Research article/ Systems and Computer Engineering https://doi.org/10.15446/ing.investig.97363

A Survey of Virtualization Technologies: Towards a New Taxonomic Proposal

Una revisión de las tecnologías de virtualización: hacia una nueva propuesta taxonómica

Luis E. Sepúlveda-Rodríguez[1], Julio C. Chavarro-Porras[2], John A. Sanabria-Ordoñez[3], Harold E. Castro[4], and Jeanna Matthews[5]

ABSTRACT

At present, there is a proliferation of virtualization technologies (VTs), which are part of the basic and underlying infrastructure of popular cloud computing. Those interested in VTs are faced with a non-unified volume of information and various approaches to modes of operation, classification structures, and the performance implications of these technologies. This makes it difficult to decide which type of VT is appropriate for a particular context. Therefore, this paper reviews the state of the art on VT taxonomic models. Methodologically, a literature review is carried out to identify VT classification models, recognizing their features and weaknesses. With this in mind, a new taxonomy of virtualization technologies is proposed, which responds to the weaknesses identified in the analyzed schemes. The new VT taxonomy combines the Abstraction Level and Virtual Machine Type approaches, providing the reader with a means to visualize VTs. In doing so, the reader can locate the level of abstraction at which each VT is developed, in addition to the type of machine projected, whether it is a complete system or an execution environment for processes. The proposed taxonomy can be used in the academic environment to facilitate teaching processes or in the business environment to facilitate decision-making when implementing VTs.

Keywords: container, taxonomy, virtualization, virtual machine, virtualization technologies

RESUMEN

En la actualidad existe una proliferación de tecnologías de virtualización (VTs), las cuales constituyen una parte de la infraestructura fundamental y subyacente al tan popular cloud computing. Los interesados en las VTs se enfrentan a un volumen de información no unificada y con enfoques diversos acerca de los modos de operación, estructuras de clasificación e implicaciones del desempeño de estas tecnologías. Esto hace difícil decidir sobre el tipo de VT adecuado para un contexto particular. Por lo anterior, este trabajo realiza una revisión del estado del arte acerca de los modelos taxonómicos de las VTs. Metodológicamente, se realiza una revisión de la literatura para identificar modelos de clasificación de las VTs, reconociendo sus características y debilidades. Considerando lo anterior, se propone una nueva taxonomía de las tecnologías de virtualización, que responde a las debilidades identificadas en los esquemas analizados. La nueva taxonomía de VTs combina los enfoques de Nivel de Abstracción y Tipo de Máquina Virtual, proporcionando al lector un medio para visualizar las VTs. Al hacerlo, el lector puede ubicar el nivel de abstracción en el que se desarrolla cada VT, además del tipo de máquina proyectada, ya sea un sistema completo o un entorno de ejecución para procesos. La taxonomía propuesta puede ser utilizada en el ámbito académico para facilitar los procesos de enseñanza o en el ámbito empresarial para favorecer la toma de decisiones a la hora de implementar VTs.

Palabras clave: contenedor, máquina virtual, taxonomía, tecnologías de virtualización

Received: July 20th, 2021
Accepted: June 9th, 2022

Introduction

In recent years Virtualization Technology (VT) has been used to obtain benefits such as isolation, resource splitting, consolidation, security, migration, and ease of management (Varasteh and Goudarzi, 2017). VT builds an abstraction of applications and hardware in a virtual view (AbdElRahem *et al.*, 2016). This virtual view can be different from the physical

How to cite: Sepúlveda-Rodríguez, L. E., Chavarro-Porras, J. C., Sanabria-Ordóñez, J. A., Castro, H., Matthews, J. ((2022). A Survey of Virtualization Technologies: Towards a New Taxonomic Proposal. *Ingeniería e Investigación, 42*(3), e97363. https://doi.org/10.15446/ing.investig.97363

[1] Systems and computing engineer. MSc in Open Software. Affiliation: PhD student in Engineering with emphasis on Computer Science, Universidad Tecnológica de Pereira, Pereira, Colombia. Professor at Universidad del Quindío. Armenia, Colombia. Email: lesepulveda@uniquindio.edu.co

[2] Systems engineer. PhD in Engineering, Universidad del Valle, Cali, Colombia. Affiliation: Universidad Tecnológica de Pereira, Colombia. Email: jchavar@utp.edu.co

[3] Systems engineer. PhD in Computer Information Science and Engineering, Universidad de Puerto Rico. Affiliation: Universidad del Valle, Cali, Colombia. Email: john.sanabria@correounivalle.edu.co

[4] Systems and computing engineer. PhD in Computer Science, Institut National Polytechnique de Grenoble (INPG), Grenoble, France. Affiliation: Computing and Systems Engineering Department, Universidad de los Andes, Bogotá, Colombia. Email: hcastro@uniandes.edu.co

[5] Mathematics and Computer Science, BS PhD in Computer Science, University of California, Berkeley, CA, USA. Affiliation: Department of Computer Science, Clarkson University, Potsdam, NY, USA. Email: jnm@clarkson.edu

view of computing resources (Stallings, 2015). In addition, Silberschatz *et al*. (2014) note that VT allows an operating systems (OS) to run as an application within another OS.

VT includes emulation, which refers to the fact that there are differences between the physical and logical architectures used by virtualized processes. Thus, a virtual machine (VM) could use the same host architecture, a different emulated architecture, or a hybrid. In addition, the processes could use a physical architecture with modifications in order to make virtualization easier (paravirtualization).

VT allows creating one or several environments, *i.e.*, many computers can look like a single large resource (resource aggregation) or, conversely, a single computer is considered as several instances of itself (resource splitting) (Hoopes, 2009; Silberschatz *et al*., 2014).

Unfortunately, the x86 computer architecture, despite being one of the most widely adopted architectures in the world, cannot be completely virtualized (Adams and Agesen, 2006). However, this situation can be solved through mechanisms and VT approaches that act at different levels of abstraction. The abstraction levels where VT takes place are the instruction set level, the hardware abstraction level (HAL), the OS level, the user library interface level, and the application level (Nanda and Chiueh, 2005).

The concept of *virtualization* was formalized in Goldberg's thesis (1973) and published in other works (Goldberg, 1974; Popek and Goldberg, 1974). In these studies, VMs were defined as "an efficient and isolated duplicate of the real machine" (Goldberg, 1973, p. 12). In later works, the term VM was expanded to include other kinds of virtualization, including applications at user level such as libraries, system calls, interfaces/services, system configurations, processes, and state files (Nanda and Chiueh, 2005).

The term *virtual machine monitor* (VMM) was also established by Popek and Goldberg (1974). It is a software layer that supports infrastructure using the resources of a lower level to create multiple independent and isolated VMs (Cafaro and Aloisio, 2011; Nanda and Chiueh, 2005). Similarly, Stallings (2015) determined that a VMM acts as an intermediary between the real machine and VMs. VMMs are also called *hypervisors* (Hoopes, 2009).

VT also brings financial benefits regarding returns on investment and reductions in the total cost of ownership of computer systems hardware (AbdElRahem *et al*., 2016). Moreover, VT uses less energy, which is related to the so-called *green computing* (Jing *et al*., 2013; Ranjith *et al*., 2017; Thathera *et al*., 2015) and plays an essential role in safeguarding the environment. Other goals of VT include increasing the scalability and availability of computing environments, as well as improving the administrative and security structures of the existing computational infrastructure (Hui and Seok, 2014; Kusnetzky, 2011).

Kampert (2010) indicates that the benefits of VTs have revolutionized data centers in the last two decades and have motivated the development of many variations to suit different use cases. In response, several attempts have been made in the academic literature to establish classification schemes for these variations of VT.

This paper reviews VT classification schemes and proposes a new taxonomy that responds to several identified weaknesses. This taxonomy improves and unifies the previous works in the classification of VTs in three ways: first, it combines and unifies approaches that consider the VM type with those that consider the level of abstraction; second, it updates classification approaches to include examples of VTs that have emerged more recently; third, it introduces a taxonomic key diagram based on our unified classification, which can guide the selection of VTs in either academic or production environments.

The remainder of this document comprises the following sections: *VT classification schemes*, *The need for a new taxonomy*, *Proposal for a virtual machine taxonomy*, *Taxonomic key diagram*, and *Conclusions*.

VT classification schemes

This section presents the results of a literature review by means of a systematic process of combined database search and manual reference tracking using the Snowball technique (Samireh and Claes, 2012). In this way, 12 classification schemes for VTs were identified, and their characteristics were highlighted. A paragraph is added at the end of each case which highlights the strengths and weaknesses of the classification scheme.

VT taxonomy by Nanda and Chiueh

Nanda and Chiueh (2005) classified VTs according to the following five levels of abstraction of a computer system.

Instruction set architecture (ISA) level

VTs emulate an ISA, allowing VMs to run as if they were running on hardware. When the ISA offered by this layer differs from the real ISA, this is called *emulation*.

- **Hardware Abstraction Layer (HAL)**: VTs use the same ISA as the host. Here, it is possible to perform independent OS installations, and its applications run as if they were executed in a real environment.
- **Operating System**: VTs work through an OS module to provide a virtualized system call interface.
- **Library Level**: User-level libraries control the communication between the applications and the rest of the system. VTs allow implementation as an Application Binary Interface (ABI) or an Application Programming Interface (API).

- **Programming Language Level**: VTs implement the virtualization layer as an application that can create a simplex or complex VM.

Although Nanda and Chiueh (2005) establish a way to classify VTs, they do not consider virtualization types at the same level of abstraction. Besides, it is necessary to include some VTs that have emerged in recent years.

VM taxonomy by Smith and Nair

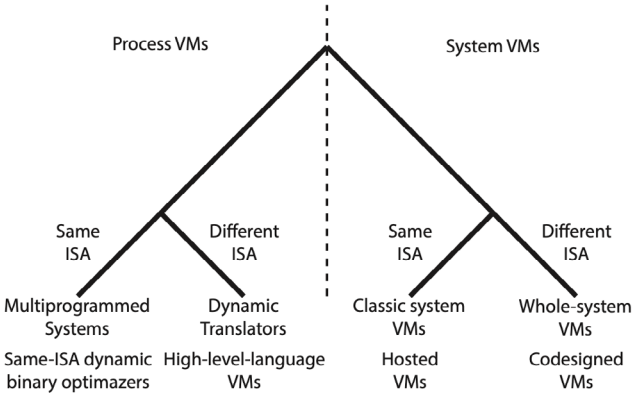

Figure 1. VM taxonomy proposed by Smith and Nair in 2005
Source: Smith and Nair (2005)

Smith and Nair (2005) presented a taxonomy with two main categories: Process VMs and System VMs. Furthermore, these categories divide VTs according to whether the ISA supported in the VM is the same as the underlying hardware (Figure 1).

Process VMs

This category describes an environment in the ABI interface or at the API level. It is called a *Multiprogrammed System* when it uses the same ISA; otherwise, it is called *Dynamic Emulator* or *Binary Translator*. The subcategories are described below:

- **Multiprogrammed Systems** are multiprogramming OSs that implement the management of timeshare access to the available underlying hardware resources. These systems use the same ISA and can handle multiple user processes 'simultaneously'. The OS delivers an individual VM for each user process that runs concurrently. One implementation in this context involves dynamic binary optimizers using the same ISA from the host system.

- **Dynamic Emulators** use process VMs to support compiled binary programs for an ISA different from the underlying hardware. This condition implies executing an emulation effort performed through interpretation, which can be relatively slow. However, this situation can be compensated when a software cache is implemented in order to deal with the overload.

System VMs

These are characterized by hosting one or several complete and independent OSs running simultaneously on the same hardware of the host computer, which results from the intermediation performed by the VMM. The subcategories of the system VMs are described below:

- **Classic System VMs** use the VMM and execute it directly on the bare hardware without an underlying OS. Thus, the VMM has real access to hardware resources and serves as an intermediary between the guest OSs and the hardware itself. In this case, the VMs are called *Hosted VMs*.

- **Whole-system VMs** provide virtualization of a complete environment, but guest systems use an ISA different from those used in the underlying hardware, unlike the previous category. In this case, the VMs are called *Codesigned VMs*.

Smith and Nair's study (2005) can be considered an essential basis for classifying VTs that provide a virtual environment for a complete system or processes. However, this work does not contemplate what was established by Nanda and Chiueh (2005) regarding the levels of abstraction. Another important aspect is that this classification model does not have a high degree of detail; it uses very general descriptions, without even including specific technologies. It is essential to consider that this study was carried out in 2005 and does not include subsequently developed technologies.

Virtualization taxonomy by the SCOPE Alliance

The SCOPE Alliance (2008) proposed an extension of the work carried out by Smith and Nair in 2005. The proposal includes more branching of the main categories and more examples of VTs (Figure 2).

This classification places type I and type II hypervisors as distinctions of the Classic OS VM model of System VMs that support the same ISA as the underlying hardware.

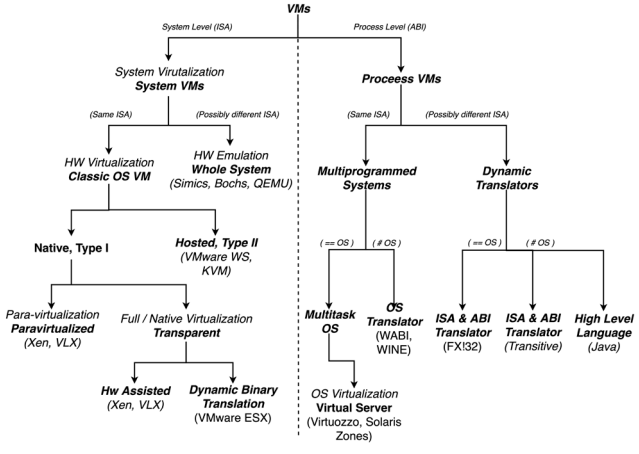

Figure 2. Virtualization taxonomy by the SCOPE Alliance
Source: SCOPE Alliance (2008)

Regarding the Process VMs category, this classification distinguishes between a Multiprogrammed System and Dynamic Translators. Multiprogrammed Systems are further classified depending on whether the OS provided by the underlying system is the same as the OS used by the application. If it uses the same OS, the category is called *Multitask OS*, which contains OS Virtualization. If the OS is different, it is called *OS Translator*. When the processes are based on a different ISA, they are called *Dynamic Translators*. Finally, if the VMs use the same OS, they are called *ISA & ABI Translators*; otherwise, they are called *High-level Language*.

Although the SCOPE Alliance's study (2008) contributes to complementing the taxonomy of VTs, the research does not contemplate aspects such as the levels of abstraction indicated by Nanda and Chiueh (2005). This situation gives rise to problems of conceptual inference, in which, for example, type I and type II hypervisors are perceived to be at the same level of abstraction. Additionally, according to the date of publication of the study, it is necessary to expand concepts and update VTs that have emerged in recent years.

Taxonomy of VTs by Kampert

Kampert (2010) presented his taxonomy of VTs using different virtualization techniques. This taxonomy uses the unified modeling language, as shown in Figure 3, where all elements are classes. For example, the class Domain is a superclass of the classes Server, Application, Desktop, Storage, and Network.

Kampert's taxonomy (2010) aims to cover the domains in a complete way in which the concept of virtualization takes place, including storage and network virtualization not seen in previous taxonomies. However, this taxonomy itself does not offer the level of granularity necessary to identify VTs in each of the specified domains.

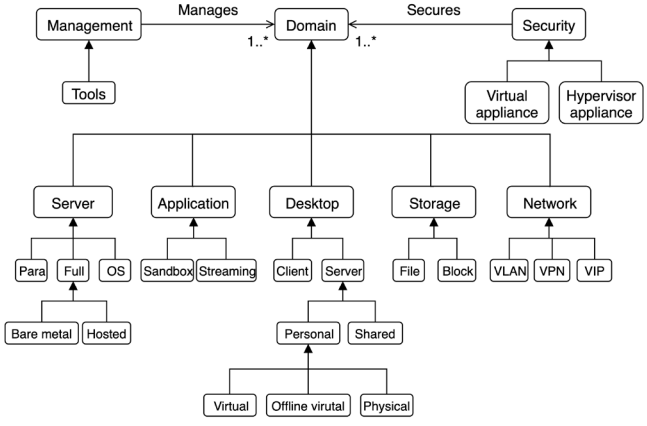

Figure 3. VT taxonomy by Pual Kampert
Source: Kampert (2010)

Virtualization model by Kusnetzky

Kusnetzky's virtualization model (2011) is composed of seven parts, five distributed in layers, and two arranged parallel to the layers above Kampert (2010). Each part is briefly described below:

- **Access virtualization**: Many users share the same system.
- **Application virtualization**: Many applications run transparently on different OSs and hardware platforms.
- **Processing virtualization** allows the division or aggregation of resources.
- **Network virtualization** presents a logical view of the physical network elements.
- **Storage virtualization** hides the location and type of physical storage devices in which applications store their data.
- **Security for virtual environment** controls the access to the various elements of virtual media in order to protect them from unauthorized actions.
- **Management of the virtual environment** controls the available physical resources and the generated virtual environments.

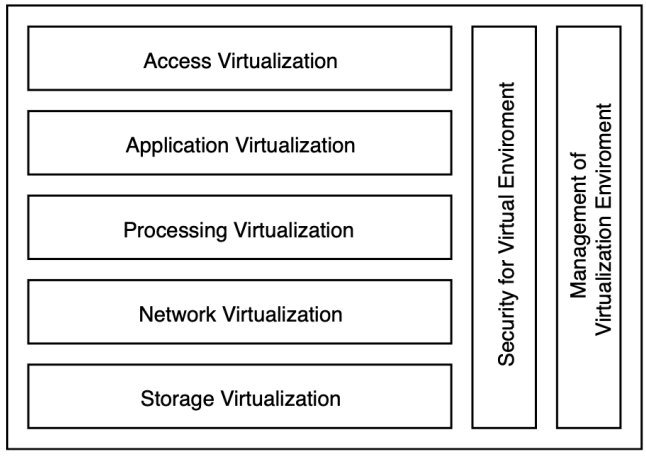

Figure 4. Kusnetzky's model of virtualization
Source: Kusnetzky (2011)

Kusnetzky presents a way to include categories for a range of virtualizable computational resources but does not provide details about the existing VTs in each layer of the model. In addition, the model does not differentiate between technologies of the same layer. For example, in Processing Virtualization, there is no evidence of a difference between the types of VMs present in type I or type II hypervisors.

Taxonomy of VTs by Pessolani

Pessolani *et al*. (2012) proposed their taxonomy of VTs with five main categories: 1) Hardware or System Virtualization, 2) Para-virtualization, 3) Virtualization based on OS, 4)

Virtualization at the Process or Application level, and 5) Virtualization of OS. Additionally, the main categories include subcategories that suggest a level of abstraction (Figure 5). These main categories are described below:

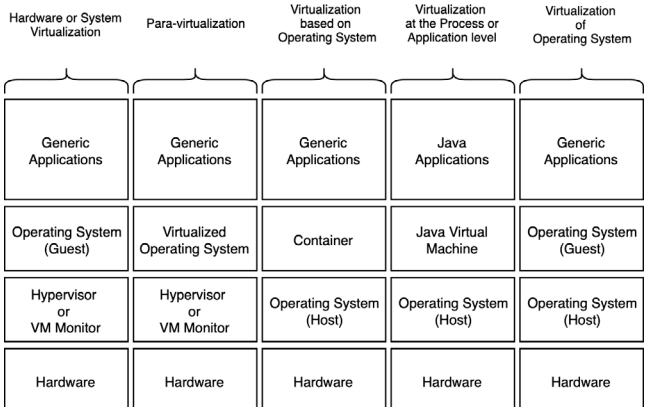

Figure 5. Taxonomy of VTs proposed by Pessolani *et al.*
Source: Pessolani *et al*. (2012)

- **Hardware or System Virtualization** puts the type I hypervisor on top of the hardware with its VMs and their respective guest OSs.
- **Paravirtualization** distributes its elements to Hardware or System Virtualization, but the guest OS is modified to be aware that it is virtualized.
- **Virtualization based on OS** is founded on using independent workspaces called *containers*, which are based on the host OS.
- **Virtualization at the Process or Application level** uses an application on the host OS to provide a VM that allows the execution of processes based on it.
- **Virtualization of OS** needs a host OS to carry out the functions of a hypervisor in order to support the guest OSs, which in turn have their own completely independent applications.

Pessolani's taxonomy (2012) does not explicitly consider the levels of abstraction to which these technologies apply. In addition, it focuses only on the conceptual elements, leaving specific examples aside, nor does it establish a way to divide types of VMs within each main category.

Taxonomy of virtualization concepts by Pék

Pék *et al.* published a taxonomy of virtualization concepts in 2013. This work extends the studies by Smith and Nair (2005) an the SCOPE Alliance (2008) (Figure 6).

This taxonomy adds elements and several components, such as in the Hosted category, equivalent to type II hypervisors from the study by the SCOPE Alliance (2008). It also includes the Paravirtualization subcategory.

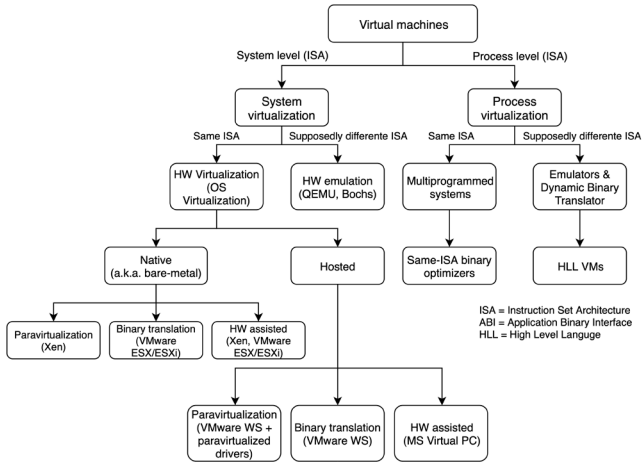

Figure 6. Taxonomy of virtualization concepts by Pék *et al.*
Source: Pék *et al*. (2013)

Although the study by Pék *et al*. (2013) presents an extension to some previous works, this taxonomy leaves a gap in the search for the details of VT categorization, since they do not contemplate the levels of abstraction at which VTs are implemented.

Taxonomy of virtualization by Ameen

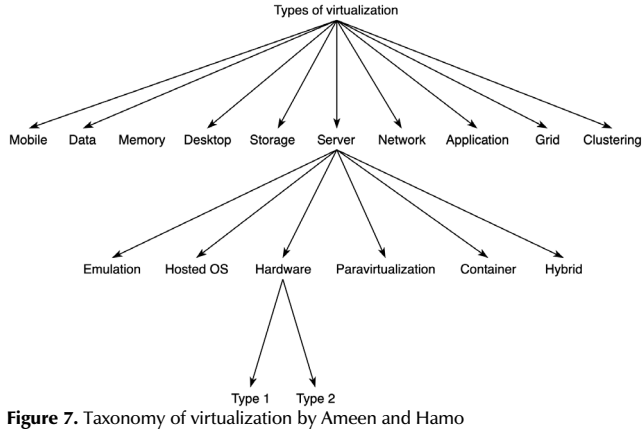

Figure 7. Taxonomy of virtualization by Ameen and Hamo
Source: Ameen and Hamo (2013)

Ameen and Hamo presented a taxonomy with three levels in 2013. (Figure 7). The first level contains the following ten categories:

- **Mobile** software that is embedded on a mobile phone to decouple the applications and data from the underlying hardware (VMware, 2022).
- **Data** abstracts the source of individual data items and provides a common data access layer for different data access methods such as SQL, XML, JDBC, File access, MQ, JMS, *etc.* (Mann, 2006).
- **Memory** adds an extra level of address translation to give each VM the illusion of having zero memory address space, as real hardware provides (Waldspurger, 2002).

- **Desktop** is the ability to display a graphical desktop from one computer system on another computer (von Hagen, 2008).
- **Storage** creates logical abstractions of physical storage systems (B. Li *et al.*, 2005).
- **Server** is a type of virtualization that allows running many OSs both in isolation and independence.
- **Network** provides an abstraction layer that can decouple the physical network equipment from the delivered business services over the network (Annapareddy, 2011).
- **Application** allows the user to run the application using local resources without installing the application in his system completely (Annapareddy, 2011; White and Pilbeam, 2010).
- **Grid** provides a way to abstract multiple physical servers from the application they are running (Mann, 2006).
- **Clustering** causes several locally connected physical systems to appear to the application and end-users as a single processing resource (Mann, 2006).
- The following describes the virtualization types at the second level of the taxonomy, which are derived from the Server category, as indicated by Ameen and Hamo (2013):
- **Emulation** is a virtualization method in which you can create a complete hardware architecture in software (Ameen and Hamo, 2013).
- **Hosted OS** uses software-only. The hypervisor is over an OS (Ameen and Hamo, 2013; von Hagen, 2008).
- **Hardware** the hypervisor is assisted by processor hardware such as AMD-V or Intel VT-x processor virtualization technologies (von Hagen, 2008).
- **Paravirtualization,** according to Ameen and Hamo (2013, p. 7), is "a technique in which the guest OS includes modified (para-virtualized) I/O drivers for the hardware".
- **Container** is a kernel-layer abstraction and refers to techniques in which the abstraction technology is built directly into the OS kernel rather than having a separate hypervisor layer (Ameen and Hamo, 2013; Q. Lin *et al.*, 2012).
- **Hybrid** is a combination of Full Virtualization and Paravirtualization that uses input/output (I/O) acceleration techniques (White and Pilbeam, 2010).

At the third level of the taxonomy are the type I and type II hypervisor categories derived from Server/Hardware.

Ameen and Hamo's taxonomy (2013) is closely related to the works by Kampert (2010) and Kusnetzky (2011). Furthermore, it presents a classification scheme through a three-level hierarchical structure. However, although this graphical representation is interesting, it is unbalanced, since it focuses only on detailing the Server category.

Taxonomy of VTs by Abdulhamid

Abdulhamid *et al.* (2014) presented a taxonomy focused on cloud computing (Abdekhoda *et al.*, 2019; Fareghzadeh *et al.*, 2019) and based on the work by Sahoo *et al.* (2010), which includes categories such as Full Virtualization, OS-Layer Virtualization, Hardware-Layer Virtualization, Paravirtualization, Application Virtualization, Resource virtualization, and Storage virtualization. In addition, this work adds the Grid Virtualization and Cloud Virtualization categories (Figure 8).

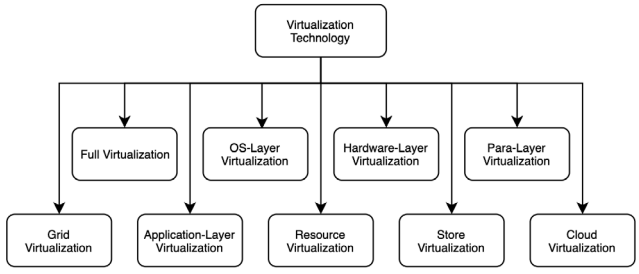

Figure 8. Taxonomy of VTs by Abdulhamid
Source: Abdulhamid *et al.* (2014)

Some categories have already been described. Below is a brief description of the new ones.

- **Grid Virtualization** focuses on the virtualization of grid resources either for a virtual organization (VO) or for a Virtual Organization Cluster (Abdulhamid *et al.*, 2014).
- **Cloud Virtualization** or **Cloud Computing** (Sehgal and Bhatt, 2018) enables on-demand provisioning of virtual resources through the Web, as well as applying the concept of pay-per-use. In this category, the VTs form cloud computing services, provisioning virtual resources to customers on demand. (Abdulhamid *et al.*, 2014; Aceto *et al.*, 2013).

Although the taxonomy by Abdulhamid *et al.* (2014) shows two levels, only one level can be observed which comprises its nine categories from a hierarchical perspective. On the other hand, the description of each category lacks details and examples of VTs.

Types of VMs by Li

X.-F. Li (2016) presented his work with four types of VMs:

- **Type 1**: The Full ISA VM allows full ISA-level emulation or virtualization. The OS and its applications can run on top of the VM as a real machine (X.-F. Li, 2016).

- **Type 2**: The ABI VM allows ABI-level emulation of the processes in the guest OS. These applications can run in conjunction with native ABI applications (X.-F. Li, 2016).

- **Type 3**: The Virtual ISA VM provides a runtime engine for applications encoded in the virtual ISA to run on it (X.-F. Li, 2016).

- **Type 4**: The Language VM gives a runtime engine that runs programs written in a guest language (source). The runtime engine needs to interpret or translate the program.

Although the study by X.-F. Li (2016) presented a four-type classification scheme, it does not indicate a hierarchical structure that clarifies how they relate. It also does not have a supporting graph to facilitate understanding. This work does not contemplate many of the categories indicated in other previously presented taxonomies.

Taxonomy of VMs by Bugnion

Bugnion (2017) presented a structure with two levels that shows the concepts related to VMs. The first level is related to abstraction, and it includes the following categories: Language-based VM, System-level VM, and Lightweight VM. The second level is related to the platform, and it includes two categories derived from System-level VM, which are called *Machine Simulator* and *Hypervisor*. The latter is divided into Bare-metal Hypervisor (type I) and Hosted Hypervisor (type II) (Figure 9).

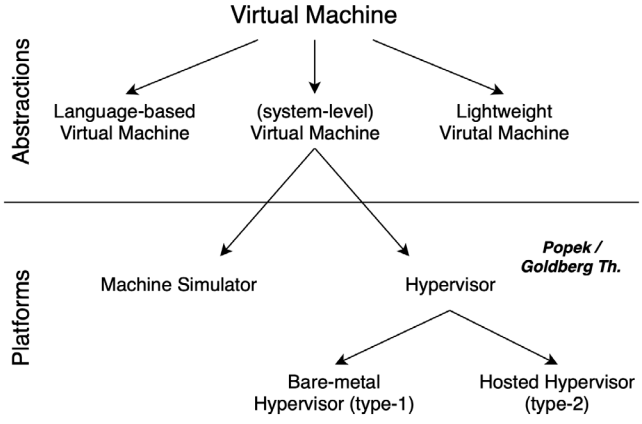

Figure 9. Taxonomy of VMs and the platforms that run them presented by Bugnion
Source: Bugnion *et al.* (2017)

- **Language-based VM** refers to any managed language runtime environment such as the Java VM, Microsoft Common Language Runtime, and JavaScript engines embedded in browsers.

- **Lightweight VM** refers to software mechanisms to ensure that applications run directly on the processor as securely isolated from other environments and the underlying OS.

- **System-level VM** refers to the computer environment that resembles the hardware of a computer, so that the VM can run an OS and its applications in complete isolation from the other VMs and the rest of the environment. This category includes two Hypervisor types (type I and type II).

Bugnion's work (2017) is less a taxonomy than a book focusing on the core architectural support provided by hardware to run VMs efficiently.

The need for a new taxonomy

The taxonomies described above have many elements that contribute to the classification of VTs. However, in each of these classification schemes, some aspects that need improvement have been identified. Each scheme offers a taxonomic approach, such as a) Abstraction Level, b) Type of VM, and c) Virtualization Domains. Table 1 summarizes the classification schemes analyzed in this paper by author, year, and taxonomic approach, which were published between 2005 and 2017. It is worth noting that we found no taxonomies published between 2018 and 2021.

The Type of VM is the most popular approach, as demonstrated by CS2, CS3, CS6, and ACS7. On the other hand, CS4 and CS5 take a different perspective; their objective is to consider, in a general way, the largest number of technological domains in which it is possible to carry out virtualization processes, hence the name Virtualization Domain. Some taxonomies have a dual approach; for example, CS8 and CS9 combine the Type of VM with the Virtualization Domain, and CS11 combines the Type of VM with the Abstraction Level. Lastly, CS1 and CS10 consider the Abstraction Level approach as fundamental for the categorization of VTs. These differences in viewing VTs can confuse the community interested in this field when reading different authors.

Table 1. Summary of classification schemes

Classification Schemes (CS)		
ID	*Author*(s)	*Taxonomy approach*
CS1	Nanda and Chiueh (2005)	Abstraction Level
CS2	Smith and Nair (2005)	Type of VM
CS3	SCOPE Alliance (2008)	Type of VM
CS4	Kampert (2010)	Virtualization Domain
CS5	Kusnetzky (2011)	Virtualization Domain
CS6	Pessolani *et al.* (2012)	Type of VM
CS7	Pék *et al.* (2013)	Type of VM
CS8	Ameen and Hamo (2013)	Type of VM and Virtualization Domain
CS9	Abdulhamid *et al.* (2014)	Type of VM and Virtualization Domain
CS11	X.-F. Li (2016)	Abstraction Level
CS12	Bugnion (2017)	Type of VM and Abstraction Level

Source: Authors

Therefore, there is a need for a new taxonomy that provides a unified, organized, and current view of VTs. Therefore, this paper makes the following contributions:

- A review of the literature with the identification, analysis, and comparison of 12 classification VT schemes (Table 1).
- A proposal for a new VM taxonomy. This work identified, expanded, and combined different studies, offering a single view of multiple concepts such as the Types of VMs and their corresponding Abstraction Level. The taxonomy includes examples of older VTs in order to provide a reference factor to those who have some knowledge about them. It also includes examples of new VTs that have gained wide recognition in the industry and academia, such as those related to containers. The taxonomy is also intended to be an instrument to support the pedagogical processes within the academic community with interests in VTs (Figure 10).
- A taxonomic key diagram that facilitates the visualization of the technological ecosystem that surrounds this topic and consequently helps the academic and industrial community in the decision-making processes regarding the selection of VTs (Figure 11).

Proposal for a virtual machine taxonomy

This section presents a new taxonomic proposal for virtualization technologies. This taxonomy considers the 12 studies reviewed in this research, but it focuses mainly on studies such as CS1, CS2, CS3, CS7, and CS11 (Figure 10). The proposal presents an innovative contribution that integrates the Abstraction Level and Type of VM taxonomic approaches. In addition, it contributes by extending the examples of VTs, which are placed in the diagram representing the new taxonomy. The first approach considers the layers of the classical architecture of a computer system and makes it possible to visualize the VTs according to the level of abstraction they occupy at the time of execution. The second approach considers the types of virtual machines, be it complete systems or execution environments for processes. The description of the taxonomy is shown below, making a cross-analysis between the two approaches.

Approach 1: abstraction layers

The first approach of this taxonomy uses the abstraction layers in a computer system, such as the Hardware Abstraction Layer (HAL), the Operating System (OS), the Application Binary Interface (ABI), the Application Programming Interface (API), Type I/Type II Hypervisors, and Libraries. In Figure 10, the labels located on the left side indicate the abstraction level, and they are the title of rectangular structures with horizontal distribution in the taxonomy. With these layers, the taxonomy makes it possible to locate VTs depending on the level at which they take place. Thus, the reader can quickly infer aspects such as the dependence or not of an underlying OS, as well as determine the number of intermediaries involved in the virtualization process. Furthermore, this information allows inferring the possible performance of these technologies. The abstraction layers are described below from bottom to top.

Hardware Abstraction Layer (HAL)

HAL includes those VTs that are placed directly on top of the hardware. This arrangement is also known as *BareMetal* and is identified by the absence of intermediaries between the VMs and the underlying hardware, suggesting a higher performance for the set of VTs placed here. This layer contains the category called *Type I Hypervisor* and can have several types of VTs.

Operating System (OS)

This layer contains the sublayers Application Binary Interface (ABI) and Application Programming Interface (API). In the ABI sublayer, the VTs use the OS as an intermediary to access the underlying hardware. The virtualization is carried by OS calls and uses Dynamic Binary Translation, Type II Hypervisors, or Libraries. This situation suggests that the VTs may present degradation in performance due to intermediation costs between the different environments. VTs implement virtualization based on high-level languages, offering portability in the API sublayer, as APIs support multiple hardware and software platforms. However, this sublayer has considerable degradation given the multiple interpreters between the VTs and the hardware functions.

Approach 2: Type of VM

The second approach of this taxonomy considers VTs according to their type: System VMs or Process VMs. System VMs contain a whole OS (guest OS) within their virtual environment. On the other hand, Process VMs use the host OS as an intermediary between the virtual environment and the actual hardware.

System VMs

This type of virtualization has two categories. The first is Classic System VMs and is characterized by the fact that the host and guest OSs have the same ISA. The second category is Whole-System VMs and is characterized by the host OS and guest OS having a different ISA.

- **Classic System VMs**: This category is known as *Hardware Virtualization* and includes two subcategories: the first one is Native VMs, and the second one is Hosted VMs. It is important to note that each subcategory takes place at different levels of abstraction.
    - **Native VMs**: This VT is also known as *Type I Hypervisor* and corresponds to the HAL abstraction

level. It uses a software layer directly on top of the hardware. It also presents a subdivision, as shown below:

- **Transparent** indicates that the OS inside the VM is unaware of its virtualization state and is divided into the following types:
  - **Hardware-Assisted** virtualization involves the use of physical components to facilitate the management of VMs. Examples of this are: KVM (2021), Microsoft Hyper-V (Jason *et al*., 2009; Syrewicze and Siddaway, 2018), Xen (Xen Cambridge, 2022), VLX (Armand and Gien, 2009), and VMware ESX/ESXi (Z. Li, 2021; VMware, 2022).
  - **Dynamic Binary Translation** implies that the Type I Hypervisor catches and inspects the code of each guest OS request to convert it into a proper request towards the underlying hardware, *e.g.*, VMware ESX/ESXi (Z. Li, 2021; VMware, 2022) and XtratuM (Wessman *et al*., 2021; Xtratum, 2022) .
- **Para-virtualized** is also known as *Operating System-Assisted Virtualization* and refers to efficient communication between the guest OS and the hypervisor. This implies modifying the guest OS to be aware of virtualization and to take advantage of that condition. Examples of this are: Xen (Barham *et al*., 2003; Matthews *et al*., 2008; Xen Cambridge, 2022; Xen Project, 2022), VLX (Armand and Gien, 2009), KVM (Abeni and Faggioli, 2020; KVM, 2021), and VMware VMI (VMware, 2022).

- **Hosted VMs**: This subcategory is also known as *Type 2 Hypervisors,* corresponds to the ABI abstraction level, and uses a layer of software on a Host OS. It presents the same subdivision and functions of the **Native VMs** category, so only examples of VTs will be listed below.
  - **Transparent**
    - **Hardware-Assisted**: VMware Workstation/ Fusion (VMware, 2022), Parallels Desktop (Parallels, 2021), and Oracle VirtualBox (Oracle, 2021b).
    - **Dynamic Binary Translation**: VMware Workstation/Fusion (VMware, 2022; Z. Li, 2021), Microsoft Virtual PC (Honeycutt, 2003), Plex86 (2021), Parallels Desktop (Parallels, 2021), and Oracle VirtualBox (Oracle, 2021b).
  - **Para-virtualized**: VMware Workstation, with the addition of the corresponding para-virtualization driver to the network in the guest OS (El-Anani, 2021; VMware, 2022).
- **Whole system VMs**: This category is called *Hardware Emulation* and presents an ISA different from the underlying hardware. It takes place at the API abstraction level, evidencing a preexisting OS on which emulation can occur. The subcategory is called *Dynamic Binary Translation* and features VTs such as QEMU (Díaz *et al*., 2021; QEMU, 2021), Simcs (Magnusson *et al*., 2002), Bochs (Bochs, 2021), Rosetta (Apple Inc, 2009), and BIRD (Nanda *et al*., 2006).

Process VMs

This type of virtualization also has the same two categories as System VMs, depending on whether the host OS and guest OS have the same ISA. When the ISA is the same, the category is called *Multiprogrammed Systems*; otherwise, the category is called *Dynamic Translators*. Both categories are located at the OS layer.

- **Multi-programmed systems**: In this category, the VTs share the OS among many processes, generating independent execution spaces for each one. This generates the illusion that, for a moment, a process is an exclusive executor in the system. This category is then divided into two, depending on whether there is an OS. When the same OS is projecting, the category is called *Multitasking OS*; otherwise, it is called *OS Translators*.
  - **Multitasking OS** is divided into Operating System Virtualization and Same-ISA Dynamic Binary Optimizer.
    - **Operating System Virtualization** happens at the ABI abstraction level and uses system calls for interaction with the underlying hardware. It uses the preexisting OS, and it allows generating independent workspaces for the processes. This type of virtualization is booming and is often known as l*ightweight virtualization*, *container-based*, or simply *containers* (Tfrifonov, 2018). For example: FreeBSD Jails (Biederman, 2006; Kamp and Watson, 2000) (Ryding and Johansson, 2020) , Solaris Zones/Containers (Oracle, 2021a), OpenVZ (2021), Linux-VServer (Linux-VServer, 2018), AIX Workload Partitions WPAR (Gibson, 2007), Parallels Virtuozzo Containers (Virtuozzo, 2022), Denali (Whitaker *et al*., 2002), Google Native Client (Yee *et al*., 2009), Vx32 (Ford and Cox, 2008), User-Mode Linux (Dike, 2006; User-Mode Linux, 2022), Minix Over Linux (Pessolani and Jara, 2011), Ensim (2022), LXC (Canonical Ltd., 2021), Docker (Docker, 2022; Ryding and Johansson,

2020), and Singularity (Chang *et al.*, 2021; Sylabs.io, 2022).

- **Same-ISA Dynamic Binary Optimizers** are translators implemented in software that perform optimized translations of binary code with an equal ISA. Their operation is transparent, and even the system's native binaries can be optimized. An example of this is the Dynamo project (Bala *et al.*, 2011).

- **Operating System Translators** allow the execution of applications built for OSs different from the system host, *e.g.*, WINE (Jones *et al.*, 2018; Wine, 2022), WABI (Oracle, 2018), Lxrun (2022), Visual MainWin (Fisher *et al.*, 2006), and Vcuda (Balis *et al.*, 2021; S. Lin *et al.*, 2009).

- **Dynamic Translators**: Dynamic ISA translators can support processes that use the same host OS, *e.g.*, FX!32 (Chernoff *et al.*, 1998). It can also be the case of dynamic ISA translations for processes that use a different OS than the host, such as Transitive (eWeek, 2008; IBM, 2008). For the above cases, the translation occurs at the Library level. It can also be the case of dynamic

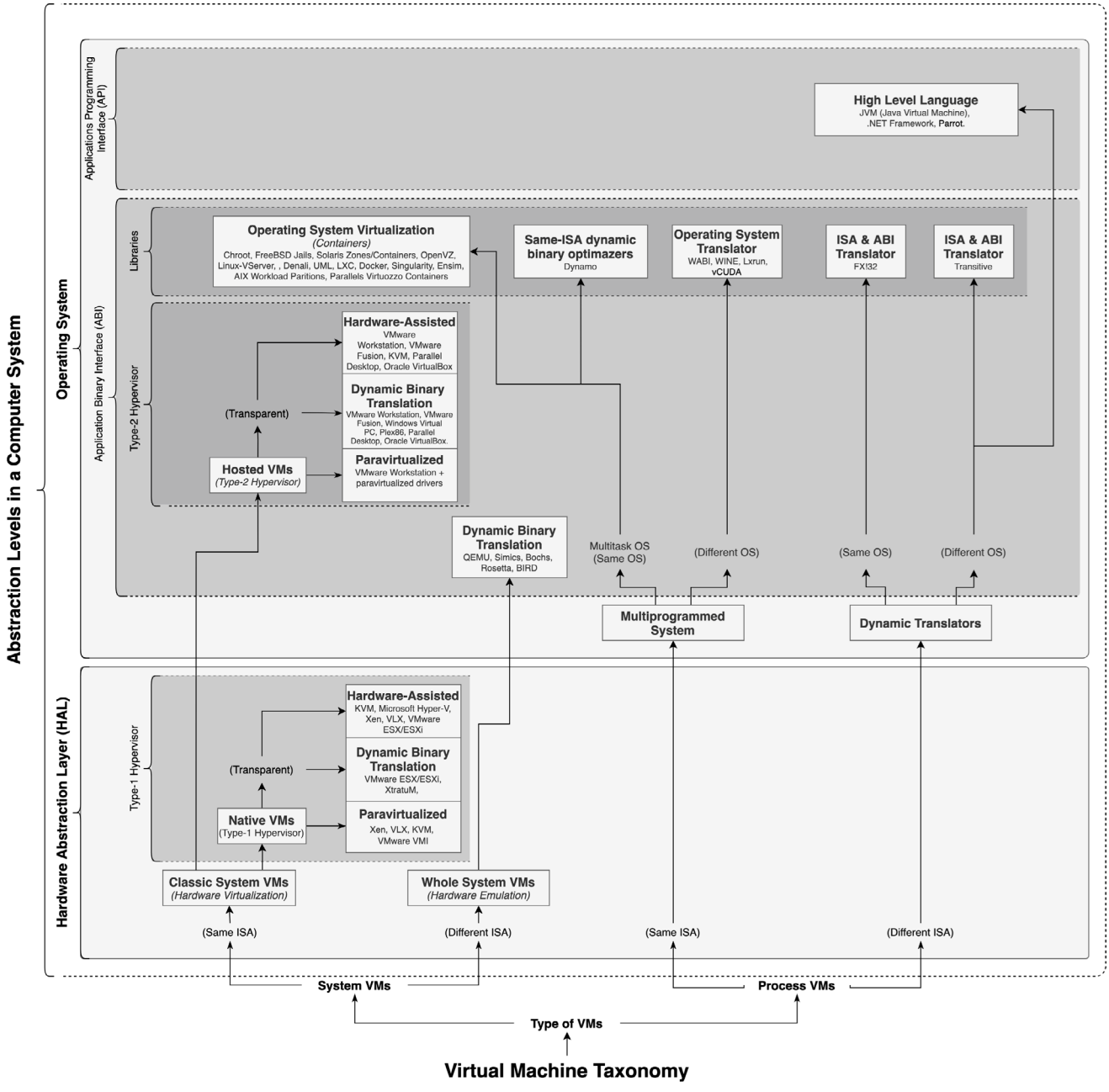

Figure 10. Proposal for a new virtual machine taxonomy
Source: Authors.

ISA translators for processes using a different OS and acting through high-level languages such as Java Virtual Machine (JVM) (Lindholm *et al*., 1997; Beronić *et al*., 2021), the Microsoft .NET common language infrastructure (CLI) (Thai and Lam, 2003), and Parrot (2022).

Taxonomic key diagram

This work also proposes a taxonomic key diagram to guide decision-making about the technologies related to VMs, as indicated in the proposed taxonomy (Figure 11). The diagram uses a set of questions, which, depending on each possible answer, establishes a path that leads to identifying a VT defined in the aforementioned taxonomy. For example, the diagram can be used by asking the question 'Do you need to virtualize the entire system or just some of its processes?' If the complete system needs to be virtualized, the following question will inquire about the specific need. If the desired virtual system needs an ISA different from the underlying hardware, the answer from the taxonomic key is the Dynamic Binary Translation category, *e.g.*, QEMU, Simics, and Bochs.

Conclusions

A review of literature on the different classification schemes for virtualization technologies proposed since 2005. These schemes have been introduced using a timeline that has allowed the identification of the following taxonomic approaches: Abstraction Level, Virtual Machine Type, and Virtualization Domains.

When performing the analysis of each classification scheme, it was possible to identify weaknesses. These include the presence of a single taxonomic approach in each scheme and the lack of topicality considering the date of publication, as well as the absence of the details on the inclusion of technologies.

The proposed taxonomy responds to the needs identified in the analyzed classification schemes. As a result, the proposal combines the Abstraction Level and Virtual Machine Type approaches, giving the reader a means of visualizing the virtualization technologies relating to virtual machines. By doing so, the reader is always aware of the level of abstraction at which each technology takes place, in

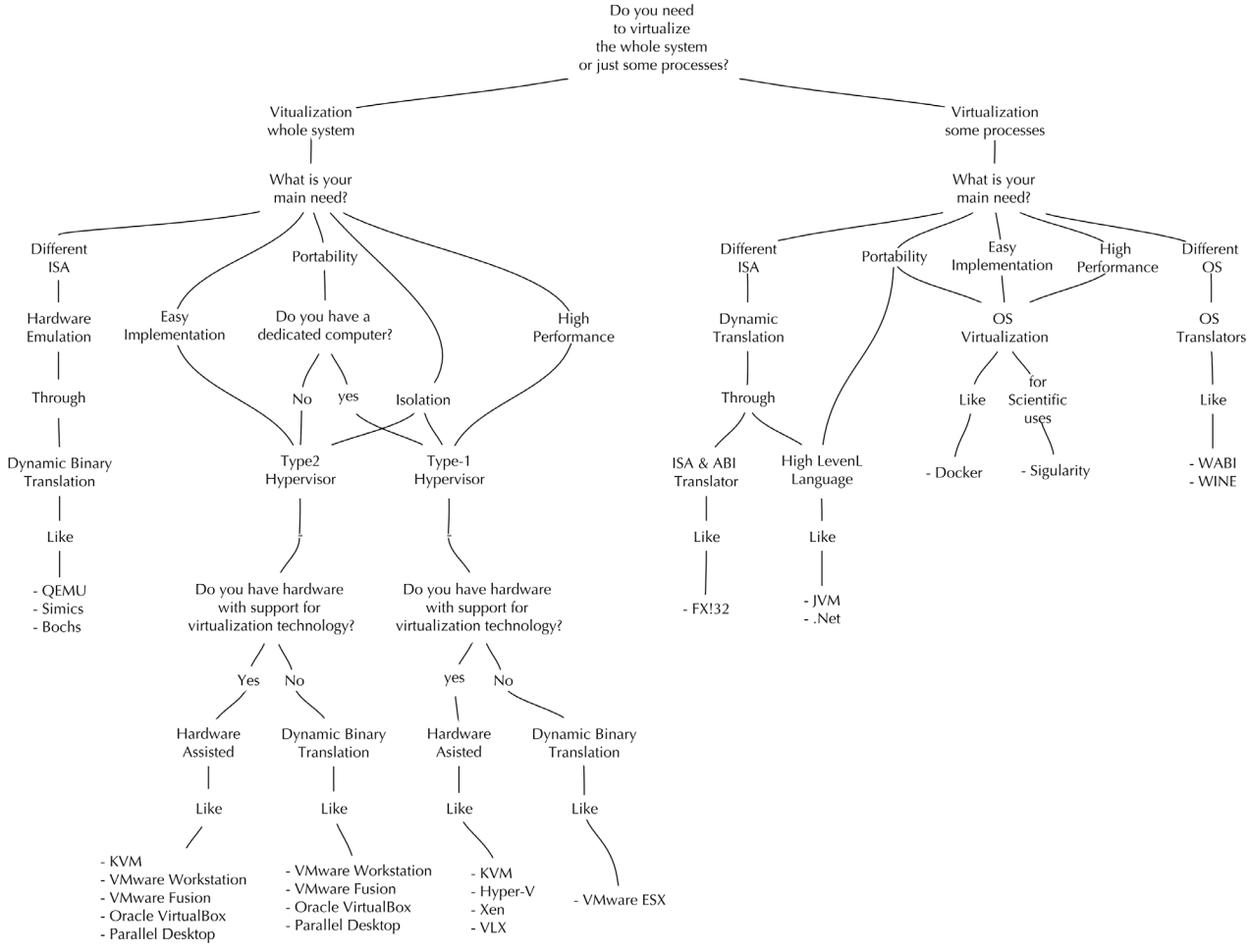

Figure 11. Taxonomic key diagram to select VTs
Source: Authors

addition to the type of machine projected, be it a complete system or an execution environment for processes.

The proposed taxonomy can be used in academic contexts to facilitate teaching and learning or in the business field to favor decision-making when implementing technologies related to virtual machines.

The taxonomy allows for the classification of VTs present in more than one conceptual branch, as these tools evolve, meeting the needs of more than one approach by themselves or using extensions.

Finally, a taxonomic key diagram has been created for use by the industry in order to aid the selection of virtualization technologies.

References

Abdekhoda, M., Asadi, Z., and Nadrian, H. (2019). Cloud computing services adoption among higher education faculties: Development of a standardized questionnaire. *Education and Information Technologies, 25*(1), 175-191. https://doi.org/10.1007/s10639-019-09932-0

AbdElRahem, O., Bahaa-Eldin, A. M., and Taha, A. (2016, December 20-21). *Virtualization security: A survey* [Conference presentation]. 2016 11th International Conference on Computer Engineering & Systems (ICCES), Cairo, Egypt. https://doi.org/10.1109/ICCES.2016.7821971

Abdulhamid, S. M., Latiff, M. S. A., and Bashir, M. B. (2014). On-demand grid provisioning using cloud infrastructures and related virtualization tools: A survey and taxonomy. *arXiv preprint*. https://doi.org/10.48550/arXiv.1402.0696

Abeni, L., and Faggioli, D. (2020). Using Xen and KVM as real-time hypervisors. *Journal of Systems Architecture, 106*, 101709. https://doi.org/10.1016/j.sysarc.2020.101709

Aceto, G., Botta, A., de Donato, W., and Pescapè, A. (2013). Cloud monitoring: A survey. *Computer Networks, 57*(9), 2093-2115. https://doi.org/10.1016/j.comnet.2013.04.001

Adams, K., and Agesen, O. (2006). A comparison of software and hardware techniques for x86 virtualization. *ACM SIGARCH Computer Architecture News, 34*(5), 2-13. https://doi.org/10.1145/1168919.1168860

Ameen, R. Y., and Hamo, A. Y. (2013). Survey of server virtualization. *arXiv preprint*. https://doi.org/10.48550/arXiv.1304.3557

Annapareddy, N. D. R. (2011). *An approach to storage virtualization*. Texas A & M University-Kingsville.

Apple Inc. (2009). *Universal binary programming guidelines*. https://web.archive.org/web/20120327121744/http://developer.apple.com/legacy/mac/library/documentation/MacOSX/Conceptual/universal_binary/universal_binary.pdf

Armand, F., and Gien, M. (2009, January 10-13). *A practical look at micro-kernels and virtual machine monitors* [Conference presentation]. 2009 6th IEEE Consumer Communications and Networking Conference, Las Vegas, NV, USA.. https://doi.org/10.1109/CCNC.2009.4784874

Bala, V., Duesterwald, E., and Banerjia, S. (2011). Dynamo: a transparent dynamic optimization system. *ACM SIGPLAN Notices, 46*(4), 41-52. https://doi.org/10.1145/1988042.1988044

Balis, B., Antonelli, L., Bracciali, A., Gruber, T., Hyun-Wook, J., Kuhn, M., Scott, S., Unat, D., Wyrzykowski, R., and Eiling, N. (2021, August 24-25). *An open-source virtualization layer for CUDA applications* [Conference presentation]. Euro-par 2020: Parallel Processing, Warsaw, Poland.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. (2003). Xen and the art of virtualization. In ACM (Eds.), *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles* (pp. 164-177). ACM. https://doi.org/10.1145/945445.945462

Beronić, D., Pufek, P., Mihaljević, B., and Radovan, A. (2021). *On analyzing virtual threads – A structured concurrency model for scalable applications on the JVM* [Conference presentation]. 2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO). Opatija, Croatia. https://doi.org/10.23919/MIPRO52101.2021.9596855

Biederman, E. W. (2006). *Multiple instances of the Global Linux Namespaces* [Conference presentation]. Linux Symposium, Ottawa, Ontario, Canada.

Bochs (2021). *bochs. The cross Platform IA-32 Emulator*. https://bochs.sourceforge.io

Bugnion, E., Nieh, J., Tsafrir, D., and Martonosi, M. (2017). *Hardware and software support for virtualization*. Morgan & Claypool. https://doi.org/10.2200/S00754ED1V01Y201701CAC038

Cafaro, M., and Aloisio, G. (2011). Grids, clouds, and virtualization. In M. Cafaro and G. Aloisio (Eds.), *Grids, Clouds and Virtualization* (pp. 1-21). Springer.

Canonical Ltd. (2021). *Container and virtualization tools*. https://linuxcontainers.org

Chang, Y. T. S., Heistand, S., Hood, R., and Jin, H. (2021, November 14). *Feasibility of running singularity containers with hybrid MPI on NASA high-end computing resources* [Conference presentation]. 2021 3rd International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC), St. Louis, MO, USA. https://doi.org/10.1109/CANOPIEHPC54579.2021.00007

Chernoff, A., Herdeg, M., Hookway, R., Reeve, C., Rubin, N., Tye, T., Bharadwaj Yadavalli, S., and Yates, J. (1998). FX! 32: A profile-directed binary translator. *IEEE Micro, 18*(2), 56-64. https://doi.org/10.1109/40.671403

Díaz, E., Mateos, R., Bueno, E. J., and Nieto, R. (2021). Enabling parallelized-QEMU for hardware/software co-simulation virtual platforms. *Electronics, 10*(6), 759. https://doi.org/10.3390/electronics10060759

Dike, J. (2006). *User mode linux* (vol. 2). Prentice Hall Englewood Cliffs.

Docker (2022). *Docker*. https://www.docker.com

El-Anani, B. R. (2021). *Server virtualization: Para- and full virtualization: XenServer vs. KVM*. https://www.theseus.fi/handle/10024/507277

Ensim (2022). *Ensim*. http://www.ensim.com

eWeek. (2008). *IBM acquiring transitive to increase virtualization capabilities of power systems*. https://www.eweek.com/virtualization/ibm-acquiring-transitive-to-increase-virtualization-capabilities-of-power-systems/

Fareghzadeh, N., Seyyedi, M. A., and Mohsenzadeh, M. (2019). Toward holistic performance management in clouds: taxonomy, challenges and opportunities. *Journal of Supercomputing, 75*(1), 272-313. http://doi.org/10.1007/s11227-018-2679-9

Fisher, M., Sharma, S., Lai, R., and Moroney, L. (2006). *Java EE and .NET interoperability: Integration strategies, patterns, and best practices*. Prentice Hall Professional.

Ford, B., and Cox, R. (2008). *Vx32: Lightweight User-level Sandboxing on the x86* [Conference presentation]. USENIX Annual Technical Conference. https://www.usenix.org/legacy/events/usenix08/tech/full_papers/ford/ford.pdf

Gibson, C. (2007). WPAR Power AIX workload partition explained. *IBM Systems Magazine*. http://www.ibmsystemsmag.com/opensystems/december07/coverstory/18606p1.aspx

Goldberg, R. P. (1973). *Architectural principles for virtual computer systems*. Defense Technical Information Center.

Goldberg, R. P. (1974). Survey of virtual machine research. *Computer, 7*(6), 34-45. https://doi.org/10.1109/MC.1974.6323581

Honeycutt, J. (2003). *Microsoft virtual PC 2004 technical overview*. Microsoft.

Hoopes, J. (2009). *Virtualization for security: Including sandboxing, disaster recovery, high availability, forensic analysis, and honeypotting*. Syngress.

Hui, L. Y., and Seok, K. H. (2014). A study of savings of power consumption and server space through integrated virtualization of UNIX servers. *International Journal of Software Engineering and Its Applications, 8*(5), 219-230. http://dx.doi.org/10.14257/ijseia.2014.8.5.17

IBM (2008). *Transitive.* https://www-03.ibm.com/press/us/en/pressrelease/26106.wss

Jason, K., Velte, A., and Velte, T. (2009). *Microsoft virtualization with Hyper-V*. McGraw-Hill, Inc.

Jing, S.-Y., Ali, S., She, K., and Zhong, Y. (2013). State-of-the-art research study for green cloud computing. *The Journal of Supercomputing, 65*(1), 445-468. https://doi.org/10.1007/s11227-011-0722-1

Jones, M., Kepner, J., Orchard, B., Reuther, A., Arcand, W., Bestor, D., Bergeron, B., Byun, C., Gadepally, V., Houle, M., Hubbell, M., Klein, A., Milechin, L., Mullen, J., Prout, A., Rosa, A., Samsi, S., Yee, C., and Michaleas, P. (2018, September 25-27). *Interactive launch of 16,000 Microsoft Windows Instances on a supercomputer* [Conference presentation]. 2018 IEEE High Performance extreme Computing Conference (HPEC), Waltham, MA, USA. https://doi.org/10.1109/HPEC.2018.8547782

Kamp, P.-H., and Watson, R. N. (2000). *Jails: Confining the omnipotent root* [Conference presentation]. 2nd International SANE Conference. https://papers.freebsd.org/2000/phk-jails/

Kampert, P. (2010). *A taxonomy of virtualization technologies.* [Master's thesis, Delft University of Technology]. https://d1rkab7tlqy5f1.cloudfront.net/TBM/Over%20faculteit/Afdelingen/Engineering%20Systems%20and%20Services/People/Professors%20emeriti/Jan%20van%20den%20Berg/MasterPhdThesis/Masters_Thesis_Paulus_Kampert_August_2010-2.pdf

Kusnetzky, D. (2011). *Virtualization: A manager's guide*. O'Reilly Media, Inc.

KVM (2021). *Kernel Virtual Machine*. https://www.linux-kvm.org

Li, B., Shu, J., and Zheng, W. (2005). Design and implementation of a storage virtualization system based on SCSI target simulator in SAN. *Tsinghua Science and Technology, 10*(1), 122-127. https://doi.org/10.1016/S1007-0214(05)70018-3

Li, X.-F. (2016). *Advanced design and implementation of virtual machines*. CRC Press.

Li, Z. (2021, November 12-14). *Comparison between common virtualization solutions: VMware Workstation, Hyper-V and Docker* [Conference presentation]. 2021 IEEE 3rd International Conference on Frontiers Technology of Information and Computer (ICFTIC), Greenville, SC, USA. https://doi.org/10.1109/ICFTIC54370.2021.9647226

Lin, Q., Qi, Z., Wu, J., Dong, Y., and Guan, H. (2012). Optimizing virtual machines using hybrid virtualization. *Journal of Systems and Software, 85*(11), 2593-2603. https://doi.org/10.1016/j.jss.2012.05.093

Lin, S., Hao, C., and Jianhua, S. (2009, May 23-29). *vCUDA: GPU accelerated high performance computing in virtual machines* [Conference presentation]. 2009 IEEE International Symposium on Parallel & Distributed Processing, Rome, Italy. https://doi.org/10.1109/IPDPS.2009.5161020

Lindholm, T., Yellin, F., Bracha, G., and Buckley, A. (1997). *The Java virtual machine specification*. Addison-Wesley.

Linux-VServer (2018). *Linux-VServer*. http://www.linux-vserver.org

Lxrun (2022, 2008/03/09). *Official lxrun web site*. https://web.archive.org/web/20151025205205/http://www.ugcs.caltech.edu/~steven/lxrun/

Magnusson, P. S., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson, F., Moested, A., and Werner, B. (2002). Simics: A full system simulation platform. *Computer, 35*(2), 50-58. https://doi.org/10.1109/2.982916

Mann, A. (2006). *Virtualization 101: Technologies, benefits, and challenges*. Enterprise Management Associates, Inc.

Matthews, J. N., Dow, E. M., Deshane, T., Hu, W., Bongio, J., Wilbur, P. F., and Johnson, B. (2008). *Running Xen: A hands-on guide to the art of virtualization*. Prentice Hall PTR.

Nanda, S., and Chiueh, T.-C. (2005). *A survey on virtualization technologies*. Stony Brook University. http://comet.lehman.cuny.edu/cocchi/CMP464/papers/VirtualizationSurveyTR179.pdf

Nanda, S., Li, W., Lam, L.-C., and Chiueh, T.-C. (2006). *BIRD: Binary interpretation using runtime disassembly* [Conference presentation]. 2006 International Symposium on Code Generation and Optimization, New York, NY, USA. https://doi.org/10.1109/CGO.2006.6

OpenVZ (2021). *OpenVZ*. https://openvz.org

Oracle (2018). *WABI* https://docs.oracle.com/cd/E19957-01/802-6306/802-6306.pdf

Oracle (2021a). *Oracle Solaris Zones*. https://docs.oracle.com/cd/E18440_01/doc.111/e18415/chapter_zones.htm#OPCUG426

Oracle (2021b). *Oracle Virtual Box*. https://www.oracle.com/virtualization/virtualbox/.

Parallels (2021). *Parallels*. https://www.parallels.com

Parrot (2022). *Parrot*. http://www.parrot.org

Pék, G., Buttyán, L., and Bencsáth, B. (2013). A survey of security issues in hardware virtualization. *ACM Computing Surveys (CSUR), 45*(3), 40. https://doi.org/10.1145/2480741.2480757

Pessolani, P., and Jara, O. (2011, November 7-11). *Minix over Linux: A user-space multiserver operating system* [Conference presentation]. 2011 Brazilian Symposium on Computing System Engineering, Florianopolis, Brazil. https://doi.org/10.1109/SBESC.2011.17

Pessolani, P., Gonnet, S. M., Tinetti, F. G., and Cortes, T. (2012). *Sistema de virtualización con recursos distribuidos* [Conference presentation]. XIV Workshop de Investigadores en Ciencias de la Computación. http://sedici.unlp.edu.ar/handle/10915/18375

Plex86 (2021). *The new Plex86, x86 Virtual Machine Project*. http://plex86.sourceforge.net

Popek, G. J., and Goldberg, R. P. (1974). Formal requirements for virtualizable third generation architectures. *Communications of the ACM, 17*(7), 412-421. https://doi.org/10.1145/361011.361073

QEMU (2021). *QEMU, the FAST! processor emulator*. https://www.qemu.org

Ranjith, D., Tamizharasi, G. S., and Balamurugan, B. (2017, April 20-22). *A survey on current trends to future trends in green computing* [Conference presentation]. 2017 International Conference of Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India. https://doi.org/10.1109/ICECA.2017.8203616

Ryding, C., and Johansson, R. (2020). *Jails vs Docker: A performance comparison of different container technologies* [Undergraduate thesis, Mid Sweden University]. http://urn.kb.se/resolve?urn=urn:nbn:se:miun:diva-39517

Sahoo, J., Mohapatra, S., and Lath, R. (2010, April 23-25). *Virtualization: A survey on concepts, taxonomy and associated security issues* [Conference presentation]. 2010 Second International Conference on Computer and Network Technology, Bangkok, Thailand. https://doi.org/10.1109/ICCNT.2010.49

Samireh, J., and Claes, W. (2012). *Systematic literature studies: database searches vs. backward snowballing* [Conference presentation]. ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, Lund, Sweden. https://doi.org/10.1145/2372251.2372257

SCOPE Alliance (2008). *Virtualization: State of the art*. https://profsandhu.com/cs6393_s14/SCOPE-Virtualization-StateofTheArt-Version-1.0.pdf

Sehgal, N. K., and Bhatt, P. C. (2018). *Cloud computing*. Springer.

Silberschatz, A., Galvin, P. B., and Gagne, G. (2014). *Operating system concepts essentials*. John Wiley & Sons, Inc.

Smith, J. E., and Nair, R. (2005). The architecture of virtual machines. *Computer, 38*(5), 32-38. https://doi.org/10.1109/MC.2005.173

Stallings, W. (2015). *Operating systems: Internals and design principles* (9th ed.). Pearson.

Sylabs.io (2022). *Sylabs.io Singularity*. https://www.sylabs.io

Syrewicze, A., and Siddaway, R. (2018). *Pro Microsoft Hyper-V 2019: Practical guidance and hands-on labs*. Apress.

Tfrifonov, D. V., Hristo. (2018). Virtualization and containerization systems for Big Data. *Fundamental Science and Applications, 24*, 129-132. Thai, T. L., and Lam, H. (2003). *.NET framework essentials*. O'Reilly Media, Inc.

Thathera, H., Shashi, H., and Rajput, D. S. (2015). Green computing: An earth friendly system. *International Journal of Science and Research, 8*(4), 25540-25550 https://research.vit.ac.in/publication/green-computing-an-earth-friendly-system

User-Mode Linux (2022). *The user-mode Linux Kernel home page*. http://user-mode-linux.sourceforge.net

Varasteh, A., and Goudarzi, M. (2017). Server consolidation techniques in virtualized data centers: A survey. *IEEE Systems Journal, 11*(2), 772-783. https://doi.org/10.1109/JSYST.2015.2458273

Virtuozzo (2022). *Virtuozzo*. https://www.virtuozzo.com

VMware (2022). *VMware*. http://www.vmware.com

von Hagen, W. (2008). *Professional Xen virtualization*. John Wiley & Sons, Inc.

Waldspurger, C. A. (2002). Memory resource management in VMware ESX server. *ACM SIGOPS Operating Systems Review, 36*(SI), 181-194. https://doi.org/10.1145/844128.844146

Wessman, N. J., Malatesta, F., Andersson, J., Gomez, P., Masmano, M., Nicolau, V., Rhun, J. L., Cabo, G., Bas, F., Lorenzo, R., Sala, O., Trilla, D., and Abella, J. (2021, August 23-27). *De-RISC: The first RISC-V space-grade platform for safety-critical systems* [Conference presentation]. 2021 IEEE Space Computing Conference (SCC), Laurel, MD, USA. https://doi.org/10.1109/SCC49971.2021.00010

Whitaker, A., Shaw, M., and Gribble, S. D. (2002). *Denali: Lightweight virtual machines for distributed and networked applications*. http://web.cs.ucla.edu/~miodrag/cs259-security/whitaker02denali.pdf

White, J., and Pilbeam, A. (2010). A survey of virtualization technologies with performance testing. *arXiv preprint*. https://doi.org/10.48550/arXiv.1010.3233

Wine (2022). *Wine*. https://www.winehq.org

Xen Cambridge (2022). *Xen Cambridge. The virtual machine monitor*. https://www.cl.cam.ac.uk/research/srg/netos/projects/archive/xen/

Xen Project (2022). *Xen Project Website*. https://www.xenproject.org

Xtratum (2022). *Xtratum*. http://www.xtratum.org

Yee, B., Sehr, D., Dardyk, G., Chen, J. B., Muth, R., Ormandy, T., Okasaka, S., Narula, N., and Fullagar, N. (2009). *Native client: A sandbox for portable, untrusted x86 native code* [Conference presentation]. 2009 30th IEEE Symposium on Security and Privacy, Oakland, CA, USA. https://doi.org/10.1109/SP.2009.25

Luis E. Sepúlveda-Rodríguez, Julio C. Chavarro-Porras,
John A. Sanabria-Ordonez, Harold E. Castro,
Jeanna Matthews

A Survey of Virtualization Technologies: Towards a New Taxonomic Proposal
Una revisión de las tecnologías de virtualización: hacia una nueva propuesta taxonómica

Ingeniería e Investigación
vol. 42, no. 3, e214, 2022
Facultad de Ingeniería, Universidad Nacional de Colombia.,
ISSN: 0120-5609
ISSN-E: 2248-8723

DOI: https://doi.org/10.15446/ing.investig.97363