at least up to the introduction of the VT extension [17]. These architectures exploit the hierarchical protection domains (or protection rings) mechanism to protect data and functionality from faults (fault tolerance) and malicious behavior. This approach is diametrically opposite to that of capability-based security [14]. A protection ring is one of two or more hierarchical levels or layers of privilege within the architecture of a computer system. This is generally hardware-enforced by some CPU architectures, that provide different CPU modes at the firmware level. Rings are arranged in a hierarchy from most privileged (most trusted, usually numbered zero, or supervisor mode) to least privileged (least trusted, usually with the highest ring number). On most operating systems, Ring 0 is the level with the most privileges and interacts most directly with the physical hardware such as the CPU and mem-

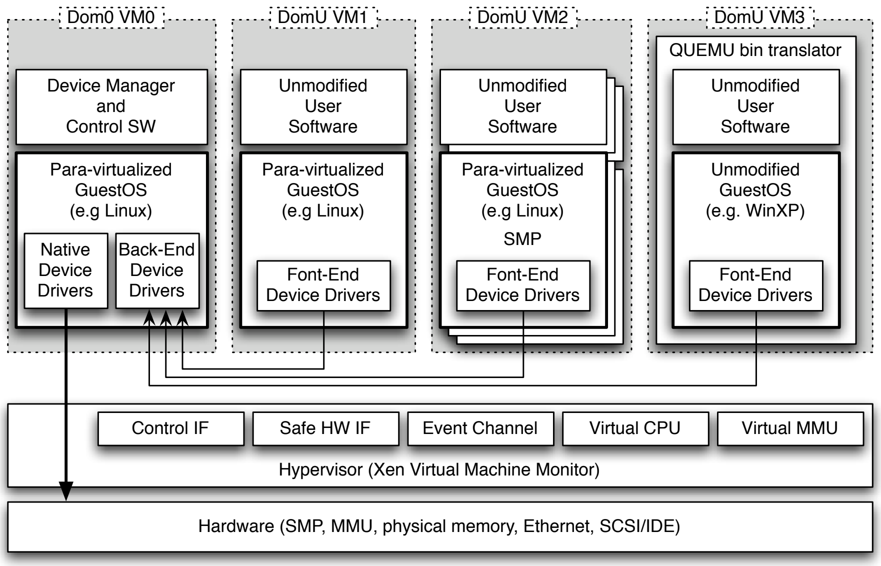

Figure 4: Xen architecture.

ory. Special gates between rings are provided to allow an outer ring to access an inner ring's resources in a predefined manner, as opposed to allowing arbitrary usage. Correctly gating access between rings can improve security by preventing programs from one ring or privilege level from misusing resources intended for programs in another.

To implement a VM system with the hierarchical protection model, certain supervisor instructions must be handled by the VMM for correct virtualization. Unfortunately, in pre-VT x86 architectures, executing these instructions with insufficient privilege fails silently rather than causing a convenient trap. Efficiently virtualizing the x86 MMU is also difficult. These problems can be solved by means of the mentioned binary translation technique, but only at the cost of increased complexity and reduced performance. VirtuaLinux adopts Xen as VMM to provide the user with para-virtualized VMs. Xen is an open-source VMM and currently supports IA-32, x86-64, IA-64 and PowerPC architectures [3].

4.2 Xen Architecture

Xen is a free source VMM for IA-32, x86-64, IA-64 and PowerPC architectures. It is para-virtualization software that runs on a host OS and allows one to run several guest OSes on top of the host on the same computer hardware at the same time. Modified versions of Linux and NetBSD can be used as hosts. Several modified Unix-like OSes may be employed as guest systems; on certain hardware (as x86-VT, see Appendix A), unmodified versions of Microsoft Windows and other proprietary operating systems can also be used as guest (via the QEMU binary translator) [27].

In the Xen terminology, the VMM is called *hypervisor*. A *domain* is a running VM within which a guest OS executes. The Domain0 (*Dom0*) is the first domain that is automatically started at boot time. Dom0 has permission to control all hardware on the system, and is used to manage the hypervisor and the other domains. All other domains are "unprivileged" (User Domains or *DomU*), i.e. domains with no special hardware access. Xen 3.0 architecture is sketched in Fig. 4, we refer back to the literature for an extensive description [3, 27].

5 VirtuaLinux Virtual Clustering

A Virtual Cluster (VC) is a cluster of VMs that can cooperate with a private virtual network, and that share physical resources of the same physical cluster. Different VCs behave as they were different physical clusters: i.e. they can cooperate one each other via standard networking mechanisms, but do not natively share any virtualized device nor user authentication mechanism. VirtuaLinux natively supports the dynamic creation and management of VCs by means of the following layers:

- *VM implementation,* i.e. the layer implementing the single VM. VirtuaLinux currently adopts the Xen Virtual Machine Monitor ([3], see also Sec. 4.1). The support of QEMU KVM kernel-based VMs is currently under evaluation for the next VirtuaLinux version [22].

- *VM aggregation,* i.e. the layer that aggregates many VMs in a VC, and dynamically creates and manages different VCs. This is realized via the *VirtuaLinux Virtual Cluster Manager* (VVCM), whose functionality is described in the rest of the section.

Overall, the VVCM enables the system administrator to dynamically create, destroy, suspend and resume from disk a number of VCs. The VCs are organized in a two-tier network: each node of a VC are connected to a private virtual network, and to the underlying physical network via a gateway node chosen in the VC. The gateway node of each VC is also reachable by nodes of all VCs. The nodes of a VC are homogeneous in terms of virtualized resources (e.g. memory size, number of CPUs, private disk size, etc.) and OS. Different clusters may exploit different configurations of virtual resources and different OSes. Running VCs share the physical resources according to a creation time mapping onto the physical cluster. VCs may be reallocated by means of the run-time migration of the VM between physical nodes[4].

[4]Migration of virtual nodes is currently an experimental feature, and it not included in the VVCM front-end.

Each virtual node of a VC is (currently) implemented by a Xen virtual machine (VM) that is configured at the VC creation time. Each virtual node includes:

- a virtual network interface with a private IP;
- a private virtual disk of configurable size;
- a private virtual swap area of configurable size;
- a VC-wide shared virtual storage.

The virtualization of devices is realized via the standard Xen virtualization mechanisms.

5.1 VC Networking

Xen supports VM networking via *virtualized Ethernet interfaces.* These interfaces can be connected to underlying physical network devices either via *bridged* or *routed* networking. Bridging basically copies data traffic among bridged interfaces at the data link layer (OSI model layer 2), thus bypassing any explicit routing decisions at higher layers, such as the IP layer (whereas routing between MAC addresses can still be performed). On the contrary, routing takes place at the OSI model layer 3, and thus is able to distinguish IP networks and to establish routing rules among them. On the one hand, bridging requires less setup complexity and connection tracking overhead as compared to the routing method. On the other hand, bridging impairs insulation among different networks on the same bridge, and it lacks flexibility since it can hardly be dynamically configured to reflect the dynamic creation and destruction of VC-private networks. For this, VirtuaLinux currently adopts the routed networking.

VirtuaLinux sets up VC-private networks in a simple yet efficient manner: all nodes in the VC are assigned addresses from a private network chosen at creation time, and the VC does not share the same subnet as the physical cluster, so the communications among physical and virtual clusters are handled by setting up appropriated routing policies on each physical node, which acts as a router for all the VMs running on it. Routing policies are dynamically set up at the deployment time of the VM. These ensure that:

- All VMs of all VCs can be reached from all physical nodes of the cluster.
- Each VC can access to the underlying physical network without any master gateway node. Internet can be accessed through physical cluster gateway (that is passively replicated).
- Therefore, all VMs are reachable each other. Virtual nodes of a VC are simply VMs on the same virtual subnet. However, each virtual network is insulated from the others. In particular, from within a VC is not possible to sniff the packets passing across virtual networks of other VCs.

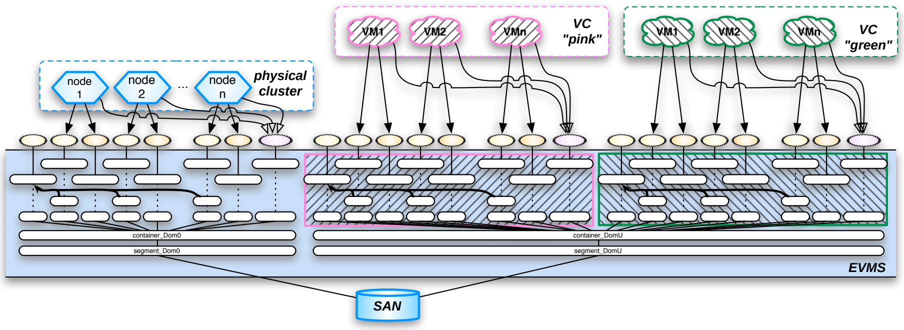

Figure 5: VirtuaLinux virtualized storage architecture.

- The routing configuration is dynamic, and has a VC lifespan. The configuration is dynamically updated in the case virtual nodes are re-mapped onto the physical cluster (see also 5.4).

VirtuaLinux virtual networks relies on the TCP/IP protocol on top of the Ethernet and Infiniband networks. In the latter case, the IP layer is implemented via the *IPoverIB* kernel module. Currently no user space drivers (Infiniband *verbs*) are available within VCs, whereas they are available at within the privileged domain (Dom0) as "native drivers" (see Fig. 4). User-space drivers enable direct access to the Infiniband API from the user application significantly boosting networking performance (see also MPI performance measures in Sec. 6). To the best of our knowledge, a version of user-space Infiniband verbs working within a Xen VM is currently under development, and it will be integrated in VirtuaLinux as soon as it will reach the "release candidate" quality [15].

5.2 VC Disk Virtualization

One of the most innovative features of VirtuaLinux concerns disk virtualization. Typically, VM-private disks are provided either via disk partitions or disk image files. The former method usually provides a speed edge while the latter guarantees a greater flexibility for dynamic creation of VMs. However, neither of them is suitable for supporting dynamically created VCs. As a matter of fact, both methods require the whole root file system of the host OS as many times as the number of nodes in the VC. This leads to an very high data replication on the physical disk, a very long VC creation time (for example the creation of a 10 nodes VC may require at least 40 GBytes), and a significant additional pressure on the network and the external SAN. VirtuaLinux copes with these issue by means of the EVMS snapshotting technique described in Sec. 2.3. As sketched in Fig. 5, the VirtuaLinux physical domain storage architecture (also see Fig. 1) is dynamically replicated at VC creation time. All private disks of a VC are obtained as snapshots of a single image including the VC guest OS.

As discussed in Sec. 2.3.1, this leads to a VC creation time that is independent of the number of nodes in the VC (in the range of seconds) and all benefit discussed in Sec. 2.3. Once created, EVMS volumes are dynamically mounted on physical nodes according to the virtual-to-physical mapping policy chosen for the given VC.

As for the physical cluster, each VC comes with its own VC-private shared storage, which relies on OCFS2 distributed file system to arbitrates concurrent read and write accesses from virtual cluster nodes. However, since Xen does not currently enable the sharing of disks between VMs on the same physical nodes, the VC shared disk cannot be directly accessed from within virtual nodes. VirtuaLinux currently overcomes the problem by wrapping the shared storage with a NFS file system. At VC deployment time, each physical node involved in the deployment mounts the VC shared storage, which is in turn virtualized and make available to virtual nodes.

5.3 VC Mapping and Deployment

VirtuaLinux provides two strategies for virtual-to-physical mapping of VMs:

- *Block.* Mapping aims to minimise the spread of VMs on the physical nodes. This is achieved by allocating on the physical node the maximum allowed number of VMs. For example, if we consider that the physical nodes are equipped with four cores, and the VC has been configured with one virtual node per core constraint, a VC consisting in 4 uniprocessor virtual nodes will be mapped and deployed on a single physical node.

- *Cyclic.* Try to spread the cluster's VM across all the cluster's physical nodes. For example, a virtual cluster consisting of 4 uniprocessor virtual nodes will be deployed on four different physical nodes.

The two strategies discussed can be coupled with the following modifiers:

1. *Strict.* The deployment can be done only if there are enough free cores.

2. *Free.* The constraint between the number of VM processors and physical cores is not taken into account at all.

Notice that the mapping strategy of a VC can be changed after the first deployment provided it is the suspended state.

5.4 VVCM: VirtuaLinux Virtual Cluster Manager

The VVCM consist of a collection of Python scripts to create and manage the VCs. These scripts use the following main components:

- A *database* used store all information about physical and virtual nodes. The database also stores all information about the mapping between physical and virtual nodes, the state of each virtual machine (if it is running or stopped). The information is maintained consistent between different

Dom0_IB	Ubuntu Dom0, Infiniband user-space verbs (MPI-gen2)
Dom0_IPoIB	Ubuntu Dom0, Infiniband IPoverIB (MPI-TCP)
Dom0_Geth	Ubuntu Dom0, Giga-Ethernet (MPI-TCP)
DomU_IPoIB	Ubuntu DomU, virtual net on top of Infiniband IPoverIB (MPI-TCP)

Table 1: Intel MBI experiments legend.

cluster launches. With simple scripts it is possible to retrieve from the database the current cluster running status and the physical mapping.

- A *command-line library* for the creation, the activation and the destruction of the VCs. In the current implementation the library is composed by a bunch of Python scripts that implement three main commands:
  1. *VC_Create* for the creation of the VC, it fills the database with VC-related static information.
  2. *VC_Control* able to launch a previously created VC and deploy it on the physical cluster according to a mapping policy. It is also able to stop, suspend or resume a VC.
  3. *VC_Destroy* that purges a VC from the system; it makes the clean-up of the database.
- A *communication layer* used for the staging and the execution of the VMs. The current implementation is build on top of the Secure Shell support (ssh).
- A *virtual cluster start-time support* able to dynamically configure the network topology and the routing policies on the physical nodes. The *VC_Control* command relies on these feature for VC start-up or shutdown.

6 Experiments

The implementation of VirtuaLinux has been recently completed and tested. We present here some preliminary experiments. The experimental platform consists of a 4U-case Eurotech cluster hosting 4 high-density blades, each of them equipped with a two dual-core AMD Opteron@2.2GHz and 8 GBytes of memory. Each blade has 2 Giga-Ethernets and one 10 Gbits/s Infiniband NIC (Mellanox InfiniBand HCA). The blades are connected with a (not managed) Infiniband switch. Experimental data has been collected on two installation of VirtuaLinux based on two different base Linux distributions:

- Ubuntu Edgy 6.10, Xen 3.0.1 VMM, Linux kernel 2.6.16 Dom0 *(Ub-Dom0)* and DomU *(Ub-DomU)*;
- CentOS 4.4, no VMM Linux kernel 2.6.9 *(CentOS)*.

Micro-benchmark	Unit	Ub-Dom0	Ub-DomU	CentOS
Simple syscall	usec	0.6305	0.6789	0.0822
Simple open/close	usec	5.0326	4.9424	3.7018
Select on 500 tcp fd's	usec	37.0604	37.0811	75.5373
Signal handler overhead	usec	2.5141	2.6822	1.1841
Protection fault	usec	1.0880	1.2352	0.3145
Pipe latency	usec	20.5622	12.5365	9.5663
Process fork+execve	usec	1211.4000	1092.2000	498.6364
float mul	nsec	1.8400	1.8400	1.8200
float div	nsec	8.0200	8.0300	9.6100
double mul	nsec	1.8400	1.8400	1.8300
double div	nsec	9.8800	9.8800	11.3300
RPC/udp latency localhost	usec	43.5155	29.9752	32.1614
RPC/tcp latency localhost	usec	55.0066	38.7324	40.8672
TCP/IP conn. to localhost	usec	73.7297	57.5417	55.9775
Pipe bandwidth	MB/s	592.3300	1448.7300	956.21

Micro-benchmark	Ub-Dom0 *vs* CentOS	Ub-DomU *vs* CentOS	Ub-DomU *vs* Ub-Dom0
Simple syscall	+667%	+726%	+7%
Simple open/close	+36%	+34%	-2%
Select on 500 tcp fd's	+51%	+51%	0%
Signal handler overhead	+112%	+127%	+7%
Protection fault	+246%	+293%	+13%
Pipe latency	+115%	+31%	-40%
Process fork+execve	+143%	+119%	-10%
float mul	~0%	~0%	~0%
float div	~0%	~0%	~0%
double mul	~0%	~0%	~0%
double div	~0%	~0%	~0%
RPC/udp latency localhost	+35%	-7%	-31%
RPC/tcp latency localhost	+35%	-5%	-30%
TCP/IP conn. to localhost	+32%	+3%	-22%
Pipe bandwidth	-38%	+51%	+144%

Table 2: VirtuaLinux: evaluation of ISA and OS performances with the LMbench micro-benchmark toolkit[16], and their differences (percentage).

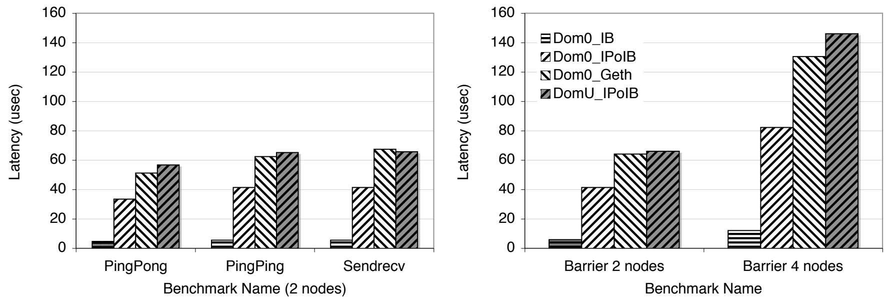

Figure 6: VirtuaLinux: evaluation of network communication latency with the Intel MBI Benchmarks [12].

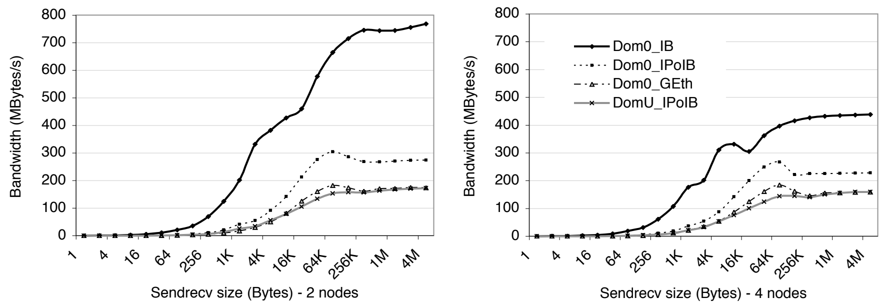

Figure 7: VirtuaLinux: evaluation of network bandwidth with the Intel MBI Benchmarks [12].

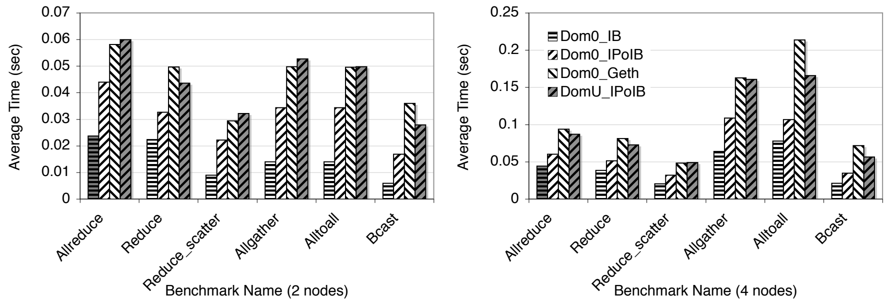

Figure 8: VirtuaLinux: evaluation of collective communication performance with the Intel MBI Benchmarks [12].

The experiments here reported are mainly focused to the evaluation of virtualization overhead with respect to two main aspects: guest OS primitives and networking (either virtualized or not). At this end, two sets of micro-benchmarks have been used: the *LMbench* benchmark suite [16], which has been used to evaluate the OS performance, and the *Intel MBI Benchmarks* [12] with *MVAPICH MPI* toolkit (mvapich2-0.9.8) [18], which has be used to evaluate networking performance.

Results of the LMbench suite are reported in Table 2. As expected, the virtualization of system calls has a non negligible cost: within both the privileged domain (Ub-Dom0) and user domain (Ub-DomU) a simple syscall pay a consistent overhead ($\sim +700\%$) with respect to the non-virtualized OS (CentOS) on the same hardware (while the difference between the privileged and the user domain is negligible). Other typical OS operations, such as fork+execve, exhibit a limited slowdown due to virtualization ($\sim +120\%$). However, as expected in a para-virtualized system, processor instructions exhibit almost no slowdown[5]. Overall, the OS virtualization overhead is likely to be amortized to a large extent in real business code. The evaluation of this extent for several classes of applications is currently ongoing.

The second class of experiments are related to networking. Fig. 6, Fig. 7, and Fig. 8 report an evaluation of the network latency, bandwidth and collective communications, respectively (the legend is reported in Table 1). Experiments highlight that the only configuration able to exploit Infiniband potentiality is the one using user-space Infiniband verbs (that are native drivers, see also Fig. 4). In this case, experiment figures are compliant with state-of-the-art performances reported in literature (and with CentOS installation, not reported here). Since native drivers bypass the VMM, virtualization introduces no overheads. As mentioned in Sec. 5.1, these drivers cannot be currently used within the VM (DomU), as they cannot be used to deploy standard Linux services, which are based on the TCP/IP protocol. At this aim, VirtuaLinux provides the TCP/IP stack on top of the Infiniband network (via the *IPoverIB*, or *IPoIB* kernel module). Experiments show that this additional layer is a major source of overhead (irrespectively of the virtualization layer): the TCP/IP stack on top of the 10 Gigabit Infiniband (*Dom0_IPoIB*) behave as a 2 Gigabit network. The performance of a standard Gigabit network is given as reference testbed (*Dom0_GEth*). Network performance is further slowed down by user domain driver decoupling that require data copy between front-end and back-end network drivers (see also Fig. 4). As result, as shown by *DomU_IPoIB* figures, VC virtual networks on top of a 10 Gigabit network, exhibits a Giga-Ethernet-like performances.

Extensive evaluation of disk abstraction layer performance is currently ongoing. Preliminary results show that VirtuaLinux succeeds to drawn around the 75% of raw throughput from an external SAN.

[5]A subset of benchmarks have been reported here. Excluded tests are coherent with ones reported in the table.

7 Conclusions

We have presented a coupled hardware-software approach based on open source software aiming at the following goals:

1. Avoiding fragility due to the presence of disks on the blades by removing disks from blades (high-density disk-less cluster) and replacing them with a set of storage volumes. These are abstract disk implemented via an external SAN, which is accessed by nodes of the cluster through the iSCSI protocol, and is abstracted out through EVMS, which enables the flexible and dynamic partitioning of the SAN.

2. Avoiding single point of failure by removing master from the cluster. Master node features, i.e. the set of services implemented by the master node, are categorized and made redundant by either active or passive replication in such a way they are, at each moment, cooperatively implemented by the running nodes.

3. Improving management flexibility and configuration error resilience by means of transparent node virtualization. A physical cluster may support one or more virtual clusters (i.e. cluster of virtual nodes) that can be independently managed and this can be done with no impact on the underlying physical cluster configuration and stability. Virtual clusters run a guest OS (either a flavor of Linux or Microsoft Windows) that may differ from host OS, governing physical cluster activities. Xen and QEMU are used to provide the user with both cluster para-virtualization and hardware-assisted binary translation.

These goals are achieved independently through solutions that have been designed to be coupled, thus to be selectively adopted. A suite of tools, called VirtuaLinux, enables the boot, the installation, the configuration and the maintenance of a cluster exhibiting the previously described features. VirtuaLinux is currently targeted to AMD/Intel x86 64-based nodes, and includes:

- Several Linux distributions, currently Ubuntu Edgy 6.10 and CentOS 4.4.
- An install facility able to install and configure included distributions according to goals 1-3, and easily expandable to other Linux distributions and versions.
- A recovery facility able to reset a misconfigured node.
- A toolkit to manage virtual clusters (VVCM) and one or more pre-configured virtual cluster images (currently Ubuntu Edgy 6.10 and CentOS 4.4).

VirtuaLinux is an open source software under GPL. It is already tested and available on Eurotech HPC platforms. In this regard, Eurotech laboratory experienced a tenfold drop of clusters installation and configuration time. To the best of our knowledge few existing OS distributions achieve the described goals, and none achieve all of them. VirtuaLinux introduces a novel disk abstraction layer, which is the cornerstone of several VirtuaLinux features, such as the time and space efficient implementation of virtual clustering. Preliminary experiments show that VirtuaLinux exhibits a reasonable efficiency, which will naturally grow with virtualization technology evolution.

Acknowledgements and Credits

VirtuaLinux has been developed at the HPC laboratory of Computer Science Department of the University of Pisa and Eurotech HPC, a division of Eurotech Group. VirtuaLinux project has been supported by the initiatives of the LITBIO Consortium, founded within FIRB 2003 grant by MIUR, Italy. VirtuaLinux is a open source software under GPL available at `http://virtualinux.sourceforge.net/`.

We are grateful to Gianmarco Spinatelli and Francesco Polzella who contributed to VirtuaLinux development, and to Peter Kilpatrick for his help in improving the presentation.

A VT-x: Enabling Virtualization in the x86 ISA

Virtualization was once confined to specialized, proprietary, high-end server and mainframe systems. It is now becoming more broadly available and is supported in off-the-shelf systems. Unfortunately, the most popular recent processor architectures, e.g. x86 Intel Architecture prior to the VT extension, are not designed to support virtualization [17]. In particular, the x86 processor exhibits a peculiar implementation of processor protection domains (rings).

Conceptually rings are a way to divide a system into privilege levels in such a way it is possible to have an OS running in a level that a user's program cannot modify. This way, if an user program goes wild, it will not crash the system, and the OS can take control, shutting down the offending program cleanly: rings enforce control over various parts of the system. There are four rings in the x86 architecture: 0, 1, 2, and 3, with the lower numbers being higher privilege. A simple way to think about it is that a program running at a given ring cannot change things running at a lower numbered ring, but something running at a low ring can interfere with a higher numbered ring[6].

In practice, only rings 0 and 3, the highest and lowest, are commonly used. OSes typically run in ring 0 while user programs are in ring 3. One of the ways the 64-bit extensions to x86 "clean up" the Instruction Set Architecture (ISA) is by losing the middle rings, i.e. 1 and 2. Mostly no one cares that they are gone, except the virtualization people. The VMM obviously have to run in ring 0, but if they want to maintain complete control, they need to keep the host OS (i.e. the OS running on top of the VM) out of ring 0. If a runaway task can overwrite the VM, it negates the reason protection rings exist. The obvious solution is to force the hosted OS to run in a higher ring, like ring 1. This would be fine except that the OSes are used to running in ring 0, and having complete control of the system. They are set to go from 0 to 3, not 1 to 3. In a para-virtualized environment, the OS has been changed to operate correctly with it. This solution is clearly not transparent with respect to hosted OSes.

The problem here is that some instructions of x86 will only work if they are going to or from ring 0, and others will behave oddly if not in the right ring. These odd behaviours mainly spring from an extremely weak design of x86 ISA. We mention some of those for the sake of exemplification, whereas we refer the reader to literature for the full details [17]. One of the most curious, is that some privileged instructions do not fault when executed outside ring 0 (they silently fail). Therefore, the classical approach of trapping the call issued by the host OS, and serving it within the VMM, simply, will not work.

VT-x augments x86 ISA with two new forms of CPU operation: VMX root operation and VMX non-root operation. VMX root operation is intended for use by a VMM, and its behavior is very similar to that of x86 ISA without VT-x. VMX non-root operation provides an alternative x86 ISA environment controlled by a VMM and designed to support a VM. Both forms of operation support all four privilege levels, allowing guest software to run at its intended

[6]The underlying assumption here (not fully sound nor legitimate) is that code running in a low ring, such as the OS, will behave correctly.

privilege level, and providing a VMM with the flexibility to use multiple privilege levels.

VT-x defines two new transitions: a transition from VMX root operation to VMX non-root operation is called a VM entry, and a transition from VMX non-root operation to VMX root operation is called a VM exit. VM entries and VM exits are managed by a new data structure called the virtual-machine control structure (VMCS). The VMCS includes a guest-state area and a host-state area, each of which contains fields corresponding to different components of processor state. VM entries load processor state from the guest-state area. VM exits save processor state to the guest-state area and then load processor state from the host-state area.

Processor operation is changed substantially in VMX non-root operation. The most important change is that many instructions and events cause VM exits. Some instructions (e.g., INVD) cause VM exits unconditionally and thus can never be executed in VMX non-root operation. Other instructions (e.g., INVLPG) and all events can be configured to do so conditionally using VM-execution control fields in the VMCS [17].

B Eurotech HPC Solutions

Highly scalable, Eurotech HPC solutions provide turn-key systems for a large number of applications. The T-Racx technology provides more than 2TFlops per rack using the latest Intel Xeon and AMD Opteron processors. Each T-Racx model integrates Infiniband connectivity to provide in a single infrastructure very low latency communications and large bandwidth with direct IB storage access. T-Racx solutions integrate an elegant Italian Design by Linea Guida with a powerful air cooling and a power distribution system that enable even the most dense and demanding installations.

C VirtuaLinux-1.0.6 User Manual

This section focuses on the basic installation and configuration of VirtuaLinux version 1.0.6 meta-distribution of the Linux operating system. It is targeted to system administrators familiar with cluster computing and moderately familiar with administering and managing servers, storage, and networks. General concepts and design of VirtuaLinux are covered in the previous sections, while advanced configuration is covered in the software man pages and info files. VirtuaLinux has been extensively tested on Eurotech clusters described in Sec. 6, however VirtuaLinux can be installed in any homogeneous cluster equipped with an external SAN that exports an iSCSI disk. Notice that iSCSI disk setup procedure is not part of VirtuaLinux.

C.1 VirtuaLinux Pre-requisites

VirtuaLinux DVD enables the seamless installation, configuration and recovery of a Linux-based fault-tolerant cluster platform. Target platforms are disk-less clusters equipped with an external SAN meeting the following requirements:

- Nodes (blades, boards) of the cluster should be networked with at least one Ethernet switch (boot network). Ethernet and Infiniband are supported as additional networks.
- Nodes should be homogeneous and based on either AMD or Intel x86 architecture.
- The external SAN should provide iSCSI connectivity, on either the Ethernet or Infiniband NIC. It exposes a known static IP. The SAN is supposed to autonomously exploit fail-safe features. All other disks despite the SAN (e.g. blades onboard disks) are ignored during the installation process.
- The cluster should be equipped with an internal or external USB DVD reader.
- Eurotech clusters are the only tested hardware.
- VirtuaLinux DVD installs a homogeneous software configuration on all nodes of the cluster (master-less cluster). All installed services are made fault-tolerant via either active or passive replication technique. Therefore, once installed, the cluster does not exhibit single point of failure due to nodes hardware or software crashes and malfunctions.

The VirtuaLinux DVD provides the user with three facilities:

1. Cluster boot.
2. Cluster set up (configuration and installation).
3. Cluster or node restore.

Name	Meaning[default value]	
storage_ip	IP address of the external SAN [192.168.0.252]	**required**
storage_port	port assigned to the iSCSI service on storage_ip [3260]	**required**
sif	network interface for the external SAN [eth0 | ib0]	**required**
sifip	IP address to be assigned to sif [192.168.0.254]	optional
sifnm	netmask to be assigned to sif [255.255.255.0]	optional
sifm	kernel modules needed to bring up sif [tg3 | ib_xxx]	optional
ivid	iSCSI exported disk name [ ]	optional
root	root device [/dev/evms/defaultcontainername]	**required**

Table 3: Boot parameters description.

These features rely on a highly cooperative behavior of nodes starting from early stages of first boot. The cooperative behavior is apt to eliminate the single point of failures during both cluster set up and ordinary work.

C.2 Physical cluster

C.2.1 Cluster Boot

The boot feature enables the boot from VirtuaLinux DVD of a node of the cluster whether no other nodes are already running. This feature should be used to start the boot sequence since the cluster cannot boot via iSCSI from an external SAN. Assuming the cluster is switched off, proceed as follow:

1. Connect a DVD reader to one of the nodes of the cluster and load the reader with the VirtuaLinux DVD.

2. Switch on cluster main power supply, any external network switch, and the external SAN (that should be connected to either Ethernet or Infiniband switch of the cluster).

3. Switch on the node that is connected to the DVD reader. The node BIOS should be configured to boot from DVD reader.

4. The selected node will boot from the VirtuaLinux DVD. A menu will appear on the screen: select Boot Eurotech Cluster, check all boot parameter are properly configured, then press Enter. Boot parameters depend on how the external SAN is connected with the cluster, and they can be changed by pressing F6 key. Parameters meaning is described in Table 3.

5. After a little while a debug console will appear. Then all other nodes can be simply switched on (in open order). As soon as at least another node had completed the boot, the DVD reader should be detached from the cluster and first node can be rebooted (typing Ctrl-D on the debug console).

param_list	::=	*parameter*=*value* *parameter_list* | `sifm=`*value_list* *param_list* | *parameter*=*value* `---`
parameter	::=	*storage_ip* | *storage_port* | `sif` | `sifip` | `sifnm` | `ivid` | `root`
value_list	::=	*value,value_list* | *value*
value	::=	⟨ *ASCII string* ⟩

Table 4: Boot screen: command line syntax.

The command line syntax (in the Extended Backus-Naur form) is shown in Table 4. In the case the same parameter appear twice or more, just the first occurrence will be taken in account. Two use cases are given in the following, exemplifying the boot with a Ethernet and Infiniband storage, respectively.

Ethernet Let us suppose the external SAN is connected to the cluster via a Ethernet network, implemented on cluster end through an e1000 board, while SAN iSCSI IP is `192.168.0.200:3260`, and during install procedure the name foo has been chosen as EVMS root segment name. In this case, the parameter should be adjusted as follow:

```
... root=/dev/evms/defaultfoo storage_ip=192.168.0.200 sifm=e1000 ... ---
```

Notice that only parameters that differ from their default value are shown here. The pre-defined command line includes commonly used parameters at their default values, which can be changed by the user, but not deleted. The command line should always be terminated by three hyphens (`---`).

Infiniband Let us suppose the external SAN is connected to the cluster via an Infiniband network by means of `IPoverIB` protocol, while SAN iSCSI IP is `192.168.1.2:3260` with netmask `255.255.0.0`, and during install procedure the name bar as been chosen has EVMS root segment name. In this case, the parameter should be adjusted as follow:

```
... root=/dev/evms/defaultbar sif=ib0 sifnm=255.255.0.0 storage_ip=192.168.1.2 ---
```

C.2.2 Cluster Setup (Configuration and Installation)

The cluster set up feature enables to configure VirtuaLinux for a given cluster and install it on an external SAN. The VirtuaLinux essentially consists in a Linux OS tailored and pre-configured for fault-tolerant cluster platforms. Currently, the supported base Linux distributions are either Ubuntu 6.10 or CentOS 4.4. The user is asked to choose which base distribution to install in all nodes of the cluster. These are base distribution main features:

Ubuntu Edgy 6.10 Ubuntu Edgy distribution is based on kernel 2.6.16-xen. It supports OS para-virtualization of Xen 3.0.1 virtual machines. In addition to common services (described in the next sections), the distribution is ready to launch Xen virtual machines. Also, it provides the system administrator with a number of tools for the management of virtual clusters, which in turn can be installed with other operating systems, such as Linux-Ubuntu, Linux-CentOS (Microsoft Windows XP on some cluster platforms). Virtual clustering and its management are described later in this guide.

CentOS 4.4 CentOS distribution is based on kernel 2.6.9. It does not supports OS para-virtualization.

Common services Several services are pre-installed on both flavors of VirtuaLinux, among the others Gateway, NTP, DHCP, TFTP, SSH, LDAP, NSCD, iSCSI, OpenFabrics network manager.

Configuration procedure Assuming the cluster is switched off, proceed as follow:

1. Connect a DVD reader to one of the nodes of the cluster and load the reader with the VirtuaLinux DVD. The chosen node will become the primary gateway node to the external network.

2. Switch on cluster main power supply, any external network switch, and the external SAN (that should be connected to either Ethernet or Infiniband switch of the cluster).

3. Switch on the node that is connected to the DVD reader. The node BIOS should be configured to boot from DVD reader.

4. The selected node will boot from the DVD. A menu will appear on the screen: select Install Eurotech Cluster.

5. A live version of VirtuaLinux will come up on the node (it require several minutes); a Gnome-based GUI will start automatically.

6. Launch install tool by double-clicking on its icon placed on the Gnome desktop. A number of question should be answered to proceed:

   - *Distribution:*

     Base distribution Either Ubuntu or CentOS

   - *External SAN configuration:*

     IP It should be preferably kept in the range of `xxx.yyy.zzz.[250-252]` in order to avoid possible conflicts with IP of cluster nodes.

     Netmask `[255.255.255.0]`

     Port `[3260]`

Meta-master-IP `[xxx.yyy.zzz.253]` Ensure there are no conflicts with the storage IP before changing it

Connection-kind Either Ethernet or Infiniband. Asked only if both network are plugged.

- *Cluster HW configuration:*

  Number-of-nodes

- Target storage. The iSCSI SAN is activated and scanned:

  iSCSI-target All disks exported by the iSCSI SAN target are listed. One of them should be selected for the installation.

- *Target storage virtualization.* The iSCSI target is managed via EVMS (Enterprise Volume Management System), which enables to expose to the cluster a number independent volumes, i.e. of logical partitions of one o more iSCSI connected SANs. The parameters related to storage virtual organization are the following:

  volume-size `[5120]` Size in MBytes of each node volume.

  swap-size `[2048]` Size in MBytes of each node swap volume.

  shared-size `[10240]` Size in MBytes of the volume shared among nodes of the cluster.

  free-space-allocation A list of free space segments on iSCSI-target disks are listed. One of them should be selected to hold a container (note that it is possible to build a container over the concatenation of more free space segments, even if the features is not currently supported).

  container-name `[ubuntu/centos]` Different installation on the same SAN should have different container names.

  shared-fs-id `[sharedfs]` identifier of the shared file system. Advanced configuration switch, it is safe to leave it at the default value.

  The volumes will be formatted with the ext3 file system. The shared volume will be formatted with OCFS2 (Oracle Cluster File System) in the case of Ubuntu, or with GFS (Global Files System) in the case of CentOS. Both OCFS2 and GFS support concurrent accesses and distributed locks. GFS does not support Memory Mapped I/O.

- *Gateway configuration.* One of the nodes of the cluster may act as gateway between cluster local networks and external network (e.g. Internet). Currently, the external network is supposed to be an Ethernet. The gateway will be configured in primary-backup fail-over mode with a backup node, thus if cluster-name1 stops working for any reason, cluster-name2 will inherit gateway functionality (cluster routing tables are automatically managed to achieve fail-over transparency). The mechanism is implemented through heartbeat (Linux HA), and needs a spare IP address on the local network. Parameters related to gateway configuration are the following:

primary-gateway-local-IP `[xxx.yyy.zzz.1]` IP of the primary gateway node in the local network.

backup-gateway-local-IP `[xxx.yyy.zzz.2]` IP of the backup gateway node in the local network.

spare-local-IP `[xxx.yyy.zzz.253]` Spare IP in the local network.

public-IP `[XXX.YYY.ZZZ.WWW]` IP assigned to the gateway node in the external network.

public-netmask `[255.255.255.0]` External network netmask.

public-router `[aaa.bbb.ccc.ddd]` External network router IP.

public-DNS `[AAA.BBB.CCC.DDD]` External network DNS.

domain `[eurotech.com]` Default domain name.

At this point, an automatic procedure to recognize the interfaces for the external and boot networks is started. The procedure will ask the user to approve the detected interfaces. In order to detect the boot interface, the procedure will ask the user to switch on a random node of the cluster.

confirm-ext-if `[YES/no]` External network interface.

confirm-boot-if `[YES/no]` Boot network interface.

- *Network configuration.* If the network used for the storage is also used for the network boot (that should be an Ethernet), the following parameters are required:

  first-node-IP-netboot-netstorage `[calculated address]`

  Otherwise, if two different networks are used for the storage and the network boot (e.g. Infiniband and Ethernet, or two different Ethernets), the following parameters are required:

  first-node-IP-netboot `[calculated address]`

  netmask-netboot `[calculated netmask]`

  first-node-IP-netstorage `[calculated address]`

  The calculated suggested default values represent a safe choice.

- *Nodes configuration.*

  cluster-hostname The string will be used assign nodes hostnames; nodes will be called cluster-hostnameX, where X is the node ordinal number collected during MAC addresses collection.

- *Gathering of MAC addresses.*

  MAC-gather-procedure `[1 automatic 2 manual 3 from-file]`

  `1 automatic` The procedure to gather nodes MAC addresses will automatically start. The user is required to follow instructions on the installation console that consist in repeat the following procedure for each node of the cluster (except the one where is currently running the installation procedure):

```
for each node
  Switch the node on;
  Wait the node is sensed by the meta-master (20-40 secs);
  Switch the node off;
```

2 `manual` MAC addresses of nodes are iteratively queried from the user console.

3 `from-file` Not yet implemented.

7. The data gathering is almost completed. The installation procedure proceeds to prepare the external storage (EVMS volumes creations and volumes formatting) and copy all needed data onto the proper volumes. This operation may require several minutes. A number of configuration files will be generated and stored in `/etc/install/INSTALL.info` file.

8. As last step, the installation procedure asks a password for cluster root user, that will trigger the last configuration step of the cluster.

9. After the completion of the installation, the node used fro the installation should be rebooted. Notice the DVD reader should not be disconnected since it is required to boot the cluster. The boot procedure is described in Cluster boot section of this guide. Notice, some of the data entered during install procedure are required to boot the cluster.

C.2.3 Cluster or Node Restore

Cluster set up consists of two successive steps: cluster configuration and installation. After the cluster set up, configuration data (collected during cluster set up) is saved in a predefined file (`/etc/install/INSTALL.info`). Cluster restore feature enables to reset either the whole cluster or a node to the original status (set up time status) by using configuration data. The feature helps in solving two typical problems of cluster management:

- *Reset.* A cluster can be fully reset to original status (at the time of installation).

- *Restore.* The configuration or the root file system of some nodes of the cluster has been corrupted due to administration mistakes.

The file `INSTALL.info` produced during installation should be available in a known path of the files system. In the case of full reset of the cluster (commands 2), the boot from VirtuaLinux DVD is required. The node reset can be performed by booting from DVD or from another running node in the same cluster. Restore options are:

`-f path` Path of the file `INSTALL.info` (compulsory argument).

`-w path` Path of a temporary directory `[/tmp]`.

`-i` Toggle interactive mode.

`-d n` Where `n` is the command to be executed:

- `1` Rebuild configuration files of all nodes. Disks are not formatted, software installed on nodes after the installation will be preserved.
- `2` Fully reset all nodes. Disks are reset to installation time status, software installed on nodes after the installation will be lost.
- `3` Reset a single node (to be used with `-n num`).

`-s` Toggle formatting of the shared volume.

`-F` Force the reallocation of EVMS segment.

`-n num` Node number selected for the restore (to be used with the option `-d 3`).

`-v level` Verbosity level (1=low, 2=high).

Examples:

Reset all nodes:

```
root@node1:# cd /etc/install/scripts
root@node1:# ./restore.sh -f /etc/install -d 2
```

Reset node 3:

```
root@node1:# cd /etc/install/scripts;
root@node1:# ./restore.sh -f /etc/install -d 3 -n 3
```

C.3 Cluster Virtualization

DomU installation procedure is still under refinement. This documentation reports installation steps required by Eurotech HPC version 1.0.6. These step will be further simplified in the next Eurotech HPC version. The VC management system requires a setup step, that should be performed from the live DVD. The best way to install cluster virtualization software is to perform these steps at the cluster installation time, just after the VirtuaLinux installation (before booting the cluster).

C.3.1 Installation

Run as root user the script setup.sh in the

```
/install_packages/VirtuaLinux/install_domU
```

directory to set up the VC management system. Note that all the scripts located in the directory mentioned above must be executed as root user from the directory itself. The setup.sh needs two parameters in input: a node number and a port. They are relative to a postgres database service that will be used to store VCs information. For example, if we want to use as database server the second node on port 5432 we have to start the setup.sh script as following:

```
root@node1:/shared/VC/domU_scripts#./setup.sh -n 2 -p 5432
```

At the termination of the execution the following tasks should have been accomplished:

- initialization of directories where VC configuration files are stored (`/shared/VC`).
- installation of the python objects needed to get the system working.
- initialization of the database.

At the end of DomU installation we have in the `/shared/VC/domU_scripts` directory the scripts that have to be used to manage VCs.
In the `/shared/VC/os_images_repository` directory we can find the DomU images in tgz format.

C.3.2 Tools

To easily manage VCs you can use the tools located in `/shared/VC/domU_scripts`. Such tools are:

`VC_Create` Enable the user create a new VC and deploys it on the physical cluster. The deployment is accomplished in respect of the selected allocation policies. In the case nodes of the cluster are equipped with both an Infiniband network and an Ethernet, the Infiniband network address should be used.

`VC_Control` {`start` | `stop` | `save` | `restore`} Start/stop/save/restore a VC.

`VC_Destroy` Definitively destroy the VC.

`VC_Create` Create a new VC in an interactive manner.

```
root@node1:/shared/VC/domU_scripts#./VC_Create.py -b 192.168.10.2
-p 5432
```

The script starts querying the user about the VC characteristics. The options of the command are:

`-b db IP`

`-p db port`

Note that if Infiniband network is present and configured we have to use the Infiniband `IP` in the `-b` option. The complete list of options are available via on line help.

`VC_Control` Once a VC has been created by `VC_Create` use `VC_Control` to perform the basic actions on it. This script takes in input a list of options, a VC name and an action chosen among:

`start` Start the specified VC. The VMs are created on the physical nodes according to the mapping defined during the deployment process.

`stop` Stop the specified VC.

`save` The state of each virtual machine of the cluster is saved onto secondary storage.

`restore` The state of the cluster is restored from an image previously created by a save operation.

In the `VC_Create` command you have to choose the deployment strategy of the virtual nodes on the physical one. In particular, it is possible to deploy the VC according to two different strategies:

`block` Mapping aims to minimise the spread of VMs on the physical nodes. This is achieved by allocating on the physical node the maximum allowed number of VMs. For example, if we consider that the physical nodes are equipped with four cores, and the VC has been configured with one virtual node per core constraint, a VC consisting in 4 uniprocessor virtual nodes will be mapped and deployed on a single physical node.

`cyclic` Try to spread the cluster's VM across all the cluster's physical nodes. For example, a virtual cluster consisting of 4 uniprocessor virtual nodes will be deployed on four different physical nodes.

The two strategies briefly discussed above can behave in two slightly different ways:

`strict` The deployment can be done only if there are enough free cores.

`free` The constraint between the number of VM processors and physical cores is not taken into account at all.

Example:

```
root@node1:/shared/VC/domU_scripts#./VC_Control.py -b 192.168.10.2
-p 5432 -t cyclic -m strict virtualCluster1 start
```

The command in the above example deploy the VC named “virtualCluster1” using the `cyclic` policy and the `strict` constraint. Using the option `-l` a list of available clusters is returned. The complete list of options are available via on line help.

`VC_Destroy` Definitively destroy a VC including configuration files and disk partitions. It takes in input a list of options and a virtual cluster name.

Example:

```
root@node1:/shared/VC/domU_scripts#./VC_Destroy.py -b 192.168.10.2
-p 5432 virtualCluster1
```

The complete list of options are available via on line help.

References

[1] K. Adams and O. Agesen. A comparison of software and hardware techniques for x86 virtualization. In *ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, pages 2–13, New York, NY, USA, 2006. ACM Press.

[2] Apple Inc. *Universal Binary Programming Guidelines, Second Edition*, Jan. 2007. `http://developer.apple.com/documentation/MacOSX/Conceptual/universal_binary/universal_binary.pdf`.

[3] 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 *Proc. of the 9th ACM Symposium on Operating Systems Principles (SOSP'03)*, pages 164–177. ACM Press, 2003.

[4] F. Bellard. QEMU, a fast and portable dynamic translator. In *USENIX 2005 Annual Technical Conference, FREENIX Track*, Anaheim, CA, Apr. 2005.

[5] Eurotech HPC, a division of Eurotech S.p.A. *System and components for pervasive and high performance computing*, 2007. `http://www.exadron.com`.

[6] EVMS website. *Enterprise Volume Management System*, 2007. `http://evms.sourceforge.net/`.

[7] R. P. Goldberg. Survey of virtual machine research. *Computer*, pages 34–45, June 1974.

[8] IBM. *iBoot - Remote Boot over iSCSI*. IBM, 2007. `http://www.haifa.il.ibm.com/projects/storage/iboot/index.html`.

[9] IBM. *Understanding and exploiting snapshot technology for data protection*, 2007. `http://www-128.ibm.com/developerworks/tivoli/library/t-snaptsm1/index.html`.

[10] The Infiniband Consortium. *The Infiniband trade association: consortium for Infiniband specification*, 2007. `http://www.infinibandta.org/`.

[11] Intel Corporation. *Preboot Execution Environment (PXE) Specification*, 1999. `http://www.pix.net/software/pxeboot/archive/pxespec.pdf`.

[12] Intel Corporation. *Intel MPI Benchmarks: Users Guide and Methodology Description*, ver. 3.0 edition, 2007. `http://www.intel.com/cd/software/products/asmo-na/eng/cluster/clustertoolkit/219848.htm`.

[13] iSCSI Specification. *RFC 3720: The Internet Small Computer Systems Interface (iSCSI)*, 2003. `http://tools.ietf.org/html/rfc3720`.

[14] T. A. Linden. Operating system structures to support security and reliable software. *ACM Comput. Surv.*, 8(4):409–445, 1976.

[15] J. Liu, W. Huang, B. Abali, and D. K. Panda. High performance VMM-Bypass I/O in virtual machines. In *USENIX 2006 Annual Technical Conference*, Boston, MA, June 2006. `http://www.usenix.org/events/usenix06/tech/full_papers/liu/liu_html/index.html`.

[16] L. McVoy and C. Staelin. *LMbench: Tools for Performance Analysis*, ver. 3.0 edition, Apr. 2007. `http://sourceforge.net/projects/lmbench/`.

[17] G. Neiger, A. Santoni, F. Leung, D. Rodgers, and R. Uhlig. Intel virtualization technology: Hardware support for efficient processor virtualization. *Intel Technology Journal*, 10(3):166–178, Aug. 2006.

[18] The Ohio State University. *MVAPICH: MPI over InfiniBand and iWARP*, 2007. `http://mvapich.cse.ohio-state.edu/overview/mvapich2/`.

[19] E. Pinheiro, W.-D. Weber, and L. A. Barroso. Failure trends in a large disk drive population. In *Proc. of the 5th USENIX Conference on File and Storage Technologies (FAST'07)*, pages 17–28, San Jose, CA, USA, Feb. 2007.

[20] G. J. Popek and R. P. Goldberg. Formal requirements for virtualizable third generation architectures. *CACM*, 17(7):412–421, 1974.

[21] S. Pratt. EVMS: A common framework for volume management. In *Ottawa Linux Symposium*, 2002. `http://evms.sourceforge.net/presentations/evms-ols-2002.pdf`.

[22] Qumranet Inc. *KVM: Kernel-based Virtual Machine for Linux*, June 2007. `http://kvm.qumranet.com/kvmwiki`.

[23] M. Rosenblum. The reincarnation of virtual machines. *Queue*, 2(5):34–40, 2005.

[24] B. Schroeder and G. A. Gibson. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? In *Proc. of the 5th USENIX Conference on File and Storage Technologies (FAST'07)*, pages 1–16, San Jose, CA, USA, Feb. 2007.

[25] Sun Microsystems. *Sun Grid Engine*, 2007. `http://gridengine.sunsource.net/`.

[26] VMware Inc. *VMware website*, 2007. `http://www.vmware.com/`.

[27] Xen Source. *Xen wiki*, 2007. `http://wiki.xensource.com/`.