1

Virtualization

In this chapter, we introduce virtualization, which is at the root of the revolution in the networking world, as it involves constructing software networks to replace hardware networks.

Figure 1.1 shows the process of virtualization. We simply need to write a code that performs exactly the same function as the hardware component. With only a few exceptions, which we will explore later on, all hardware machines can be transformed into software machines. The basic problem associated with virtualization is the significant reduction in performance. On average (although the reality is extremely diverse), virtualization reduces performance by a factor of 100: i.e. the resulting software, executed on a machine similar to the machine that has been virtualized, runs 100 times more slowly. In order to recover from this loss of performance, we simply need to run the program on a machine that is 100 times more powerful. This power is to be found in the datacenters hosted in Cloud environments that are under development in all corners of the globe.

It is not possible to virtualize a certain number of elements, such as an antenna or a sensor, since there is no piece of software capable of picking up electromagnetic signals or detecting temperature. Thus, we still need to keep hardware elements such as the metal wires and optical links or the transmission/reception ports of a router and a switch. Nevertheless, all of the signal-processing operations can be virtualized perfectly well. Increasingly, we find virtualization in wireless systems.

In order to speed up the software processing, one solution would be to move to a mode of concretization, i.e. the reverse of virtualization, but with one very significant difference: the hardware must behave like a software. It is possible to replace the software, which is typically executed on a general machine, with a machine that can be reconfigured almost instantly, and thus behaves like a software program. The components used are derived from FPGAs (Field-Programmable Gate

Arrays) and, more generally, reconfigurable microprocessors. A great deal of progress still needs to be made in order to obtain extremely fast concretizations, but this is only a question of a few years.

The virtualization of networking equipment means we can replace the hardware routers with software routers, and do the same for any other piece of hardware that could be made into software, such as switches, LSRs (Label Switching Routers), firewalls, diverse and varied boxes, DPI (Deep Packet Inspection), SIP servers, IP PBXs, etc. These new machines are superior in a number of ways. To begin with, one advantage is their flexibility. Let us look at the example given in Figure 1.1, where three hardware routers have been integrated in software form on a single server. The size of the three virtual routers can change depending on their workload. The router uses little resources at night-time when there is little traffic, and very large resources at peak times in order to handle all the traffic.

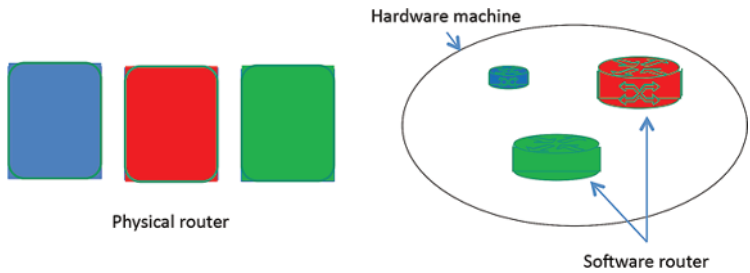

Figure 1.1. *Virtualization of three routers. For a color version of the figure, see www.iste.co.uk/pujolle/software2.zip*

Energy consumption is another argument in favor of virtualization. While, to begin with, consumption would rise because we are adding an extra piece of software (the hypervisor or container), it is possible to share the resources more effectively, and move those resources, grouping them together on physical machines, and put other machines, which have become idle, on standby.

A physical machine can accommodate virtual machines if, as mentioned above, we add a hypervisor or a container manager, which is a software program that enables multiple containers, hence multiple virtual machines, to run simultaneously. In fact, the word “simultaneously” implies a macroscopic scale: on a microscopic scale, the virtual machines are executed sequentially one after another. In the context of virtual servers, this serial execution is not a problem. In the area of networks, it may become a problem for real-time applications, which require a very short response time. Each virtual machine’s processing time must be sufficiently short to give the impression that all the virtual machines are being executed in parallel. Figure 1.2 shows the architecture of virtualization.

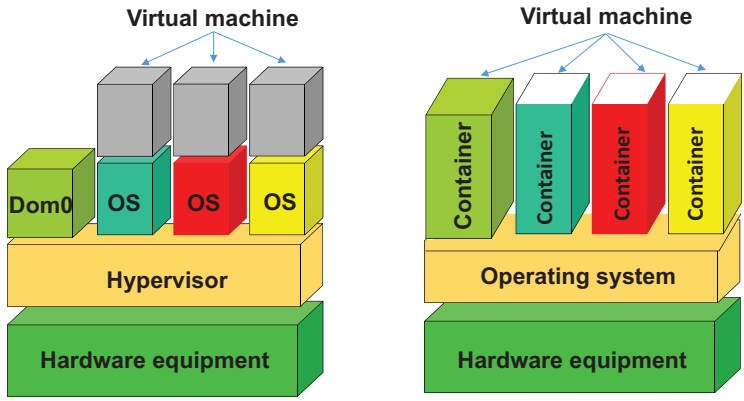

Figure 1.2. *A virtualized machine. For a color version of the figure, see www.iste.co.uk/pujolle/software2.zip*

In this section, we will go over the two solutions to obtain virtual machines, as shown in Figure 1.2. The hypervisor is a virtual machine monitor (VMM), which is often open source. Hypervisors operate on standard hardware platforms. In addition to the VMM, running directly on the physical hardware, the architecture generally comprises a number of domains running simultaneously. These domains execute virtual machines isolated from one another. Each virtual machine may have its own operating system and applications. The VMM controls access to the hardware from the various domains, and manages the sharing of the resources between the different domains. Thus, one of the VMM's main tasks is to isolate the different virtual machines, so that the execution of one virtual machine does not affect the performances of the others.

All peripheral drivers are kept in an isolated domain specific to them. Known as "domain zero" (dom0), it offers a reliable and effective physical support. Dom0 has special privileges in comparison to other domains, known as "user domains" (domU) and, for example, has unfettered access to the hardware of the physical machine. User domains have virtual drivers and operate as though they have direct access to the hardware. However, in reality, those virtual drivers communicate with the dom0 in order to access the physical hardware.

The hypervisor virtualizes a single physical network interface, de-multiplexing the incoming packets from the physical interface to the user domains and, conversely, multiplexing the outgoing packets generated by those user domains. In this procedure, known as virtualization of the network input/output, the domain 0 directly accesses the input/output peripherals, using their native drivers, and performs input/output operations on behalf of the domUs.

The user domains use virtual input/output peripherals, controlled by virtual drivers, to ask the dom0 for access to the peripheral. Each user domain has its own virtual network interfaces, known as foreground interfaces, which are required for network communications. The background interfaces are created in the dom0, corresponding to each foreground interface in a user domain, and act as proxy for the virtual interfaces in the dom0. The foreground and background interfaces are connected to one another via an input/output channel, which uses a zero-copy mechanism to match the physical page containing the packet and the target domain. Thus, the packets are exchanged between the background and foreground interfaces. The foreground interfaces are perceived by the operating systems, working on the user domains, as real interfaces. However, the background interfaces in the dom0 are connected to the physical interface and to one another via a virtual network bridge. It is the default architecture, called "bridge mode", used, for instance, by the Xen hypervisor, which was certainly one of the first to appear. Thus, both the input/output channel and the network bridge establish a path for communication between the virtual interfaces created in the user domains and the physical interface.

We will go back to hypervision techniques later in this chapter. Before this, let us introduce the second solution to support virtual machines, which seems to take the lead thanks to its simplicity and efficiency while offering a little less functionality. This solution is based on a unique operating system supporting containers that host the virtual machines. More precisely, a container is an abstract data structure, class or type that makes it possible to collect objects. The container technique is more flexible and simpler than that embedding an operating system for each virtual machine. Containers can migrate from one hardware to another, thus performing virtual machine migrations. The open source software called Kubernetes, which we will study later in this chapter, makes it possible to orchestrate migrations from one hardware's containers to another hardware in the same cluster. The Kubernetes orchestrator seems to become standard to implement virtual machines in the new generation of networks.

1.1. Software networks

Virtual machines, in turn, can be used to create virtual networks, which are also known as software networks. For this purpose, we need to link virtual machines together in the same way as we would connect different physical machines. Of course, the communication links must be shared between the different software networks. A set of software networks are represented in Figure 1.3.

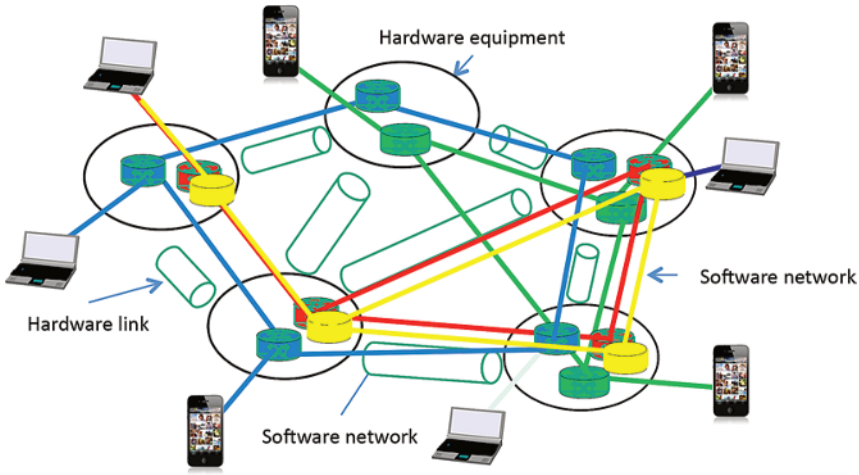

Figure 1.3. *A set of software networks. For a color version of the figure, see www.iste.co.uk/pujolle/software2.zip*

Each software network may have its own architecture and its own characteristics. One software network could be devoted to a VoIP service, another to an IPTV service, a third to a highly secure application, a fourth to support professional applications, a fifth for asynchronous applications such as electronic messaging, etc. We could, in fact, practically create a software network for each user. The personalized software network is set up at the moment when the user connects. It is eliminated when the user signs out. However, this solution does not scale up, and today, we are limited to a number of software networks suited to the hardware capacity of the underlying physical infrastructure. Each software network receives resources allocated to it on the basis of the user demands. However, resources remain shared by different techniques that allow virtual networks to recover resources from other unused virtual networks.

It should be noted that, in general, the virtual nodes are found in datacenters, which may be of varying size and importance: enormous central datacenters, regional datacenters, local datacenters and small datacenters such as femto-datacenters. We will come back later on to the choices which may be made in this field.

One of the characteristics of software networks is that the virtual machines can be migrated from one physical machine to another. This migration may be automated based on whether a node is overloaded or out of order.

In the physical nodes that support the software networks, we can add other types of virtual machines such as firewalls, SIP servers for VoIP, middle boxes, etc. The networks themselves, as stated above, may obey a variety of different protocol architectures such as TCP/IPv4, UDP/IPv4, IPv6, MPLS, Ethernet Carrier Grade, TRILL, LISP, etc.

Isolation is, of course, a crucial property, because it is essential to prevent a problem on one software network from having repercussions for the other networks. The handover of streams from one software network to another must take place via a secure gateway outside of the data plane. This is absolutely necessary to prevent contamination between networks, such as a complete shutdown for a network attacked, for example, by a distributed denial of service (DDOS).

1.2. Hypervisors and containers

Clearly, virtualization needs hardware, which can be standard. We speak of commodity hardware (white box), with open specifications, produced *en masse* to achieve particularly low prices. We will talk further about it in the chapter on open source software (Chapter 4). There are various ways of placing virtual machines on physical equipment, and they can be classified into three broad categories, as shown in Figures 1.4–1.6. The first two figures correspond to hypervisors and the third figure corresponds to containers.

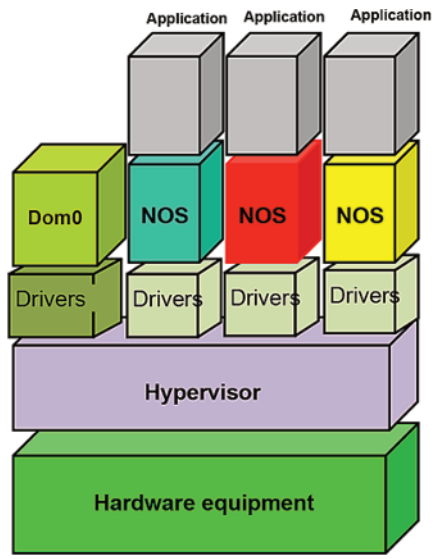

Figure 1.4. *Paravirtualization. For a color version of the figure, see www.iste.co.uk/pujolle/software2.zip*

A paravirtualization hypervisor is a program that is directly executed on a hardware platform and which hosts virtual machines linked to operating systems that have been modified so that the virtual machines' instructions are directly executed on a hardware platform. This platform is able to support guest operating systems with their drivers. The classic hypervisors in this category include Citrix Xen Server (open source), VMware vSphere, VMware ESX, Microsoft Hyper-V Server, Bare Metal and KVM (open source). These programs are also known as type-1 hypervisors.

The second category of hypervisor, or type 2 hypervisor, is a program that is executed on the hardware platform, supporting native operating systems, which means without any modification. The native operating system, when invited by the hypervisor, is executed on the device thanks to an emulator so that the underlying device takes all constructions into account. The guest operating systems are unaware that they are virtualized, so they do not require any modifications, as opposed to paravirtualization. Examples of this type of virtualization would include Microsoft Virtual PC, Microsoft Virtual Server, Parallels Desktop, Parallels Server, Oracle VM Virtual Box (free), VMware Fusion, VMware Player, VMware Server, VMware Workstation and QEMU (open source).

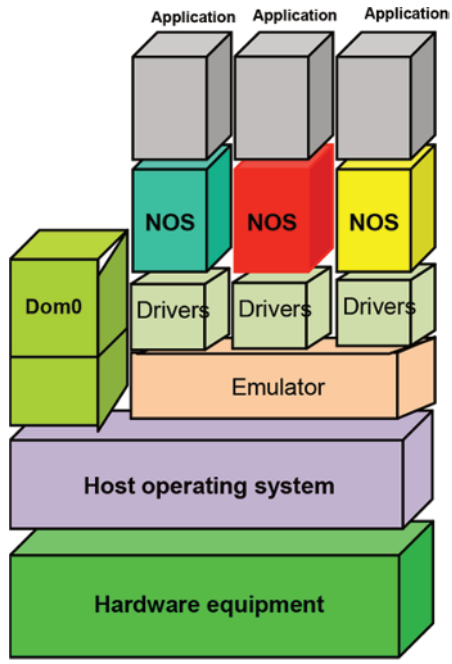

Figure 1.5. *Virtualization by emulation. For a color version of the figure, see www.iste.co.uk/pujolle/software2.zip*

The third type leaves behind the previous hypervisor systems, running several machines simultaneously as containers. In such case, we speak of an isolator. An isolator is a program that isolates the execution of the applications in an environment, called the context, or indeed the zones of execution. Thus, the isolator is able to run the same application multiple times in a multi-instance mode. This solution performs very well, because it does not cause any overload, but the environments are more difficult to isolate.

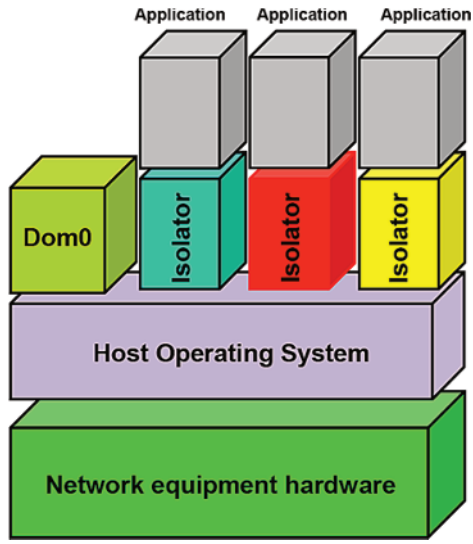

Figure 1.6. *Virtualization by containers. For a color version of the figure, see www.iste.co.uk/pujolle/software2.zip*

In summary, this last solution facilitates the execution of the applications in execution zones. In this category, we can cite the examples of Linux-Vserver, chroot, BSD Jail and Open VZ and most of the container solutions such as Docker.

1.3. Kubernetes

Kubernetes (also called K8s) is an open source system that allows deployment, rise and management of containered applications. This solution was first created by Google, which gave it to the Cloud Native Computing Foundation. This platform allows deployment automation, rise and implementation of application containers on clusters and servers. This open source software works with a whole range of container technologies, such as Docker, for example.

The Kubernetes architecture is shown in Figure 1.7. We can see Pods, which are containers or a group of containers hosted by servers that belong to a cluster of hardware. ETCD is the persisting storage unit for the cluster's configuration data. The scheduler's goal is to share the workload on servers, thus managing Pods' execution in the best possible way. Finally, Kubelet is responsible for the execution state of each server.

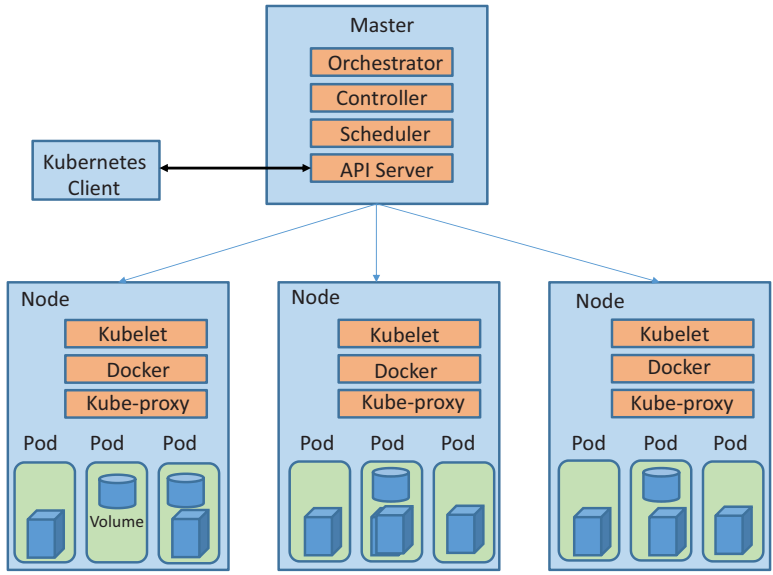

Figure 1.7. *Architecture of the Kubernetes orchestrator*

1.4. Software networks

Software networks have numerous properties that are novel in comparison to hardware networks. To begin with, we can easily move virtual machines around, because they are simply programs. Thus, we can migrate a router from one physical node to another. Migration may occur when a physical node begins to fail, or when a node is overloaded, or for any other reason decided on in advance. Migration of a node does not actually involve transporting the whole of the code for the machine, which would, in certain cases, be rather cumbersome and time-consuming. In general, the program needing to be migrated is already present in the remote node, but it is idle. Therefore, we merely need to begin running the program and send it the configuration of the node to be moved. This requires the transmission of relatively little data, so the latency before the migrated machine starts up is short. In general, we can even let both machines run at once, and change the routing so that the data only flow through the migrated node. We can then shut down the first router.

More generally, we carry out what is known as urbanization: we migrate the virtual machines to different physical machines until we obtain optimal performance. Urbanization is greatly used for optimization in terms of energy consumption or workload distribution, as well as to optimize the cost of the software networks or to make the network highly reliable or resilient. For example, in order to optimize energy consumption, we need to bring together the virtual machines on shared nodes and switch off all the nodes that are no longer active. In actual fact, these machines would not be shut down but rather placed on standby, which does still consume a small amount of energy, but only a very small amount. The major difficulty with urbanization arises when it is necessary to optimize all operational criteria at the same time because they are often incompatible – for example, optimizing consumption and performance at the same time.

A very important characteristic mentioned earlier is isolation: the software networks must be isolated from one another, so that an attack on one network does not affect the other networks. Isolation is complex, because simultaneously, we need to share the common resources and be sure that, at all times, each network has access to its own resources, negotiated at the time of establishment of the software network. In general, a token-based algorithm is used. Every virtual device on every software network receives tokens according to the resources attributed to it. For example, for a physical node, ten tokens might be distributed to network 1, five tokens to network 2 and one token to network 3. The networks spend their tokens on the basis of certain tasks performed, such as the transmission of n bytes. At all times, each device can have its own tokens and thus have a minimum data rate determined when the resources were allocated. However, a problem arises if a network does not have packets to send, because then it does not spend its tokens. A network may have all of its tokens when the other networks have already spent all of theirs. In this case, so as not to immobilize the system, we allocate negative tokens to the other two networks, which can then surpass the usage rate defined when their resources were allocated. When the sum of the remaining tokens less the negative tokens is equal to zero, then the machine's basic tokens are redistributed. This enables us to maintain isolation while still sharing the hardware resources. In addition, we can attach a certain priority to a software network while preserving the isolation, by allowing that particular network to spend its tokens as a matter of priority over the other networks. This is relative priority, because each network can, at any moment, recoup its basic resources. However, the priority can be accentuated by distributing any excess resources to the priority networks, which will then always have a token available to handle a packet. Of course, isolation requires other characteristics of the hypervisors and the virtualization techniques, which we will not discuss in this book.

Virtualization needs to be linked to other features in order to fully make sense. SDN (Software-Defined Networking) is one of the paradigms strongly linked to virtualization, because it involves the uncoupling of the physical part from the

control part. The control part can be virtualized and deported onto another machine, which enables us, for example, to have both a far great processing power than that of the original machine and also a much larger memory available.

1.5. Virtual devices

All devices can be virtualized, with the exception of those which handle the reception of terrestrial and wireless signals, such as electromagnetic signals or atmospheric pressure. For example, an antenna or thermometer could not be replaced by a piece of software. However, the signal received by that antenna or thermometer can be processed by a virtual machine. A sensor picking up a signal can select an appropriate virtual processing machine in order to achieve a result that is appropriate for the demand. One single antenna might, for example, receive signals from a Wi-Fi terminal as well as signals from a 4G terminal. On the basis of the type of signal, an initial virtual machine determines which technology is being used, and sends the signal to the virtual machine needed for its processing. This is known as SDR (Software-Defined Radio), which is becoming increasingly widely used, and enables us to delocalize the processing operation to a datacenter.

The networking machines that we know can always be virtualized, either completely or at least partially. A partial virtualization can correspond to the processing part, the control part or the management part. Thus, today, we can uncouple a physical machine that, in the past, was unique, into several different machines – one of them physical (e.g. a transceiver broadcasting along a metal cable) and the others virtual. One of the advantages of this uncoupling is that we can deport the virtual parts onto other physical machines for execution. This means that we can adapt the power of the resources to the results we wish to obtain. Operations originating on different physical machines can be multiplexed onto the same software machine executing on a single physical server. This solution helps us to economize on the overall cost of the system, as well as on the energy expended, by grouping together the necessary power using a single machine that is much more powerful and more economical.

Today, all legacy machines in the world of networking have either been virtualized already or are in the process of being virtualized – Nodes-B for processing the signals from 3G, 4G and 5G mobile networks, HLRs and VLRs, routers, switches, different types of routers/switches such as those of MPLS, firewalls, authentication or identity management servers, etc. In addition, these virtual machines can be partitioned so they execute on several physical machines in parallel.

We can appreciate the importance of the Cloud and associated datacenters, because they are placed where the processing power is available at a relatively low cost, as is the memory space needed to store the virtual machines and a whole range

of information pertaining to the networks, clients and processing algorithms. For the past few years, with server virtualization, the tendency has been to focus on huge datacenters, but with the help of distribution, the size of datacenters is decreasing. This size varies more and more and some are becoming smaller, becoming skin datacenters or femto-datacenters, or Fog and MEC (Mobile Edge Computing) datacenters.

Another interesting application of virtualization is expanding. It is about digital twins. A hardware is associated with a virtual machine executed in a datacenter located either near or far from the hardware. The virtual machine executes exactly what the hardware does. Obviously, the hardware must supply the virtual machine with power when there is a change in parameters. The virtual machine should produce the same results as the hardware. If results are not similar, this shows a dysfunction from the hardware, and this dysfunction can be studied in real time on the virtual machine. This solution makes it possible to spot malfunctions in real-time and, in most cases, to correct them.

Examples of digital twins are being used or developed just like a plane engine twin that is executed in a datacenter. Similarly, soon, vehicles will have a twin, allowing us to detect malfunctions or to understand an accident. Manufacturers are developing digital twins for objects, but in this case, the digital twin's power can be much bigger and it can perform actions which the object is not powerful enough to perform.

Scientists dream of human digital twins which could keep working while the human sleeps.

1.6. Conclusion

Virtualization is the fundamental property of the new generation of networks, where we make the move from hardware to software. While there is a noticeable reduction in performance at the start, it is compensated by more powerful, less costly physical machines. Nonetheless, the opposite move to virtualization is crucial: that of concretization, i.e. enabling the software to be executed on reconfigurable machines so that the properties of the software are retained and top-of-the-range performances can again be achieved.

Software networks form the backbone of the new means of data transport. They are agile, simple to implement and not costly. They can be modified or changed at will. Virtualization also enables us to uncouple functions and to use shared machines to host algorithms, which offers substantial savings in terms of resources and of qualified personnel.