METHOD FOR UPGRADING VIRTUALIZATION COMPONENT AND METHOD FOR SAVING STATE OF VIRTUALIZED INSTANCE, PHYSICAL MACHINE, STORAGE MEDIUM, AND PROGRAM PRODUCT

TECHNICAL FIELD

The present disclosure relates to the field of cloud computing technologies, and in particular, to a method for upgrading a virtualization component and a method for saving a state of a virtualized instance, a physical machine, a storage medium, and a program product.

BACKGROUND OF THE INVENTION

A virtualization technology is a key technology of cloud computing. Various resources of a physical machine are virtualized by using the virtualization technology, enabling a plurality of Virtual Machines (VMs) to run on a same physical machine, and implementing resources sharing while not affecting each other.

To implement virtualization, a virtual machine monitor (VMM) is deployed on the physical machine. The VMM is considered as an Operating System (OS) generated for virtualization, and is responsible for virtualization of hardware resources such as a Central Processing Unit (CPU) and a memory, and creation and management of the VM.

The VMM includes a plurality of types of components related to virtualization, for example, a plurality of components used for virtualization of physical resources, and a plurality of components configured for VM creation and management, and these components are referred to as virtualization components for short. In an actual application, for problem fixing or function upgrade, the virtualization components need to be upgraded.

However, an upgrade solution in the existing technology is not suitable for the virtualization components. Therefore, a solution for upgrading the virtualization components is urgently needed.

SUMMARY OF THE INVENTION

A plurality of aspects of the present disclosure provide a method for upgrading a virtualization component and a method for saving a state of a virtualized instance, a physical

machine, a storage medium, and a program product, to reduce impact of upgrade of the virtualization component on the virtualized instance.

Some embodiments of the present disclosure provide a method for upgrading a virtualization component, the method includes the following steps. A target virtualization component is determined, where the target virtualization component is a virtualization component to be upgraded in a virtual machine monitor running on a physical machine. A state of a virtualized instance running on the physical machine is saved, where the virtualized instance is created by the virtual machine monitor. An uninstallation operation on the target virtualization component that is not upgraded is performed, and the target virtualization component that is upgraded is reinstalled and loaded. And the state of the virtualized instance is restored based on state information of the virtualized instance.

Some embodiments of the present disclosure further provide a physical machine, including: hardware resources, where a virtual machine monitor, a virtualized instance, and an upgrade program run on the hardware resources. And the virtual machine monitor includes a plurality of virtualization components, configured to virtualize the hardware resources, and create and manage the virtualized instance. The upgrade program is configured to: determine a target virtualization component to be upgraded in the plurality of virtualization components; save a state of the virtualized instance; perform an uninstallation operation on the target virtualization component that is not upgraded, and reinstall and load the target virtualization component that is upgraded; and restore the state of the virtualized instance based on state information of the virtualized instance.

Some embodiments of the present disclosure provide a method for saving a state of a virtualized instance, the method includes the following steps. State data of a virtualized instance is collected in response to a state saving operation on the virtualized instance, and the state data is saved to a specified file, where the virtualized instance corresponds to a target memory space, the target memory space includes a system memory space, and first memory data generated by the virtualized instance is stored in the system memory space in a file form, and the first memory data is retained in place in a file sharing manner before the virtualized instance is restored.

Some embodiments of the present disclosure further provide a physical machine, including hardware resources, and the hardware resources include a memory and a processor; and the memory is configured to store a computer program, and the processor is coupled with the memory and configured to execute the computer program, to implement the steps in the foregoing method.

Some embodiments of the present disclosure further provide a computer-readable storage medium storing a computer program, where the computer program, when executed by a processor, causes the processor to implement the steps in the forgoing method.

Some embodiments of the present disclosure further provide a computer program product, and the computer program product includes a computer program or computer instructions, where the computer program or computer instructions, when executed by a processor, causes the processor to implement the steps in the forgoing method.

In the embodiments of the present disclosure, a method for upgrading a virtualization component is provided. In the method, a state of a virtualized instance is first saved, then a virtualization component that needs to be upgraded is uninstalled, and the virtualization component that is upgraded is reinstalled and loaded; and the state of the virtualized instance is restored. Uninstalling the virtualization component and then reinstalling and loading the virtualization component that is upgraded is overall replacement of a level of the virtualization component. Therefore, it is not limited to complexity of upgrade, meets upgrade requirements of various complexity levels, and is applicable to upgrade of various virtualization components. In addition, before the virtualization component is uninstalled, the state of the virtualized instance relying on the virtualization component for running is saved in advance, and after upgrade of the virtualization component is completed, the state of the virtualized instance is restored, so that deployment of the virtualized instance is completed quickly, thereby helping reduce impact of the upgrade process of the virtualization component on the virtualized instance.

BRIEF DESCRIPTION OF DRAWINGS

The accompanying drawings described herein are used for providing further understanding of the present disclosure and constitute a part of the present disclosure. The illustrative embodiments of the present disclosure and descriptions thereof are used for explaining the

present disclosure, and do not constitute an improper limitation to the present disclosure. In the accompanying drawings:

FIG. 1a is a structural schematic diagram of a physical machine according to some exemplary embodiments of the present disclosure.

FIG. 1b is a structural schematic diagram of a physical machine on which virtualization is implemented through combination of QEMU-KVM according to some exemplary embodiments of the present disclosure.

FIG. 2 is a schematic flowchart of a method for upgrading a virtualization component according to some exemplary embodiments of the present disclosure.

FIG. 3a is a structural schematic diagram of a physical machine according to some exemplary embodiments of the present disclosure.

FIG. 3b is a structural schematic diagram of another physical machine according to some exemplary embodiments of the present disclosure.

FIG. 4 is a schematic flowchart of a method for saving a state of a virtualized instance according to some exemplary embodiments of the present disclosure.

FIG. 5 is a structural schematic diagram of an apparatus for upgrading a virtualization component according to some exemplary embodiments of the present disclosure.

FIG. 6 is a structural schematic diagram of an apparatus for saving a state of a virtualized instance according to some exemplary embodiments of the present disclosure.

FIG. 7 is a structural schematic diagram of another physical machine according to some exemplary embodiments of the present disclosure.

DETAILED DESCRIPTION OF THE INVENTION

To make the objectives, technical solutions, and advantages of the present disclosure clearer, the following clearly and completely describes the technical solutions of the present disclosure with reference to specific embodiments of the present disclosure and the accompanying drawings. Apparently, the described embodiments are some rather than all of the embodiments of the present disclosure. All other embodiments obtained by a person

having ordinary skill in the art based on the embodiments of the present disclosure without creative efforts shall fall within the protection scope of the present disclosure.

It is noted that, user information (including but not limited to user equipment information, user personal information, and the like) and data (including but not limited to data for analysis, stored data, displayed data, and the like) involved in the present disclosure are all authorized by the user or information and data fully authorized by all parties. The collection, use, and processing of relevant data need to conform to relevant laws, regulations, and standards of relevant countries and regions, and corresponding operation portals are provided for the user to choose to authorize or refuse. In addition, various models (including but not limited to a language model or a large model) involved in the present disclosure comply with related laws and standards.

To implement virtualization, a virtual machine monitor is deployed on a physical machine, and the virtual machine monitor includes a plurality of components related to virtualization processing, which are referred to as virtualization components for short. In an actual application, when it comes to fixing problems or upgrading functions of the virtualization component, there is an issue of upgrading the virtualization component in the virtual machine monitor.

In the existing technology, some kernels provide a hotfix mechanism. As a hotfix mechanism provided by a kernel, Kpatch performs hot fixing on a kernel module without being perceived by a user. In theory, when there are security holes, bugs, or functional upgrades in the kernel module, Kpatch is adopted for upgrade. Theoretically, hot upgrade is also performed on various virtualization components in the virtual machine monitor in this manner. However, Kpatch exists some limitations. As a hotfix mechanism, Kpatch is more applicable to a relatively simple fix case. For a relatively complex case such as a relatively large change in a data structure, a change in a name of a data object, or introduction of a new data object, such issues cannot be resolved by patching alone. This also means that Kpatch is not applicable to a relatively complex update case. However, in an actual application, the virtualization components are not only diverse but also usually have relatively complex functions, and problem fixes or functional upgrades are also quite diverse and complex.

Therefore, other means are needed to solve an upgrade problem of the virtualization component.

In the embodiments of the present disclosure, a method for upgrading a virtualization component is provided. In the method, a state of a virtualized instance is first saved, then a virtualization component that needs to be upgraded is uninstalled, and the virtualization component that is upgraded is reinstalled and loaded; and the state of the virtualized instance is restored. Uninstalling the virtualization component and then reinstalling and loading the virtualization component that is upgraded is overall replacement of a level of the virtualization component. Therefore, there is no limitation similar to using Kpatch for hot fixing. That is, this solution is not limited to complexity of upgrade of the virtualization component, meets upgrade requirements of various complexity levels, and is applicable to upgrade of various virtualization components. In addition, before the virtualization component is uninstalled, the state of the virtualized instance relying on the virtualization component for running is saved in advance, and after upgrade of the virtualization component is completed, the state of the virtualized instance is restored, so that deployment of the virtualized instance is completed quickly, thereby helping reduce impact of upgrade of the virtualization component on the virtualized instance.

Further, in the embodiments of the present disclosure, in the process of saving and restoring the state of the virtualized instance, memory data is retained in place, to reduce a copying operation on the memory data, reduce time consumption of state saving and time consumption of state restoration, and improve overall restoration efficiency of the virtualized instance. As far as possible, the process of saving and restoring the state of the virtualized instance is made user-unaware or within a time range that users tolerate, to achieve the effect of hot upgrade.

Further, in the embodiments of the present disclosure, during state saving and state restoration of the virtualized instance, when there are a plurality of virtualized instances, states of the plurality of virtualized instances are saved and restored in a concurrent operation manner, which is conducive to further improving restoration efficiency of the virtualized instance, and achieving the effect of hot upgrade as much as possible.

Further, in the embodiments of the present disclosure, when a driver of a target physical peripheral device used by the virtualized instance needs to be upgraded, an uninstallation process, an installation process, a loading process, or an initialization process of the driver is further optimized, to improve efficiency of the uninstallation process, the installation process, or the loading process, further reduce time consumption of the entire upgrade process, improve upgrade efficiency, and reduce impact of upgrade of the virtualization component on the virtualized instance.

The following describes in detail the technical solutions provided in the embodiments of the present disclosure with reference to the accompanying drawings.

FIG. 1a is a structural schematic diagram of a physical machine according to some exemplary embodiments of the present disclosure. As shown in FIG. 1, in some embodiments of the present disclosure, the physical machine includes hardware resources 11, a virtual machine monitor 12 runs on the hardware resources 11, at least one virtualized instance runs on the virtual machine monitor 12, and the at least one virtualized instance is created and maintained by the virtual machine monitor 12.

The hardware resources of the physical machine include a CPU, a memory, a network adapter, a GPU (Graphics Processing Unit), and the like. In addition, the hardware resources of the physical machine further include a memory unit, a communication component, a display, a power supply component, an audio component, and various peripheral devices, which are not shown in FIG. 1a.

On the one hand, the virtual machine monitor 12 virtualizes various resources on the physical machine, and on the other hand, the virtual machine monitor 12 is responsible for creating and managing the virtualized instance. In addition, the virtual machine monitor 12 maintains a high-efficient and isolated running environment for the virtualized instance. In addition, the virtual machine monitor 12 is further used as a core task monitoring system of the virtualized instance, and is used for task scheduling, load balancing, reporting a software and hardware failure of the virtualized instance to a manager, controlling a use status of the virtualized instance, and the like. One virtual machine monitor 12 simultaneously manages at least one virtualized instance, which is not limited.

As shown in FIG. 1a, the virtual machine monitor 12 includes a virtualization module 121. On the one hand, the virtualization module 121 virtualizes various resources on the physical machine, and on the other hand, the virtualization module 121 is responsible for creating and managing the virtualized instance, and at least includes a component used for virtualization, for example, a component used for CPU virtualization, a component configured for memory virtualization, a component configured for input/output (IO) virtualization, or a component configured for VM creation and management.

For ease of unified description of components or modules related to virtualization, a conventional OS part of the physical machine is referred to as a host OS 122, and descriptions are provided by using an example in which the host OS belongs to the virtual machine monitor 12. That is, in some embodiments of the present disclosure, the virtual machine monitor includes the virtualization module 121 and the host OS 122, as shown in FIG. 1a. The host OS is mainly responsible for device management, file management, memory management, process management, and the like of the physical machine, and includes a device driver, a file system, a memory management module, a process management module, and the like. Some functions in the host OS are also related to virtualization. For example, during IO virtualization, a driver of the physical peripheral device (which is also referred to as an IO device) is involved, and the driver of the physical peripheral device belongs to the host OS.

For ease of description, in some embodiments of the present disclosure, all virtualization-related objects in the virtual machine monitor 12 are collectively referred to as virtualization components. These virtualization components include components in the virtualization module 121, and also include a virtualization-related driver in the host OS 122.

An implementation of the virtual machine monitor 12 is not limited in some embodiments of the present disclosure. For example, the virtual machine monitor 12 is implemented as Xen, OpenVZ, or a Kernel-based Virtual Machine (KVM), or is implemented as a combination of the KVM and a Quick EMUlator (QEMU, that is a virtual operating system simulator). The Xen is an open source virtual machine manager, directly runs on hardware resources of the physical machine to replace a software layer of an operating system, allows a plurality of operating system instances to run on a single physical machine, is configured to provide

high-performance virtualization, and does not rely on special hardware support. The OpenVZ is an operating-system-level virtualization technology (or referred to as a containerization technology) based on a Linux (an open-source UNIX-like operating system) kernel, allows the physical machine to run a plurality of operating systems, and provides a more lightweight virtualization solution. The KVM is a Linux-based open source virtualization technology. Specifically, the KVM converts Linux into a virtual machine monitor, to provide capabilities such as CPU virtualization and memory management for the virtual machine, and the KVM uses a hardware virtualization technology to improve performance of the virtual machine. The KVM and the QEMU are used in combination. The QEMU is virtualization software based on a user mode. The KVM relies on QEMU to simulate the IO device such as a virtual network adapter or a virtual disk controller. The combination of the KVM and the QEMU (referred to as QEMU-KVM) provides a complete virtualization solution.

When the virtual machine monitor uses different implementations, virtualization components included in the virtual machine monitor are slightly different. In some embodiments, an example in which the virtual machine monitor is implemented by combining the QEMU and the KVM is used to describe the virtualization component included in the virtual machine monitor, but it does not represent that the virtual machine monitor in some embodiments of the present disclosure is implemented through only combination of QEMU-KVM.

FIG. 1b is a structural schematic diagram of a physical machine on which virtualization is implemented through combination of QEMU-KVM according to some exemplary embodiments of the present disclosure. An internal software and hardware architecture of the physical machine includes, from top to bottom, an application layer, a kernel layer, and a hardware layer. The hardware layer includes various hardware resources 11 of the physical machine. As shown in FIG. 1b, the hardware resources 11 of the physical machine include, but are not limited to, a CPU 111, a GPU 112, a Random-access memory (RAM) 113, a magnetic disk 114, and the like. In addition, the hardware resources of the physical machine further include a network adapter module, a communication component, a display, a power supply component, an audio component, and various peripheral devices, which are not shown in FIG. 1b. The kernel layer runs on the hardware resources 11. The host OS 122 is located at the kernel layer. The host OS 122 includes drivers of various hardware in the hardware layer, for example, drivers of physical peripheral devices (which are referred to as peripherals for

short) such as a Network Interface Card (NIC) and a Graphics Processing Unit (GPU). In some embodiments of the present disclosure, the physical peripheral device is relative to the CPU 111 of the host. All hardware modules interconnected to the CPU 111 by using various types of buses are used as the physical peripheral devices in some embodiments of the present disclosure, and are also referred to as physical IO devices. A bus interconnected to the CPU 111 is a peripheral component interconnect express (PCIe) bus, a serial bus, or the like, which is not limited, and is specifically determined based on a physical peripheral device.

In some embodiments, the kernel layer further includes various kernel-mode virtualization components such as a KVM driver, a VFIO driver, a VFIO_PCI driver, and a VFIO_MDEV driver. The application layer includes a QEMU. The QEMU belongs to a user mode virtualization component, and coordinates with a kernel mode virtualization component to complete virtualization of various resources of the physical machine and creation and management of the virtualized instance. The virtualization module 121 spans the kernel layer and the application layer. As shown in FIG. 1b, the virtualization module 121 includes the kernel-mode virtualization components such as the VFIO driver, the VFIO_PCI driver, the VFIO_MDEV driver, and the KVM driver at the kernel layer, and the QEMU at the application layer. The application layer further includes various virtualized instances running on the physical machine. The virtualized instances is a cloud host, an elastic compute service (ECS), a virtual machine, a container, or the like, and these virtualized instances are created by the virtualization module 121.

The KVM driver is a part of a Linux kernel, and runs as a kernel module, and a main function is responsible for managing the virtualized instance, implementing CPU virtualization and memory virtualization, and the like. As a user space component, QEMU is mainly responsible for simulating various hardware devices, and providing an I/O device model and a peripheral access path for the virtualized instance. A function of the VFIO driver is to securely map the physical peripheral device to a user space, so that a user space process directly accesses the physical peripheral device by using the VFIO driver. The VFIO_PCI driver is mainly responsible for directly passing through a PCI device to the virtualized instance, so that the virtual machine directly accesses configuration space of the PCI device. A main function of the VFIO_MDEV driver is to allow hardware resources to be securely

shared between different virtual machines, and ensure isolation. It is noted herein that the kernel-mode virtualization components shown in FIG. 1b are merely examples. It does not indicate that only these components are included, nor indicate that these components are necessarily to be included.

Further, when the VFIO device is used for passthrough, the virtualized instance directly accesses the physical peripheral device, but each physical peripheral device is exclusively occupied by one virtualized instance. To resolve the problem of exclusive occupation of the physical peripheral device, a Single Root I/O Virtualization (SR-IOV) passthrough technology is used to support virtualizing one Physical Function (PF) to obtain a plurality of Virtual Functions (VFs), and the plurality of VFs are passed through for different virtualized instances to use. That is, there is a related virtualization component in the virtual machine monitor that performs virtualization processing on the physical peripheral device to obtain each virtual peripheral device. The virtual peripheral device also has a driver. To distinguish the driver of the virtual peripheral device from the driver of the physical peripheral device, the driver in the host OS 122 is referred to as a PF driver, for example, including a GPU PF driver and an NIC PF driver. Certainly, the PF driver of the physical peripheral device shown herein is merely an example. It does not indicate that only these PF drivers are included, nor indicate that these PF drivers are necessarily to be included.

Various PF drivers, KVM drivers, VFIO drivers, VFIO_PCI drivers, VFIO_MDEV drivers, QEMU, and the like shown in FIG. 1b are all examples of the virtualization component in some embodiments of the present disclosure. For the foregoing physical machine, in an actual running process, some virtualization components are faulty, or some virtualization components need to be upgraded as the virtualization technology is updated and iterated. That is, due to problem fixing or function upgrade, there is a problem of upgrading the virtualization component in the virtual machine monitor.

To upgrade the virtualization component, some embodiments of the present disclosure provides a method for upgrading a virtualization component. As shown in FIG. 2, the method includes the following steps.

In step S201, a target virtualization component is determined, where the target virtualization component is a virtualization component to be upgraded in a virtual machine monitor running on a physical machine.

In step S202, a state of a virtualized instance running on the physical machine is saved, where the virtualized instance is created by the virtual machine monitor.

In step S203, an uninstallation operation on the target virtualization component that is not upgraded is performed, and the target virtualization component that is upgraded is reinstalled and loaded.

In step S204, the state of the virtualized instance is restored based on state information of the virtualized instance.

In some embodiments, the target virtualization component is at least one virtualization component in the virtual machine monitor. With reference to the architecture shown in FIG. 1b, virtualization components such as a GPU PF driver, a NIC PF driver, a QMEU, a KVM driver, or a VFIO driver are used as the target virtualization component in some embodiments. This is not limited in the present disclosure. It is noted that, the target virtualization component includes at least one virtualization component in the virtual machine monitor; and when the target virtualization component is upgraded, in response to another virtualization component that does not need to be upgraded exists in the virtual machine monitor and running of the another virtualization component does not rely on the target virtualization component, the another virtualization component that does not need to be upgraded runs normally and is not affected by the upgrade process.

In some optional embodiments, to determine the target virtualization component, an upgrade management component in a user mode is further deployed on the hardware resources of the physical machine. The upgrade management component receives a configuration file of virtualization component upgrade, the configuration file includes identifier information of a virtualization component to be upgraded, and the target virtualization component is determined based on the configuration file. Alternatively, an interactive interface configured for virtualization upgrade is displayed. Identifier information of each virtualization component, for example, a name or an icon of the each virtualization component, is displayed on the interactive interface. In response to a selection operation of a user for the

identifier information of each virtualization component, a virtualization component corresponding to selected identifier information is determined as the target virtualization component. Alternatively, an upgrade command is sent to the upgrade management component through a command window, the upgrade command includes identifier information of a virtualization component that needs to be upgraded, and the target virtualization component is determined based on the upgrade command.

In some embodiments, upgrade of the target virtualization component affects normal running of the virtualized instance. Therefore, before upgrading the target virtualization component, a state of the virtualized instance is first saved, so that after the upgrade is completed, the state of the virtualized instance is restored based on the saved state information of the virtualized instance. When the state of the virtualized instance is saved, the virtualized instance is suspended, to avoid impact caused by upgrading the target virtualization component to the virtualized instance. Virtualized instances that require state saving and being suspended are all virtualized instances running on the physical machine at that time. Certainly, some virtualized instances that are separated from the target virtualization component and that run normally also run without state saving and being suspended. In consideration of the fact that running of the virtualized instance is usually inseparable from each virtualization component, thus a relatively simple implementation is: when the target virtualization component needs to be upgraded, state saving and suspension are uniformly performed on each virtualized instance running on the physical machine at that time, and state restoration is uniformly performed on each virtualized instance after the target virtualization component is upgraded, so that each virtualized instance continues to run. To facilitate successful running of the virtualized instance relying on the target virtualization component that is upgraded after the state of the virtualized instance is restored, when the target virtualization component is upgraded, it is achieved that the target virtualization component is compatible in terms of interfaces before and after being upgraded. The interfaces include not only an interface through which the target virtualization component provides a service to the virtualized instance (or referred to as an access interface provided to the virtualized instance or an application running in the virtualized instance), but also an interface required by the target virtualization component to invoke other virtualization components or non-virtualization kernel components. In terms of interfaces, maintaining compatible before and after being

upgraded mainly relates to that an interface name, an input parameter, an output parameter, and the like of the interface remain unchanged, which neither affects invocation of the virtualized instance for the target virtualization component that is upgraded, nor affects invocation of the target virtualization component that is upgraded for other components or modules on which the target virtualization component that is upgraded originally depends.

An operation of saving the state of the virtualized instance running on the physical machine mainly includes: saving state data and memory data of the virtualized instance. The state data of the virtualized instance refers to some data on which running of the virtualized instance depends and some data that has impact on a running state of the virtualized instance, for example, state data of a currently allocated Virtual CPU (VCPU), a state of a currently accessed virtual device, a unique identifier of the virtualized instance such as a Universally Unique Identifier (UUID), and a currently accessed file handle. Specifically, the state data varies with a function of the virtualized instance. The memory data of the virtualized instance refers to data that is generated in a running process of the virtualized instance and that is stored in an allocated memory space, and is referred to as memory data for short. The memory data also vary with a function of the virtualized instance. Further, when the virtualized instance mounts a persistent storage disk (such as a cloud disk), a saving operation on the persistent storage disk is further involved. The saving operation on the persistent storage disk mainly refers to saving a mounting relationship between the persistent storage disk and the virtualized instance.

An operation of performing an uninstallation operation on the target virtualization component that is not upgraded refers to stopping a related service of the target virtualization component, deleting various resource files corresponding to the target virtualization component from the physical machine, and cleaning a related configuration file and runtime data. An operation of deleting the various resource files corresponding to the target virtualization component includes: when the target virtualization component is a virtualization component in a user mode, uninstalling an installation package of the target virtualization component in a manner similar to that of conventional application software; or when the target virtualization component is a virtualization component in a kernel mode, directly deleting the virtualization component in the kernel mode from the operating system by using some uninstallation commands. An operation of reinstalling the target virtualization

component that is upgraded refers to downloading a new version installation package (or a kernel module). When it is a virtualization component in the user mode, the installation package is run and installed in a manner similar to that of conventional application software. When it is a virtualization component in the kernel mode, a virtualization component in the kernel mode that is upgraded is added to the operating system by using some installation commands, an administrator permission, or the like. Further, for the virtualization component in the user mode, installation is completed by executing the installation package; and the virtualization component in the kernel mode needs to be compiled and added to a kernel. An operation of loading the target virtualization component that is upgraded refers to a process of starting a newly installed target virtualization component and performing initialization; for the virtualization component in the user mode, it is a process of starting a new installed software service, and completing corresponding initialization; and for the virtualization component in the kernel mode, it is a process of loading a new target virtualization component to a kernel, and completing corresponding initialization.

Correspondingly, an operation of restoring the state of the virtualized instance based on state information of the virtualized instance mainly refers to recreating the virtualized instance based on a resource file corresponding to the virtualized instance, restoring the virtualized instance to a state at a saving moment, and providing saved memory data of the virtualized instance to the virtualized instance that is restored, so that the virtualized instance continues to run.

In some embodiments of the present disclosure, an internal implementation structure of the virtual machine monitor is not limited. Any structure that creates and manages the virtualized instance is applicable to some embodiments of the present disclosure. Optionally, the virtual machine monitor includes a first virtualization component in a user mode, which is, but is not limited to, a QEMU, a virtualization control component in a user mode, or the like. The first virtualization component is in the user mode, for ease of interaction and invoking, and does not involve more security problems. Based on this, to save the state of the virtualized instance running on the physical machine, the first virtualization component is invoked to save the state of the virtualized instance. The first virtualization component obtains various state information of the virtualized instance and has permissions to access and manage memory data of the virtualized instance. Correspondingly, an operation of restoring the state

of the virtualized instance based on the state information of the virtualized instance includes invoking the first virtualization component when the target virtualization component does not include the first virtualization component, and restoring the state of the virtualized instance based on saved state information of the virtualized instance. When the target virtualization component includes the first virtualization component, after the first virtualization component is reinstalled and loaded, the first virtualization component that is upgraded is invoked, and the state of the virtualized instance is restored based on the saved state information of the virtualized instance.

It is noted herein that saving and restoring the state of the virtualized instance based on the first virtualization component in the user mode included in the virtual machine monitor is merely an example and is not limited to this. Alternatively, the state of the virtualized instance is saved and restored by using an upgrade program for performing a upgrade method provided in some embodiments of the present disclosure, provided that the upgrade program obtains a management permission on various state information and memory data of the virtualized instance. The management permission is registered and applied for with the QEMU or the KVM driver in the virtual machine monitor. Regardless of whether the upgrade program provided in some embodiments of the present disclosure or the first virtualization module (for example, the QEMU) saves and restores the state of the virtualized instance, for detailed implementations of saving and restoring the state of the virtualized instance, refer to descriptions in the following embodiments of the present disclosure.

In some embodiments of the present disclosure, an operation of saving the state of the virtualized instance, then, uninstalling the target virtualization component that needs to be upgraded, and reinstalling and loading the target virtualization component that is upgraded is overall replacement at a driver module level, is applicable to upgrade requirements of various complexity, has no limitation similar to using Kpatch for hot fixing, and upgrades various problems occurring in the virtualization component or various newly developed functions without fear that a new function cannot be applied to existing virtualized instances.

Further, in some embodiments of the present disclosure, when the target virtualization component is upgraded, it is hoped to implement hot upgrade of the target virtualization component, that is, it is hoped to complete upgrade of the target virtualization component

without the user perceiving any interruption to the application hosted on the virtualized instance, or with an interruption time or a delay being within a tolerable range, so that online problem fixing and rollout of a new function and a new feature are performed without notifying the user, thereby greatly reducing a period of fixing a security hole (referred to as BUG), and also reducing costs of communication with the user.

To achieve the foregoing objective, an entire process from saving the state data and the memory data of the virtualized instance to restoring the virtualized instance based on state data and memory data needs to be completed as quickly as possible, and consume as short a time as possible, so short that the user is basically imperceptible or within a response delay that the user tolerates. In terms of how to shorten time consumption of the foregoing process as much as possible, in some embodiments of the present disclosure, at least one of the following technical means is used.

(1) The process of saving the state data and the memory data of the virtualized instance is improved. An operation that is relatively time-consuming in the process of saving the state of the virtualized instance is analyzed, and it is determined that a more time-consuming operation is saving of the memory data. In a saving process, to save the memory data, the memory data needs to be copied to another memory or a persistent storage space (such as a disk file). When a data amount of the memory is relatively large, this operation is time-consuming. In some embodiments of the present disclosure, a technology of retaining memory data in place is used, to ensure that a memory space in which the memory data is located is not logged out or recycled, and the memory space is not modified during upgrade of the target virtualization component. In addition, in a process of restoring the virtualized instance, it is ensured that the memory space is reallocated to the virtualized instance that is restored, to complete restoration of the memory data. In this way, in an entire process, the memory data does not need to be copied, thereby saving time, and improving efficiency of upgrading the target virtualization component.

(2) An uninstallation process, an installation process, or a loading process of the target virtualization component that needs to be upgraded is optimized, to reduce a downtime of the virtualized instance caused by upgrade of the target virtualization component, reduce time consumption of the entire process, and achieve an effect of hot upgrade as much as possible.

It is noted that, no matter whether the target virtualization component is a component in the virtualization module or a PF driver in the host OS, the target virtualization component is optimized provided that there is potential room for optimization. An optimization process includes an uninstallation process, an installation process, or a loading process, or includes any two or three processes, which is not limited, and is specifically determined based on the target virtualization component.

Further, optionally, when the target virtualization component that needs to be upgraded includes a peripheral driver (that is, the PF driver in the host OS), at least one process of an uninstallation process, an installation process, a loading process, and an initialization process of the peripheral driver is optimized. Specifically, it is determined based on a physical peripheral device corresponding to the PF driver. Room for optimization of the uninstallation process, the installation process, the loading process, and the initialization process of drivers corresponding to different physical peripheral devices is different, and is related to hardware manufacturers. The uninstallation process, the installation process, the loading process, and the initialization process of hardware drivers provided by different hardware manufacturers are also different. A specific optimization process is not limited in some embodiments, and any optimization manner that shortens a time relative to a conventional uninstallation, installation, loading, or initialization process is applicable to some embodiments of the present disclosure.

(3) The state of the virtualized instance is saved and restored in a concurrent manner.

When the state of the virtualized instance running on the physical machine is saved, and when there are a plurality of virtualized instances, states of the plurality of virtualized instances are saved in a concurrent operation manner, thereby shortening a state saving time. Correspondingly, when the state of the virtualized instance running on the physical machine is restored, and when there are a plurality of virtualized instances, states of the plurality of virtualized instances are restored in a concurrent operation manner.

In some embodiments of the present disclosure, it is not limited to using the technical means of retaining the memory data in place, and a corresponding implementation is used with reference to different memory spaces in which the memory data is located.

In some optional embodiments, when a virtualized instance is created, a system memory space is allocated to the virtualized instance. The system memory space refers to a physical memory on the host, for example, a Random Access Memory (RAM). For the memory data in the system memory space, to implement retaining of the memory data in place, when the virtualized instance generates memory data, the memory data is written, in a file manner, into the system memory space allocated to the virtualized instance, rather than directly writing generated data into the system memory space. In this way, in the process of saving the state of the virtualized instance, the memory data is retained in place in a file sharing manner, to provide a basis for the memory data to be retained in the system memory space.

With reference to the system architecture shown in FIG. 1b, a detailed process of allocating the system memory space, a process of saving the state of the virtualized instance, and a process of restoring the state of the virtualized instance are described in detail.

An operation of saving the state of the virtualized instance running on the physical machine includes the following steps. State data of the virtualized instance is collected, and the state data is saved to a specified file. Optionally, the specified file is a memory file, or is a persistent file (such as a disk file). Collecting the state data relates to capturing and storing sufficient information, so that the state of the virtualized instance is restored subsequently from a same point. Optionally, on a same physical machine, each virtualized instance has a unique identifier, and the state data corresponding to the virtualized instance is collected based on the unique identifier, including, but not limited to, configuration information of various resources such as a CPU, a memory, a network device, and a storage device. In an example, based on VM instances created by the virtual machine monitor, on a same physical machine, each VM instance is endowed with identifier information, for example, a virtual machine name (Name), a UUID, and an ID (Identifier). The identifier information is used for managing, identifying, and operating a state of the virtual machine. The UUID is used as a unique identifier, and state data of a corresponding VM is obtained by using the UUID.

An operation of saving the state of the virtualized instance running on the physical machine further includes the following steps. Target memory data in a target memory space of the virtualized instance is retained in place.

In some optional embodiments, the target memory space includes a system memory space, and the target memory data includes first memory data stored in the system memory space in a file form. A file used for storing the first memory data herein is a memory mapping file created in the system memory space based on a memory mapping technology, or is a memory file created in the system memory space by using a memory file system, which is not limited. Correspondingly, an operation of retaining, in place, target memory data in a target memory space of the virtualized instance includes the following steps. The first memory data is retained in place in a file sharing manner. The file sharing herein means that the virtualized instance shares a same memory mapping file or shares a same memory file in corresponding new and old processes before and after restoration. After the state of the virtualized instance is saved, this further involves release of the old process corresponding to the virtualized instance, to suspend running of the virtualized instance. Restoring the state of the virtualized instance further involves creating the new process corresponding to the virtualized instance, to restore the virtualized instance. Specifically, by allocating or mounting an access path of the memory mapping file or the memory file to the new process, file sharing between the new process and the old process is implemented.

In some embodiments, the first memory data is retained in place in a file sharing manner. The memory data is stored and retained in a memory mapping manner, or the memory data is stored and retained in a memory file manner. A separate description is given below.

In the memory mapping manner, a memory mapping file is a mapping from a persistent file to a memory space. Through the mapping between the persistent file and the memory space, an application (including a plurality of processes) directly performs read and write operations on the memory, to modify the persistent file. In some embodiments, the virtualized instance uses the persistent file (such as a disk file) to persistently store memory data. For ease of description and distinction, the persistent file is referred to as a target persistent file. In addition, for ease of the read and write operations, in a memory mapping manner, the target persistent file is mapped to the system memory space allocated to the virtualized instance, that is, a memory mapping file is created in the system memory space. The memory mapping file is a memory mapping of the target persistent file used by the virtualized instance. The memory mapping file maps the target persistent file to the memory space, and when the

virtualized instance performs operations (for example, reading or writing) on a mapped memory area, the operations are reflected in the target persistent file that is mapped.

The target persistent file is used for persistently storing data in the memory mapping file. To reflect the operations to the target persistent file that is mapped, the target persistent file is created for the virtualized instance, and a memory mapping file corresponding to the target persistent file is created in the system memory space; the first memory data is written into the memory mapping file in a process in which the virtualized instance generates the first memory data; and the data in the memory mapping file is stored in the target persistent file according to a set persistence policy. The set persistence policy is periodical persistence, that is, periodically writing the first memory data into a persistence file based on a time interval (that is, performing a persistence operation); or the set persistence policy is idle persistence, that is, performing a persistence operation when it is detected that a CPU is idle; or performing the persistence operation each time after the virtualized instance writes the first memory data into the memory mapping file; or performing the persistence operation when the memory mapping file is cleared; or performing the persistence operation before the virtualized instance is closed or suspended. Each persistence policy has an applicable scenario, which is selected as required, and is not limited in the present disclosure.

Based on the foregoing description, an implementation of retaining the first memory data in place in a file sharing manner includes the following steps. The first memory data is stored to a memory mapping file in the system memory space when storing the first memory data, and the memory mapping file is kept from being released during upgrade of the target virtualization component, to implement retaining of the first memory data in place. And the memory mapping file is a memory mapping of a target persistent file used by the virtualized instance, and the target persistent file is used for persistently storing data in the memory mapping file.

In the memory file manner, a memory file system is provided in some operating systems. The memory file system stores data in the system memory space in the memory file manner. A memory file is not released as the virtualized instance is released, and is released only when the host is powered off. Based on this, in some embodiments of the present disclosure, another implementation of retaining the first memory data in place in a file sharing manner

includes the following steps. When storing the first memory data, a memory file system is invoked to store the first memory data in a memory file form to a system memory space allocated to the virtualized instance, so that during upgrade of the target virtualization component, the memory file is kept from being released with a feature that the memory file cannot be released, thereby implementing retaining of the first memory data in place.

Further, Linux memory management uses a “pagination mechanism”. A default size of a standard page is 4 KB, and HugeTLB provides a larger page size (for example, 2 MB or 1 GB) of the standard page. A mapped memory page is enlarged, thereby reducing a size of a page table used for memory mapping, and improving memory access efficiency. The first memory data is retained in place between corresponding new and old processes before and after the virtualized instance is restored in a Hugetlb FS manner. That is, when creating a memory file in the system memory space, the memory file system creates a memory page of a larger granularity in the Hugetlb FS manner to store the first memory data. During upgrade of the target virtualization component, with the feature that the memory file cannot be released, the first memory data is retained in place, and the size of the page table used for memory mapping is also reduced, thereby improving the memory access efficiency.

In some embodiments of the present disclosure, the manner of retaining the target memory data in the target memory space in place is not limited, and any manner capable of retaining the target memory data in place is applicable to some embodiments of the present disclosure. In addition to the foregoing manner, the first memory data is also retained in place in a manner of locking the memory space. Specifically, in a process of saving the state of the virtualized instance, the system memory space allocated to the virtualized instance is locked. For example, a lock mark is added to the system memory space. The lock mark is used for marking that the system memory space is locked in a process of suspending the virtualized instance, and the system memory space is not released or modified; and after the virtualized instance is restored, the system memory space is reallocated to the virtualized instance that is restored based on the lock mark. Further, to facilitate determining a correspondence between the lock mark and the virtualized instance, in the process of saving the state of the virtualized instance, the lock mark is first added to the system memory space of the virtualized instance, and the lock mark is added to the state data of the virtualized instance. In this way, after the virtualized instance is restored, the lock mark is obtained from the state data of the

virtualized instance, and the system memory space with the lock mark is reallocated to the virtualized instance that is restored, to implement retaining of the first memory data in place in a process of saving and restoring the state of the virtualized instance without a need of memory copying, thereby improving restoration efficiency of the virtualized instance.

Further, optionally, an operation of restoring the state of the virtualized instance based on state information of the virtualized instance includes the following steps. The virtualized instance is recreated based on a resource file corresponding to the virtualized instance, the virtualized instance reuses a unique identifier before restoration. Further, the target memory space is reallocated to the virtualized instance based on the unique identifier (for example, a UUID) of the virtualized instance, the target memory space includes target memory data retained in place when the state of the virtualized instance is saved. And based on state data in the specified file, the virtualized instance is restored to a running state during state saving. In an example, when the virtualized instance is specifically implemented as a VM, an operation of restoring the state of the virtualized instance (the new VM) based on state information of the virtualized instance (the old VM) includes the following steps. A VM (referred to as the new VM) that is recreated based on the resource file corresponding to the old VM reuses the UUID (unique identifier) of the old VM, the UUID is associated with various resource files of the old VM, and the resource files are allocated to a restored VM. Further, the target memory space is reallocated to the new VM based on the UUID; and based on the state data in the specified file, the new VM is restored to the running state during state saving.

In some optional embodiments, the physical machine includes various peripheral devices having a memory space and having a computing capability. A peripheral device herein is a peripheral device in a broad sense, and includes a device interconnected to a CPU of the physical machine via various interconnect buses (such as PCIe). In an actual application, based on an application requirement, a virtual peripheral such as a virtual GPU is created for the virtualized instance, and a memory space (such as a video memory of the GPU) is allocated. For distinction, the memory space is referred to as a peripheral memory space. In a running process of the virtualized instance, data generated when the virtual peripheral is used is stored in the peripheral memory space.

Based on this, when the memory data of the virtualized instance is stored, memory data of the virtualized instance in the peripheral memory space is further stored. In some optional embodiments, the target memory space further includes a peripheral memory space. The peripheral memory space is a memory space on a target physical peripheral device used by the virtualized instance. The target physical peripheral device is a peripheral device having a computing capability and a memory space, for example, is but is not limited to a GPU. Further, when the target memory data includes second memory data stored in the peripheral memory space of the target physical peripheral device, an operation of retaining, in place, the target memory data in the target memory space of the virtualized instance further includes retaining the second memory data in place.

An operation of retaining the second memory data in place first includes in the process of saving and restoring the state of the virtualized instance, ensuring that the second memory data in the peripheral memory space is not changed. Specifically, the process of saving the state of the virtualized instance involves release or destruction of the old process, in this process, no second memory data in the peripheral memory space is rewritten. And the process of restoring the state of the virtualized instance involves creation of the new process, in this process, it is ensured that no peripheral memory space is initialized, to ensure that the second memory data is not changed. Further, optionally, retaining of the second memory data in place varies with whether the driver of the target physical peripheral device is the target virtualization component that needs to be upgraded, which is determined based on conditions. Specifically, when the driver of the target physical peripheral device needs to be upgraded, that is, when the target virtualization component includes the driver of the target physical peripheral device, this involves a process of uninstalling, installing, and loading the driver. In the process, different manufacturers use different policies to ensure data security and device stability. For example, some manufacturers erase data when the driver is uninstalled. Some other manufacturers initialize memory when the driver is loaded. These operations affect the second memory data. Therefore, an operation of retaining the second memory data in place further includes ensuring that the second memory data does not change before and after the driver is upgraded.

Based on the foregoing description, when the target virtualization component includes the driver of the target physical peripheral device, a process of uninstalling and loading the

driver of the target physical peripheral device is improved, so that when the driver is uninstalled, the rewrite operation is prohibited from being performed on the peripheral memory space; and when the driver that is upgraded is reloaded, the initialization operation is prohibited from being performed on the peripheral memory space, to ensure that the second memory data in the peripheral memory space allocated to the virtualized instance does not change before and after the upgrade. Further, optionally, when the target virtualization component does not include the driver of the target physical peripheral device, uninstallation and reloading are not involved, and therefore, related operations are not involved.

To deal with policies of different manufacturers and ensure that the second memory data in the peripheral memory space is retained in place, the driver is improved, and a memory retention function is added to the driver. The memory retention function includes not performing a rewrite operation on the peripheral memory space in an uninstallation process and not performing an initialization operation on the peripheral memory space in a loading process. The driver that is upgraded enables the memory retention function by default. In some optional embodiments, the memory retention function of the driver is allowed to be enabled as required. That is, before the driver is uninstalled, the memory retention function of the driver is further enabled by using an interface externally exposed by the driver. When the memory retention function is enabled, the rewrite operation is not performed on the peripheral memory space in an uninstallation process, and correspondingly, the initialization operation is not performed on the peripheral memory space in a loading process. The driver that is upgraded enables the memory retention function by default.

Optionally, whether the memory retention function of the driver is enabled is controlled by transferring parameters of the kernel module. When the driver is not upgraded, the memory retention function is not enabled. When upgrade of the driver is completed, it needs to be ensured that the memory retention function is enabled. For example, during loading, the memory retention function of the driver is enabled by transferring the parameters of the kernel module. Certainly, the manner of controlling whether the memory retention function is enabled is not limited to this. For example, in a Linux system, in addition to the manner of transferring the parameters of the kernel module, enabling of the memory retention function is also controlled by using a Linux command. In a Windows system (an operating system

developed by Microsoft Corporation based on a graphical user interface), a graphical user interface is provided, and a user configures the memory retention function by using the graphical user interface, including enabling or disabling the memory retention function.

Certainly, the memory retention function of the driver is also in a normally enabled state by default. In this case, the driver is directly uninstalled, and a rewrite operation is not performed on the peripheral memory space in the uninstallation process.

Further, optionally, when the target virtualization component includes the driver of the target physical peripheral device used by the virtualized instance, before the uninstallation operation is performed on the target virtualization component that is not upgraded, an uninstallation process, an installation process, a loading process, or an initialization process of the driver is further optimized. In this way, when the driver is subsequently uninstalled, installed, loaded, or initialized, a corresponding time is shortened, and upgrade efficiency of the driver is improved.

It is noted that although the above focuses on optimizing the uninstallation process, the installation process, the loading process, or the initialization process of the driver of the target physical peripheral device, it is mainly because the uninstallation process, the installation process, the loading process, or the initialization process of the driver of the target physical peripheral device is usually relatively complex, consumes more time, and has better room for optimization. However, this is not limited thereto. For other virtualization components, when any of the uninstallation process, the installation process, the loading process, and the initialization process has space for optimization, optimization is also performed, to further shorten an upgrade time and improve efficiency. It is additionally noted that, optimization of the uninstallation process, the installation process, the loading process, or the initialization process of the target virtualization component relates to a specific manufacturer. In some embodiments of the present disclosure, no attention is paid to a specific optimization implementation solution, and any optimization manner that shortens a time compared with a conventional solution is applicable to some embodiments of the present disclosure.

Further, optionally, before the target memory data (the first memory data or the second memory data) in the target memory space is retained in place, resource information for

configuring the virtualized instance is obtained. Whether the virtualized instance uses a peripheral device is determined based on the resource information. When the peripheral device is used, the second memory data is retained in place.

Further, optionally, before the target memory data (the first memory data or the second memory data) in the target memory space is retained in place, whether the first memory data or the second memory data is greater than or equal to a corresponding threshold is determined, and when the first memory data or the second memory data is greater than the corresponding threshold, the target memory data needs to be retained in place. Optionally, the first memory data and the second memory data are separately retained in place, or are simultaneously retained in place. This is not limited in the present disclosure.

As shown in FIG. 3a, some embodiments of the present disclosure further provide a physical machine, including: hardware resources 11, where a virtual machine monitor 12, an upgrade program 13, and a virtualized instance run on the hardware resources 11; and the virtual machine monitor 12 includes a plurality of virtualization components, configured to virtualize the hardware resources 11, and create and manage the virtualized instance.

In a specific implementation, the upgrade program 13 is an upgrade script, or is an unscripted upgrade program. The upgrade program 13 is configured to: determine a target virtualization component to be upgraded in the plurality of virtualization components; save a state of the virtualized instance; perform an uninstallation operation on the target virtualization component that is not upgraded, and reinstall and load the target virtualization component that is upgraded; and restore the state of the virtualized instance based on state information of the virtualized instance.

As shown in FIG. 3b, in some optional embodiments, the plurality of virtualization components include a first virtualization component 123 in a user mode and a second virtualization component 124 in a kernel mode. Further, the target virtualization component includes at least one the first virtualization component 123 or the second virtualization component 124. Optionally, the first virtualization component 123 in the user mode is a QEMU, and the second virtualization component 124 in the kernel mode is a KVM.

In some optional embodiments, the upgrade program 13 is specifically configured to: invoke the first virtualization component 123 to save the state of the virtualized instance; and invoke

the first virtualization component 123 when the target virtualization component does not include the first virtualization component 123, to restore the state of the virtualized instance based on state information of the virtualized instance; invoke the first virtualization component 123 that is upgraded when the target virtualization component includes the first virtualization component 123, to restore the state of the virtualized instance based on the state information of the virtualized instance.

In some optional embodiments, when saving the state of the virtualized instance, the upgrade program 13 is specifically configured to: collect state data of the virtualized instance, and save the state data to a specified file; and retain, in place, target memory data in a target memory space of the virtualized instance.

Detailed implementations and beneficial effects of the steps in the method of some embodiments have been described in detail in the foregoing embodiments, and are not described in detail herein again.

As shown in FIG. 4, some embodiments of the present disclosure provide a method for saving a state of a virtualized instance. As shown in FIG. 4, the method includes the following steps.

In step S401, state data of a virtualized instance is collected in response to a state saving operation on the virtualized instance, and the state data is saved to a specified file.

In step S402, before the virtualized instance is restored, first memory data generated by the virtualized instance is retained in place in a file sharing manner, where the virtualized instance corresponds to a target memory space, the target memory space includes a system memory space, and the first memory data is stored in the system memory space in a file form.

In some optional embodiments, an operation of retaining the first memory data in place in a file sharing manner includes the following steps. The first memory data is stored to a memory mapping file in the system memory space when storing the first memory data, and the memory mapping file is kept from being released during upgrade of the target virtualization component, to implement retaining of the first memory data in place; and the memory mapping file is a memory mapping of a target persistent file used by the virtualized

instance, and the target persistent file is used for persistently storing data in the memory mapping file.

In some optional embodiments, the method further includes the following steps. The target persistent file is created for the virtualized instance. And a memory mapping file corresponding to the target persistent file is created in the system memory space. The first memory data is written into the memory mapping file in a process in which the virtualized instance generates the first memory data. And data in the memory mapping file is stored to the target persistent file according to a set persistence policy.

In some optional embodiments, an operation of retaining the first memory data in place in the file sharing manner includes the following steps. A memory file system is invoked to store the first memory data to the system memory space in a memory file form when storing the first memory data. And the memory file is kept from being released during upgrade of the target virtualization component, to implement retaining of the first memory data in place.

In some optional embodiments, the target memory space further includes a peripheral memory space, the peripheral memory space is a memory space on a target physical peripheral device used by the virtualized instance, and the peripheral memory space stores second memory data generated in a process in which the virtualized instance uses the target physical peripheral device; and the method further includes the following steps. A rewrite operation on the peripheral memory space is prohibited performing when a driver of the target physical peripheral device is uninstalled. And an initialization operation on the peripheral memory space is prohibited performing when the driver is reloaded, to implement retaining of the second memory data in place.

In some optional embodiments, the method further includes the following steps. The virtualized instance is recreated based on a resource file corresponding to the virtualized instance, the virtualized instance reuses a unique identifier before restoration. The target memory space is reallocated to the virtualized instance based on the unique identifier. And based on state data in the specified file, the virtualized instance is restored to a running state during state saving.

Detailed implementations and beneficial effects of the steps in the method of some embodiments have been described in detail in the foregoing embodiments, and are not described in detail herein again.

It is noted that, the steps of the method provided in the foregoing embodiments are performed by a same device, or the method is performed by different devices. For example, step S401 to step S403 are performed by device A. For another example, steps S401 and S402 are performed by device A, and step 403 is executed by device B; and the like.

In addition, in some procedures described in the foregoing embodiments and accompanying drawings, a plurality of operations occurring in a specific sequence are included. However, it is clearly understood that the operations are not performed or are not performed in parallel in the sequence in which the operations occur in present disclosure. Sequence numbers of the operations, such as 401 and 402, are merely used for distinguishing different operations, and the sequence numbers do not indicate any execution sequence. In addition, the procedures include more or fewer operations, and the operations are performed in sequence or in parallel. It is noted that, descriptions such as “first” and “second” in this specification are used for distinguishing different messages, devices, modules, and the like, do not indicate a sequence, and do not limit that “first” and “second” are different types.

FIG. 5 is a structural schematic diagram of an apparatus for upgrading a virtualization component according to some exemplary embodiments of the present disclosure. As shown in FIG. 5, the apparatus 500 for upgrading a virtualization component includes: a determining module 51, a saving module 52, an upgrade module 53, and a restoration module 54.

The determining module 51 is configure to determine a target virtualization component, the target virtualization component being a virtualization component to be upgraded in a virtual machine monitor running on a physical machine.

The saving module 52 is configured to save a state of a virtualized instance running on the physical machine, the virtualized instance being created by the virtual machine monitor.

The upgrade module 53 is configured to: perform an uninstallation operation on the target virtualization component that is not upgraded, and reinstall and load the target virtualization component that is upgraded.

The restoration module 54 is configured to restore the state of the virtualized instance based on state information of the virtualized instance.

In some optional embodiments, when saving the state of the virtualized instance running on the physical machine, the saving module 52 is specifically configured to: collect state data of the virtualized instance, and save the state data to a specified file; and retain, in place, target memory data in a target memory space of the virtualized instance.

In some optional embodiments, the target memory space includes a system memory space, and the target memory data includes first memory data stored in the system memory space in a file form; and when retaining, in place, the target memory data in the target memory space of the virtualized instance, the saving module 52 is specifically configured to retain the first memory data in place in a file sharing manner.

Further, optionally, when retaining the first memory data in place in the file sharing manner, the saving module 52 is specifically configured to: store the first memory data to a memory mapping file in the system memory space when storing the first memory data, and keep the memory mapping file from being released during upgrade of the target virtualization component, to implement retaining of the first memory data in place; the memory mapping file being a memory mapping of a target persistent file used by the virtualized instance, and the target persistent file being used for persistently storing data in the memory mapping file; or invoke a memory file system to store the first memory data to the system memory space in a memory file form when storing the first memory data, and keep the memory file from being released during upgrade of the target virtualization component, to implement retaining of the first memory data in place.

In some optional embodiments, the saving module 52 is further configured to: create the target persistent file for the virtualized instance, and create a memory mapping file corresponding to the target persistent file in the system memory space; write the first memory data into the memory mapping file in a process in which the virtualized instance generates

the first memory data; and store the data in the memory mapping file to the target persistent file according to a set persistence policy.

In some optional embodiments, the target memory space includes a peripheral memory space, and the peripheral memory space is a memory space on a target physical peripheral device used by the virtualized instance; the target memory data includes second memory data stored in the peripheral memory space; and when retaining, in place, the target memory data in the target memory space of the virtualized instance, the saving module 52 is specifically configured to: when the target virtualization component includes a driver of the target physical peripheral device, prohibit performing a rewrite operation on the peripheral memory space when the driver is uninstalled; and prohibit performing an initialization operation on the peripheral memory space when the driver that is upgraded is reloaded.

In some optional embodiments, before the driver is uninstalled, the saving module 52 is further configured to enable a memory retention function of the driver by using an interface externally exposed by the driver; and the memory retention function includes skipping performing the rewrite operation on the peripheral memory space in an uninstallation process and skipping performing the initialization operation on the peripheral memory space in a loading process.

In some optional embodiments, when restoring the state of the virtualized instance based on state information of the virtualized instance, the restoration module 54 is specifically configured to: recreate the virtualized instance based on a resource file corresponding to the virtualized instance, the virtualized instance reusing a unique identifier before restoration; reallocate the target memory space to the virtualized instance based on the unique identifier; and restore, based on state data in the specified file, the virtualized instance to a running state during state saving.

In some optional embodiments, the virtual machine monitor includes a first virtualization component that creates the virtualized instance; and when saving the state of the virtualized instance running on the physical machine, the saving module 52 is specifically configured to: invoke the first virtualization component, to save the state of the virtualized instance. Correspondingly, when restoring the state of the virtualized instance based on state information of the virtualized instance, the restoration module 54 is specifically configured

to: invoke the first virtualization component when the target virtualization component does not include the first virtualization component, to restore the state of the virtualized instance based on the state information of the virtualized instance; and invoke the first virtualization component that is upgraded when the target virtualization component includes the first virtualization component, to restore the state of the virtualized instance based on the state information of the virtualized instance.

In some optional embodiments, before performing an uninstallation operation on the target virtualization component that is not upgraded, the saving module 52 is specifically configured to optimize an uninstallation process, an installation process, a loading process, or an initialization process of the driver when the target virtualization component includes the driver of the target physical peripheral device used by the virtualized instance.

In some optional embodiments, when saving the state of the virtualized instance running on the physical machine, the saving module 52 is specifically configured to: when there are a plurality of virtualized instances, save states of the plurality of virtualized instances in a concurrent operation manner.

FIG. 6 is a structural schematic diagram of an apparatus for saving a state of a virtualized instance according to some exemplary embodiments of the present disclosure. As shown in FIG. 6, the apparatus 600 for saving a state of a virtualized instance includes: a collecting module 61, a saving module 62, and a retaining module 63.

The collecting module 61 is configured to: collect state data of a virtualized instance in response to a state saving operation on the virtualized instance, and save the state data to a specified file.

The saving module 62 is configured to store first memory data generated by the virtualized instance to the system memory space in a file form, the virtualized instance corresponding to a target memory space, and the target memory space including the system memory space.

The retaining module 63 is configured to retain the first memory data in place in a file sharing manner before the virtualized instance is restored.

In some optional embodiments, the retaining module 63 is specifically configured to: store the first memory data to a memory mapping file in the system memory space when storing

the first memory data, and keep the memory mapping file from being released during upgrade of the target virtualization component, to implement retaining of the first memory data in place; the memory mapping file being a memory mapping of a target persistent file used by the virtualized instance, and the target persistent file being used for persistently storing data in the memory mapping file; or invoke a memory file system to store the first memory data to the system memory space in a memory file form when storing the first memory data, and keep the memory file from being released during upgrade of the target virtualization component, to implement retaining of the first memory data in place.

In some optional embodiments, the saving module 62 is further configured to: create the target persistent file for the virtualized instance, and create a memory mapping file corresponding to the target persistent file in the system memory space; write the first memory data into the memory mapping file in a process in which the virtualized instance generates the first memory data; and store the data in the memory mapping file in the target persistent file according to a set persistence policy.

In some optional embodiments, the target memory space further includes a peripheral memory space, the peripheral memory space is a memory space on a target physical peripheral device used by the virtualized instance, and the peripheral memory space stores second memory data generated in a process in which the virtualized instance uses the target physical peripheral device; and the retaining module 63 is further configured to prohibit performing a rewrite operation on the peripheral memory space when a driver of the target physical peripheral device is uninstalled; and prohibit performing an initialization operation on the peripheral memory space when the driver is reloaded, to implement retaining of the second memory data in place.

In some optional embodiments, the retaining module 63 is further configured to: recreate the virtualized instance based on a resource file corresponding to the virtualized instance, the virtualized instance reusing a unique identifier before restoration; reallocate the target memory space to the virtualized instance based on the unique identifier; and restore, based on state data in the specified file, the virtualized instance to a running state during state saving.

FIG. 7 is a structural schematic diagram of another physical machine according to some exemplary embodiments of the present disclosure. The physical machine includes hardware resources, and the hardware resources includes a memory 74 and a processor 75.

The memory 74 is configured to store a computer program, and is configured to store other various types of data, to support operations on the physical machine. Examples of the data include instructions for any application or method operating on the physical machine, contact data, phone book data, messages, pictures, videos, and the like.

The memory 74 is implemented by using any type of volatile or non-volatile storage devices or a combination thereof, such as a static random access memory (SRAM), an electrically erasable programmable read-only memory (EEPROM), an erasable programmable read-only memory (EPROM), a programmable read-only memory (PROM), a read-only memory (ROM), a magnetic memory, a flash memory, a magnetic disk, or an optical disc.

The processor 75 is coupled with the memory 74, and is configured to execute the computer program in the memory 74, to: determine a target virtualization component, the target virtualization component being a virtualization component to be upgraded in a virtual machine monitor running on a physical machine; save a state of a virtualized instance running on the physical machine, the virtualized instance being created by the virtual machine monitor; perform an uninstallation operation on the target virtualization component that is not upgraded, and reinstall and load the target virtualization component that is upgraded; and restore the state of the virtualized instance based on state information of the virtualized instance.

In some optional embodiments, when saving the state of the virtualized instance running on the physical machine, the processor 75 is specifically configured to: collect state data of the virtualized instance, and save the state data to a specified file; and retain, in place, target memory data in a target memory space of the virtualized instance.

In some optional embodiments, the target memory space includes a system memory space, and the target memory data includes first memory data stored in the system memory space in a file form; and when retaining, in place, the target memory data in the target memory space of the virtualized instance, the processor 75 is specifically configured to retain the first memory data in place in a file sharing manner.

Further, optionally, when retaining the first memory data in place in the file sharing manner, the processor 75 is specifically configured to: store the first memory data to a memory mapping file in the system memory space when storing the first memory data, and keep the memory mapping file from being released during upgrade of the target virtualization component, to implement retaining of the first memory data in place; the memory mapping file being a memory mapping of a target persistent file used by the virtualized instance, and the target persistent file being used for persistently storing data in the memory mapping file; or invoke a memory file system to store the first memory data to the system memory space in a memory file form when storing the first memory data, and keep the memory file from being released during upgrade of the target virtualization component, to implement retaining of the first memory data in place.

In some optional embodiments, the processor 75 is further configured to: create the target persistent file for the virtualized instance, and create a memory mapping file corresponding to the target persistent file in the system memory space; write the first memory data into the memory mapping file in a process in which the virtualized instance generates the first memory data; and store the data in the memory mapping file to the target persistent file according to a set persistence policy.

In some optional embodiments, the target memory space includes a peripheral memory space, and the peripheral memory space is a memory space on a target physical peripheral device used by the virtualized instance; the target memory data includes second memory data stored in the peripheral memory space; and when retaining, in place, the target memory data in the target memory space of the virtualized instance, the processor 75 is specifically configured to: when the target virtualization component includes a driver of the target physical peripheral device, prohibit performing a rewrite operation on the peripheral memory space when the driver is uninstalled; and prohibit performing an initialization operation on the peripheral memory space when the driver that is upgraded is reloaded.

In some optional embodiments, before the driver is uninstalled, the processor 75 is further configured to enable a memory retention function of the driver by using an interface externally exposed by the driver; and the memory retention function includes skipping performing an rewrite operation on the peripheral memory space in an uninstallation process

and skipping performing an initialization operation on the peripheral memory space in a loading process.

In some optional embodiments, when restoring the state of the virtualized instance based on the state information of the virtualized instance, the processor 75 is specifically configured to: recreate the virtualized instance based on a resource file corresponding to the virtualized instance, the virtualized instance reusing a unique identifier before restoration; reallocate the target memory space to the virtualized instance based on the unique identifier; and restore, based on state data in the specified file, the virtualized instance to a running state during state saving.

In some optional embodiments, the virtual machine monitor includes a first virtualization component that creates the virtualized instance; and when saving the state of the virtualized instance running on the physical machine, the processor 75 is specifically configured to: invoke the first virtualization component, to save the state of the virtualized instance. Correspondingly, when restoring the state of the virtualized instance based on the state information of the virtualized instance, the processor 75 is specifically configured to: invoke the first virtualization component when the target virtualization component does not include the first virtualization component, to restore the state of the virtualized instance based on the state information of the virtualized instance; and invoke the first virtualization component that is upgraded when the target virtualization component includes the first virtualization component, to restore the state of the virtualized instance based on the state information of the virtualized instance.

In some optional embodiments, before performing the uninstallation operation on the target virtualization component that is not upgraded, the processor 75 is specifically configured to optimize an uninstallation process, an installation process, a loading process, or an initialization process of the driver when the target virtualization component includes the driver of the target physical peripheral device used by the virtualized instance.

In some optional embodiments, when saving the state of the virtualized instance running on the physical machine, the processor 75 is specifically configured to: when there are a plurality of virtualized instances, save states of the plurality of virtualized instances in a concurrent operation manner.

Further, as shown in FIG. 7, the physical machine further includes: a communication component 76, a display 77, a power supply component 78, an audio component 79, and other components. Some components are schematically provided in FIG. 7, which does not mean that the physical machine includes only the components shown in FIG. 7. In addition, components in dashed boxes in FIG. 7 are optional components rather than mandatory components, and are specifically determined based on a product form of a worker node. The worker node in some embodiments is implemented as a terminal device such as a desktop computer, a notebook computer, a smartphone, or an IOT device, or is a serving-end device such as a conventional server, a cloud server, or a server array. When the worker node in some embodiments is implemented as the terminal device such as the desktop computer, the notebook computer, or the smartphone, the physical machine includes the components in the dashed boxes in FIG. 7. When the worker node in some embodiments is implemented as the serving-end device such as the conventional server, the cloud server, or the server array, the physical machine does not include the components in the dashed boxes in FIG. 7.

Some embodiments of the present disclosure further provide a physical machine. An implementation structure of the physical machine is the same as or similar to the implementation structure of the physical machine shown in FIG. 7, and the physical machine is implemented with reference to the structure of the physical machine shown in FIG. 7. A difference between the physical machine provided in some embodiments and the physical machine in the embodiment shown in FIG. 7 mainly lies in that the processor executes an computer program stored in the memory to implement different functions. For the physical machine provided in some embodiments, the processor of the physical machine executes the computer program stored in the memory, to: collect state data of a virtualized instance in response to a state saving operation on the virtualized instance, and save the state data to a specified file, where the virtualized instance corresponds to a target memory space, the target memory space includes a system memory space, and first memory data generated by the virtualized instance is stored in the system memory space in a file form; and retain the first memory data in place in a file sharing manner before the virtualized instance is restored.

In some optional embodiments, when retaining the first memory data in place in the file sharing manner, the processor is specifically configured to: store the first memory data to a memory mapping file in the system memory space when storing the first memory data, and

keep the memory mapping file from being released during upgrade of the target virtualization component, to implement retaining of the first memory data in place; the memory mapping file being a memory mapping of a target persistent file used by the virtualized instance, and the target persistent file being used for persistently storing data in the memory mapping file; or invoke a memory file system to store the first memory data to the system memory space in a memory file form when storing the first memory data, and keep the memory file from being released during upgrade of the target virtualization component, to implement retaining of the first memory data in place.

In some optional embodiments, the processor is further configured to: create the target persistent file for the virtualized instance, and create a memory mapping file corresponding to the target persistent file in the system memory space; write the first memory data into the memory mapping file in a process in which the virtualized instance generates the first memory data; and store the data in the memory mapping file in the target persistent file according to a set persistence policy.

In some optional embodiments, the target memory space further includes a peripheral memory space, the peripheral memory space is a memory space on a target physical peripheral device used by the virtualized instance, and the peripheral memory space stores second memory data generated in a process in which the virtualized instance uses the target physical peripheral device; and the processor is further configured to prohibit performing a rewrite operation on the peripheral memory space when a driver of the target physical peripheral device is uninstalled; and prohibit performing an initialization operation on the peripheral memory space when the driver is reloaded, to implement retaining of the second memory data in place.

In some optional embodiments, the processor is further configured to: recreate the virtualized instance based on a resource file corresponding to the virtualized instance, the virtualized instance reusing a unique identifier before restoration; reallocate the target memory space to the virtualized instance based on the unique identifier; and restore, based on state data in the specified file, the virtualized instance to a running state during state saving.

Detailed implementations and beneficial effects of the steps in the method of some embodiments have been described in detail in the foregoing embodiments, and are not described in detail herein again.

Correspondingly, some embodiments of the present disclosure further provide a computer-readable storage medium storing a computer program. The computer program, when executed by a processor, causes the processor to implement the steps in the foregoing method embodiments.

Correspondingly, some embodiments of the present disclosure further provide a computer program product. The computer program product includes a computer program or computer instructions. The computer program or computer instructions, when executed by a processor, enables the processor to implement the steps in the foregoing method embodiments.

The foregoing memory is implemented by any type of volatile or non-volatile storage device or a combination thereof, such as a static random-access memory (SRAM), an electrically erasable programmable read only memory (EEPROM), an erasable programmable read only memory (EPROM), a programmable read-only memory (PROM), a read-only memory (ROM), a magnetic memory, a flash memory, a magnetic disk, or an optical disc.

The communication component is configured to facilitate communication in a wired or wireless manner between a device in which the communication component is located and other devices. The device at which the communication component is located accesses a wireless network based on a communication standard, for example, a mobile communication network such as Wi-Fi, 2G, 3G, 4G/LTE, 5G, or a combination thereof. In some exemplary embodiments, the communication component receives a broadcast signal or broadcast related information from an external broadcast management system through a broadcast channel. In some exemplary embodiments, the communication component further includes a near field communication (NFC) module, to facilitate short-range communication. For example, the NFC module is implemented based on a radio frequency identification (RFID) technology, an infrared data association (IrDA) technology, an ultra wide band (UWB) technology, a Bluetooth (BT) technology, and other technologies.

The display includes a screen, and the screen includes a liquid crystal display (LCD) and a touch panel (TP). When the screen includes the touch panel, the screen is implemented as a

touchscreen, to receive an input signal from a user. The touch panel includes at least one touch sensor to sense a touch, a slide, and a gesture on the touch panel. The at least one touch sensor not only senses a boundary of a touch or slide operation, but also detects duration and pressure related to the touch or slide operation.

The power supply component provides power to various components of a device in which the power supply component is located. The power supply component includes a power supply management system, at least one power supply, and other components associated with generating, managing, and allocating power for the device in which the power supply component is located.

The audio component is configured to output or input an audio signal. For example, the audio component includes a microphone (MIC). When a device in which the audio component is located is in an operation mode, such as a call mode, a recording mode, and a voice identification mode, the microphone is configured to receive an external audio signal. The external audio signal is further stored in a memory or sent by using the communication component. In some embodiments, the audio component further includes a speaker, configured to output an audio signal.

A person having ordinary skill in the art understands that the embodiments of the present disclosure are provided as methods, systems, or computer program products. Therefore, the present disclosure uses a form of hardware-only embodiments, software-only embodiments, or embodiments with a combination of software and hardware. Moreover, the present disclosure uses a form of a computer program product that is implemented on at least one computer-readable storage media (including but not limited to a disk memory, a compact disc read-only memory (CD-ROM), an optical memory, and the like) that includes computer-usable program code.

The present disclosure is described with reference to a flowchart and/or block diagram of a method, a device (system), and a computer program product according to the embodiments of the present disclosure. It is understood that computer program instructions are used for implementing each procedure or each block in the flowcharts or the block diagrams and a combination of procedures and blocks in the flowcharts and the block diagrams. The computer program instructions are provided for a general-purpose computer, a dedicated

computer, an embedded processor, or a processor of any other programmable data processing device to generate a machine, so that the instructions executed by a computer or a processor of any other programmable data processing device generate an apparatus for implementing a specific function in one or more processes in the flowcharts or in one or more blocks in the block diagrams.

The computer program instructions are alternatively stored in a computer-readable memory that instructs the computer or another programmable data processing device to work in a specific manner, so that the instructions stored in the computer-readable memory generate an artifact that includes an instruction apparatus. The instruction apparatus implements a specified function in one or more procedures in the flowcharts or in one or more blocks in the block diagrams.

The computer program instructions are further loaded onto a computer or another programmable data processing device, so that a series of operations and steps are performed on the computer or the another programmable device, thereby generating computer-implemented processing. Therefore, the instructions executed on the computer or the other programmable device provide steps for implementing a specific function in one or more processes in the flowcharts or in one or more blocks in the block diagrams.

In a typical configuration, the computing device includes at least one processor (such as Central Processing Unit), an input/output interface, a network interface, and a memory.

The memory includes a form such as a non-permanent memory, a random access memory (RAM), or a non-volatile memory in the computer-readable medium, such as a read-only memory (ROM) or a flash memory (flash RAM). The memory is an example of the computer-readable medium.

The computer-readable medium includes a non-volatile medium and a volatile medium, a removable medium and a non-removable medium, which implements storage of information by using any method or technology. The information is computer-readable instructions, a data structure, a program module, or other data. Examples of storage mediums of the computer include, but are not limited to, a phase-change random access memory (PRAM), a static random access memory (SRAM), a dynamic random access memory (DRAM), other types of random access memory (RAM), a read-only memory (ROM), an electrically

erasable programmable read-only memory (EEPROM), a flash memory or another memory technology, a compact disc read-only memory (CD-ROM), a digital video disc (DVD) or another optical memory, a magnetic cassette, a magnetic disk storage, another magnetic storage device, and any other non-transmission medium that are configured to store information accessible to the computing device. As defined in present disclosure, the computer-readable medium does not include computer readable transitory media, such as modulated data signals and carriers.

It is further noted that, the term “include”, “comprise”, or any other variant is intended to cover a non-exclusive inclusion, so that a procedure, a method, an article, or a device that includes a list of elements not only includes those elements but also includes other elements that are not explicitly listed, or further includes elements inherent to such a procedure, method, article, or device. In absence of more constraints, an element preceded by a statement “includes a...” does not preclude the presence of additional identical elements in the procedure, method, article, or device that includes the element.

The foregoing descriptions are merely embodiments of the present disclosure, but are not intended to limit the present disclosure. For a person having ordinary skill in the art, the present disclosure has various changes and variations. Any modification, equivalent replacement, improvement, or the like made without departing from the spirit and principle of the present disclosure shall fall within the scope of claims of the present disclosure.

Claims

1. A method for upgrading a virtualization component, comprising:

   determining a target virtualization component, the target virtualization component being a virtualization component to be upgraded in a virtual machine monitor running on a physical machine;

   saving a state of a virtualized instance running on the physical machine, the virtualized instance being created by the virtual machine monitor;

   performing an uninstallation operation on the target virtualization component that is not upgraded, and reinstalling and loading the target virtualization component that is upgraded; and

   restoring the state of the virtualized instance based on state information of the virtualized instance.

2. The method according to claim 1, wherein saving the state of the virtualized instance running on the physical machine comprises:

   collecting state data of the virtualized instance, and saving the state data to a specified file; and

   retaining, in place, target memory data in a target memory space of the virtualized instance.

3. The method according to claim 2, wherein the target memory space comprises a system memory space, and the target memory data comprises first memory data stored in the system memory space in a file form; and

   retaining, in place, the target memory data in the target memory space of the virtualized instance comprises: retaining the first memory data in place in a file sharing manner.

4. The method according to claim 3, wherein retaining the first memory data in place in the file sharing manner comprises:

   storing the first memory data to a memory mapping file in the system memory space when storing the first memory data, and keeping the memory mapping file from being released during upgrade of the target virtualization component, to implement retaining of the first memory data in place; the memory mapping file being a memory mapping of a target persistent file used by the virtualized instance, and the target persistent file being used for persistently storing data in the memory mapping file;

or

invoking a memory file system to store the first memory data to the system memory space in a memory file form when storing the first memory data, and keeping the memory file from being released during upgrade of the target virtualization component, to implement retaining of the first memory data in place.

5. The method according to claim 2, wherein the target memory space comprises a peripheral memory space, and the peripheral memory space is a memory space on a target physical peripheral device used by the virtualized instance; the target memory data comprises second memory data stored in the peripheral memory space; and

retaining, in place, the target memory data in the target memory space of the virtualized instance comprises:

when the target virtualization component comprises a driver of the target physical peripheral device, prohibiting performing a rewrite operation on the peripheral memory space when the driver is uninstalled; and

prohibiting performing an initialization operation on the peripheral memory space when the driver that is upgraded is reloaded.

6. The method according to claim 5, wherein before uninstalling the driver, the method further comprises:

enabling a memory retention function of the driver by using an interface externally exposed by the driver; and the memory retention function comprises skipping performing the rewrite operation on the peripheral memory space in an uninstallation process and skipping performing the initialization operation on the peripheral memory space in a loading process;

wherein the driver that is upgraded enables the memory retention function by default.

7. The method according to claim 2, wherein restoring the state of the virtualized instance based on the state information of the virtualized instance comprises:

recreating the virtualized instance based on a resource file corresponding to the virtualized instance, the virtualized instance reusing a unique identifier before restoration;

reallocating the target memory space to the virtualized instance based on the unique identifier; and

restoring, based on the state data in the specified file, the virtualized instance to a running state during state saving.

8. The method according to any one of claims 1 to 7, wherein before performing the uninstall operation on the target virtualization component, the method further comprises:

optimizing an uninstallation process, an installation process, a loading process, or an initialization process of a driver when the target virtualization component comprises the driver of a target physical peripheral device used by the virtualized instance.

9. A physical machine, comprising: hardware resources, wherein a virtual machine monitor, a virtualized instance, and an upgrade program run on the hardware resources; and the virtual machine monitor comprises a plurality of virtualization components, configured to virtualize the hardware resources, and create and manage the virtualized instance;

wherein the upgrade program is configured to: determine a target virtualization component to be upgraded in the plurality of virtualization components; save a state of the virtualized instance; perform an uninstallation operation on the target virtualization component that is not upgraded, and reinstall and load the target virtualization component that is upgraded; and restore the state of the virtualized instance based on state information of the virtualized instance.

10. The physical machine according to claim 9, wherein the plurality of virtualization components comprise a first virtualization component in a user mode and a second virtualization component in a kernel mode; the target virtualization component comprises at least one of the first virtualization component or the second virtualization component; and

the upgrade program is configured to: invoke the first virtualization component to save the state of the virtualized instance; and

invoke the first virtualization component when the target virtualization component does not comprise the first virtualization component, to restore the state of the virtualized instance based on the state information of the virtualized instance;

invoke the first virtualization component that is upgraded when the target virtualization component comprises the first virtualization component, to restore the state of the virtualized instance based on the state information of the virtualized instance.

11. A method for saving a state of a virtualized instance, comprising:

collecting state data of a virtualized instance in response to a state saving operation on the virtualized instance, and saving the state data to a specified file;

wherein the virtualized instance corresponds to a target memory space, the target memory space comprises a system memory space, and first memory data generated by the virtualized instance is stored in the system memory space in a file form; and

retaining the first memory data in place in a file sharing manner before the virtualized instance is restored.

12. The method according to claim 11, wherein retaining the first memory data in place in the file sharing manner comprises:

storing the first memory data to a memory mapping file in the system memory space when storing the first memory data, and keeping the memory mapping file from being released during upgrade of the target virtualization component, to implement retaining of the first memory data in place; the memory mapping file being a memory mapping of a target persistent file used by the virtualized instance, and the target persistent file being used for persistently storing data in the memory mapping file;

or

invoking a memory file system to store the first memory data to the system memory space in a memory file form when storing the first memory data, and keeping the memory file from being released during upgrade of the target virtualization component, to implement retaining of the first memory data in place.

13. The method according to claims 11 or 12, wherein the target memory space further comprises a peripheral memory space, the peripheral memory space is a memory space on a target physical peripheral device used by the virtualized instance, and the peripheral memory space stores second memory data generated in a process in which the virtualized instance uses the target physical peripheral device; and the method further comprises:

prohibiting performing a rewrite operation on the peripheral memory space when a driver of the target physical peripheral device is uninstalled; and prohibiting performing an initialization operation on the peripheral memory space when the driver is reloaded, to implement retaining of the second memory data in place.

14. A physical machine, comprising hardware resources, wherein the hardware resources comprise a memory and a processor, the memory stores a computer program, and the

processor is coupled with the memory and configured to execute the computer program, to implement the steps of the method as claimed in any one of claims 1 to 8 and claims 11 to 13.

15. A computer-readable storage medium storing a computer program, the computer program, when executed by a processor, causing the processor to implement the steps of the method as claimed in any one of claims 1 to 8 and claims 11 to 13.

16. A computer program product, comprising a computer program or computer instructions, the computer program or the computer instructions, when executed by a processor, causing the processor to implement the steps of the method as claimed in any one of claims 1 to 8 and claims 11 to 13.