

SCHEDULING METHOD FOR TASKS, COMPUTING DEVICE, AND STORAGE MEDIUM

The present application claims priority to Chinese patent application No. 202010475800.4 filed on May 29, 2020 and entitled "SCHEDULING METHOD FOR
5 TASKS, COMPUTING DEVICE, AND STORAGE MEDIUM", which is incorporated herein by reference in its entirety.

Technical Field

The present application relates to the technical field of computers and, in particular, to a scheduling method for tasks, a computing device, and a storage medium.

10 Background

With the rapid development of the Internet, cloud services have a number of advantages over conventional computing services, such as lower price, ease of extensibility, scalability, ease of management, and high availability, and thus occupy an increasingly critical position in the information Internet era. In addition, container technology is widely used in the field
15 of cloud services because of its characteristics.

However, due to the rapid development of artificial intelligence, heterogeneous computing has become an emerging hotspot for cloud services. As a result, cloud servers can contain multiple kinds of hardware resources, such as multiple kinds of physical processing devices, while how to schedule these hardware resources in an efficient and
20 convenient way is the key to scheduling resources.

Summary of the Invention

Various aspects of the present application provide a scheduling method for tasks, a computing device, and a storage medium that achieve efficient and convenient scheduling of hardware resources.

The embodiments of the present application provide a scheduling method for tasks, comprising: acquiring task information of a current task to be executed, the task information describing the current task to be executed; determining an actual execution time for the current task to be executed according to the task information; and comparing the actual
5 execution time with a preset scheduling time corresponding to the current task to be executed, and adjusting an actual scheduling time corresponding to a next task to be executed according to the comparison result so as to determine whether to schedule the next task to be executed.

The embodiments of the present application further provide a scheduling method for
10 container tasks, comprising: receiving a current task to be executed sent by any container, and acquiring task information of the current task to be executed, the task information describing the current task to be executed; determining an actual execution time for the current task to be executed according to the task information; and comparing the actual execution time with a preset scheduling time corresponding to the current task to be executed,
15 and adjusting an actual scheduling time corresponding to a next task to be executed from the same container according to the comparison result so as to determine whether to schedule the next task to be executed from the same container.

The embodiments of the present application further provide a scheduling apparatus for tasks, comprising: an acquisition module for acquiring task information of a current task to
20 be executed, the task information describing the current task to be executed; a determination module for determining an actual execution time for the current task to be executed according to the task information; and an adjustment module for comparing the actual execution time with a preset scheduling time corresponding to the current task to be executed, and adjusting an actual scheduling time corresponding to a next task to be executed

according to the comparison result so as to determine whether to schedule the next task to be executed.

The embodiments of the present application further provide a scheduling apparatus for container tasks, comprising: a receiving module for receiving a current task to be executed
5 sent by any container, and acquiring task information of the current task to be executed, the task information describing the current task to be executed; a determination module for determining an actual execution time for the current task to be executed according to the task information; and an adjustment module for comparing the actual execution time with a preset scheduling time corresponding to the current task to be executed, and adjusting an actual
10 scheduling time corresponding to a next task to be executed from the same container according to the comparison result so as to determine whether to schedule the next task to be executed from the same container.

The embodiments of the present application further provide a computing device comprising a memory and a processor, wherein the memory is used for storing a computer
15 program; and the processor is used for executing the computer program for: acquiring task information of a current task to be executed, the task information describing the current task to be executed; determining an actual execution time for the current task to be executed according to the task information; and comparing the actual execution time with a preset scheduling time corresponding to the current task to be executed, and adjusting an actual
20 scheduling time corresponding to a next task to be executed according to the comparison result so as to determine whether to schedule the next task to be executed.

The embodiments of the present application further provide a computing device comprising a memory and a processor, wherein the memory is used for storing a computer program; and the processor is used for executing the computer program for: receiving a
25 current task to be executed sent by any container, and acquiring task information of the

current task to be executed, the task information describing the current task to be executed;
determining an actual execution time for the current task to be executed according to the task
information; and comparing the actual execution time with a preset scheduling time
corresponding to the current task to be executed, and adjusting an actual scheduling time
5 corresponding to a next task to be executed from the same container according to the
comparison result so as to determine whether to schedule the next task to be executed from
the same container.

The embodiments of the present application further provide a computer-readable storage
medium storing a computer program, wherein the computer program, when executed by one
10 or more processors, causes the one or more processors to implement steps in the method
described above.

In the embodiments of the present application, task information of a current task to be
executed is acquired, the task information describing the current task to be executed; an
actual execution time for the current task to be executed is determined according to the task
15 information; and the actual execution time is compared with a preset scheduling time
corresponding to the current task to be executed, and an actual scheduling time
corresponding to a next task to be executed is adjusted according to the comparison result
so as to determine whether to schedule the next task to be executed. By comparing the actual
execution time with the preset scheduling time after determining the actual execution time,
20 the actual scheduling time corresponding to the next task to be executed can be adjusted so
as to decide whether to perform task scheduling, thus achieving quick and easy task
scheduling and full utilization of hardware resources. Moreover, since the actual scheduling
time is adjusted, the actual scheduling time for each task can be accurately and efficiently
allocated, and the scheduling time can be balanced among the tasks, thus reducing the

situation where the current task to be executed excessively occupies the scheduling time for other tasks.

In addition, as there is no use of a device driver to acquire the drive method to perform task scheduling, this is very friendly for the non-open source device drivers because there is
5 no need to care about whether the device driver is open source or not (i.e., it can be open source or non-open source), and the development of device drivers can be reduced.

Brief Description of the Drawings

The accompanying drawings illustrated herein are used to provide further understanding of the present application and form part of the present application. The schematic
10 embodiments of the present application and the description thereof are used to explain the present application and do not constitute an improper limitation to the present application.

In the accompanying drawings:

FIG. 1 is a schematic structural diagram of a scheduling system for tasks of an exemplary embodiment of the present application;

15 FIG. 2 is a flow chart of a scheduling method for tasks of an exemplary embodiment of the present application;

FIG. 3 is a flow chart of a scheduling method for tasks of another exemplary embodiment of the present application;

20 FIG. 4 is a flow chart of a scheduling method for container tasks of another exemplary embodiment of the present application;

FIG. 5 is a schematic structural diagram of a scheduling apparatus for tasks provided in another exemplary embodiment of the present application;

FIG. 6 is a schematic structural diagram of a scheduling apparatus for container tasks provided in another exemplary embodiment of the present application;

FIG. 7 is a schematic structural diagram of a computing device provided in an exemplary embodiment of the present application; and

FIG. 8 is a schematic structural diagram of a computing device provided in another exemplary embodiment of the present application.

5 Detailed Description

In order to make the objectives, technical solutions and advantages of the present application clearer, the technical solutions of the present application will be clearly and completely described below in conjunction with the specific embodiments of the present application and the corresponding accompanying drawings. Obviously, the described
10 embodiments are only part of the embodiments of the present application, but not all of them. Based on the embodiments in the present application, all the other embodiments obtained by those of ordinary skill in the art without making any inventive labor fall within the scope of protection of the present application.

Container technology is a technology for packaging software into standardized units for
15 use in development, delivery, and deployment. It ensures the consistency of the application operating environment, achieves faster startup, and has properties such as isolation, extensibility, ease of migration, and sustainable delivery and deployment. Based on the above properties, container technology is widely applied in the field of cloud services. Due to the rapid development of artificial intelligence, heterogeneous computing has become an
20 emerging hotspot for cloud services.

In a cloud server containing multiple processing devices, such as GPUs (Graphics Processing Units), NPUs (neural-network processing units), as well as containers, the critical core is how to efficiently and easily schedule GPU hardware resources so that they can be shared seamlessly across multiple containers, while keeping the overhead of the scheduling
25 algorithm perse to be small enough.

The embodiments of the present application propose a method that can meet the above requirements and can also work well for scheduling black-box GPU device drivers that are not open source.

The technical solutions provided by the embodiments of the present application are
5 described in detail below in conjunction with the accompanying drawings.

FIG. 1 is a schematic structural diagram of a scheduling system for tasks provided in an exemplary embodiment of the present application. As shown in FIG. 1, this system 100 may include: first device 101 and second device 102.

Among them, first device 101 may be a device with certain computing power, which can
10 implement the functions of sending data to second device 102 and acquiring response data from second device 102. The basic structure of first device 101 may include: at least one processor. The number of processors may depend on the configuration and type of the apparatus with certain computing power. The apparatus with certain computing power may also include memory, which may be volatile, such as RAM, or may be non-volatile, such as
15 read-only memory (ROM), flash memory, etc., or may include both types at the same time. The memory usually stores an operating system (OS), one or more applications, and may also store program data, among other things. In addition to the processing unit and memory, the apparatus with certain computing power further includes some basic configurations, such as a network card chip, an IO bus, display components, and some peripheral devices.
20 Alternatively, these some peripheral devices may include, for example, a keyboard, an input pen, and the like. Other peripheral devices are well known in the field and will not be elaborated herein. Alternatively, first device 101 may be a smart terminal, e.g., a cell phone, a desktop computer, a laptop, a tablet computer, etc.

Second device 102 is a device that can provide computing and processing services in a
25 network virtual environment, and may be a device that performs data processing and

performs task scheduling using the network. In terms of physical implementation, second device 102 may be any device capable of providing computing services, responding to service requests, and returning data processing results. For example, it may be a cloud server, a cloud host, a virtual center, a conventional server, etc. The composition of second device 5 102 mainly includes a processor, a hard disk, a memory, a system bus, etc., and is similar to that of a common computer architecture.

In the embodiments of the present application, first device 101 sends data to second device 102. Second device 102 receives the data and sends the data to a corresponding virtual device, such as a container. In second device 102, at least one virtual device, such as a 10 container, is included. Meanwhile, second device 102 may further include a device driver (which may also be referred to as a device driver layer or a device driver program) as well as an intermediate driver layer. A task scheduler may be provided in this intermediate driver layer for use in task scheduling. The intermediate driver layer communicates with the device driver via a communication interface. When the virtual device in second device 102 receives 15 the data, it needs to process the data, that is, process the data to generate a task (i.e., a task to be executed) and send it to the task scheduler in the intermediate driver layer. Upon receiving it, the task scheduler acquires task information of the current task to be executed, the task information describing the current task to be executed; determines an actual execution time for the task to be executed according to the task information; and compares 20 the actual execution time with a preset scheduling time corresponding to the current task to be executed, and adjusts an actual scheduling time corresponding to a next task to be executed according to the comparison result so as to determine whether to schedule the next task to be executed (which may be from the same source or from a different source). Moreover, the task scheduler sends the task to be executed through the device driver to a 25 corresponding processing device, such as a GPU, to enable it to process the task, and after

processing the task, the corresponding processing device returns the result of the task to the corresponding virtual device. The virtual device then returns the result of the task to first device 101.

It should be noted that the process of returning the result of the task to first device 101 by the virtual device may be a process of execution by the device driver in second device 102, and the specific execution process is similar to that described above, and will not be described here.

In addition, the task scheduler may also be provided in the driver device, provided that the driver device is open source. There is no need to deploy an additional intermediate driver layer.

In the application scenario of the embodiments of the present application, a user sends data, such as a request for account login which may carry an account number and a password, to second device 102, such as a cloud server, via first device 101, such as a computer. The user may send the data via a browser installed on first device 101. A virtual device in second device 102, such as container 1, receives this data and acquires the account number and the password. Container 1 sends a task to the task scheduler in the intermediate driver layer of second device 102, wherein the task may be an execution instruction for verifying whether the account number and the password are correct. After receiving the task, the task scheduler acquires the above execution instruction based on a storage address carried by the task, and may input the execution instruction into a preset model for use in prediction of the actual execution time. At the same time, the task scheduler may send the task via the device driver to a corresponding processing device, such as a GPU. After receiving the task, the GPU processes it. If the result of processing is that the account number and password are correct, it can be determined that the result is correct and login can be performed. Container 1

acquires the result of processing and returns it to the computer of the user via second device 102, indicating that the login is successful.

In addition, the task scheduler, after predicting the actual execution time, compares it with the corresponding preset scheduling time and, if it is greater than the preset scheduling time, determines an excess time; and the task scheduler reduces the actual scheduling time for the next task to be executed from container 1, with the magnitude of the reduced time being the magnitude of the excess time, and then uses the updated actual scheduling time as the actual scheduling time for the subsequent task to be executed; and until the actual scheduling time for the subsequent task to be executed from container 1 is updated to be reduced to 0, the task scheduler may then not perform the scheduling when scheduling this subsequent task to be executed, but wait for the next time when task scheduling can be performed, and then schedule this task to be executed when the next preset scheduling time arrives;

and if the predicted actual execution time is less than the preset scheduling time, the task scheduler determines an idle time. In this case, the task scheduler may either not perform any processing or, alternatively, compensate this idle time to the actual scheduling time for the next task to be executed (which may be a task from the same container or a task from a different container). After the task scheduler executes the adjustment of the actual scheduling time for the current task as well as the task scheduling, it performs the scheduling of the next task, i.e., repeating the foregoing contents.

In this embodiment, network connection is implemented between first device 101 and second device 102, where the network connection may be a wireless connection. If there is a communication connection between first device 101 and second device 102, the network standard of the mobile network may be any one of 2G (GSM), 2.5G (GPRS), 3G (WCDMA, TD-SCDMA, CDMA2000, UTMS), 4G (LTE), 4G+ (LTE+), WiMax, 5G, etc.

The scheduling process for tasks will be described in detail below in conjunction with a method embodiment.

FIG. 2 is a flow chart of a scheduling method for tasks of an exemplary embodiment of the present application. This method 200 provided by the embodiment of the present application is performed by a computing device, such as, for example, a cloud server, and more specifically, a task scheduler in the cloud server. This method 200 includes the following steps:

201: acquiring task information of a current task to be executed, the task information describing the current task to be executed;

10 202: determining an actual execution time for the current task to be executed according to the task information; and

203: comparing the actual execution time with a preset scheduling time corresponding to the current task to be executed, and adjusting an actual scheduling time corresponding to a next task to be executed according to the comparison result so as to determine whether to
15 schedule the next task to be executed.

It should be noted that according to the foregoing contents, it can be known that the task scheduler may either be provided in a device driver in the cloud server or in an intermediate driver layer which may be deployed over the device driver, that is, the intermediate driver layer is communicatively connected to the device driver. In the case where the task scheduler
20 is deployed in the intermediate driver layer, it is not necessary to be concerned about whether the device driver is open source or whether it belongs to a third party.

Details of the above steps are provided below:

201: acquiring task information of a current task to be executed, the task information describing the current task to be executed.

Here, the task to be executed (hereinafter also may be referred to as a task to be executed) is a task to be processed by a hardware resource (which may also be referred to as a hardware device) in a physical device, such as a task to be processed by a GPU in the cloud server, and the task may be a task instructing at least one processing device to verify whether a password and a user name are correct. The current task to be executed is a task that the task scheduler is preparing to schedule at the moment.

The hardware resources may also include, for example, a CPU, an NPU, a memory, a disk, a network, etc. In addition to the processing device, other hardware devices may assist the processing device to perform task processing, etc., so that the utilization of other hardware devices can also be balanced.

The approach to acquiring the current task to be executed may include: receiving, via a communication interface, the current task to be executed sent by a virtual device, each virtual device corresponding to an independent communication interface. That is, each virtual device has a one-to-one correspondence with a communication interface. In addition, each of the virtual devices and communication interfaces has its own identifier, such as ID. Therefore, based on this correspondence, upon receiving the current task to be executed, the task scheduler in the cloud server may learn about the virtual device to which the current task to be executed belongs based on the identifier of the communication interface used to transmit the current task to be executed.

Here, this communication interface may achieve communication by creating node files.

The virtual device is a standalone device, such as a container, a sandbox, etc., that runs directly based on the operating system of a physical device. Here, the physical device may refer to a physical host, such as a cloud server host. It should be understood that, for a virtual machine, it is not running directly based on the operating system of the physical device, but indirectly based on that operating system, although the virtual machine has its own virtual

operating system and also virtual hardware resources, etc., such as a virtual CPU, these are not really hardware resources of the physical device, but a virtualization of the hardware resources.

5 In addition, virtual devices capable of implementing the embodiments of the present application are within the scope of protection of the embodiments of the present application, and are not limited to containers.

It should also be noted that the virtual device is already created on the physical device. Or, in other words, the physical device may create the virtual device as needed. As the creation process belongs to the prior art, it will not be elaborated here. It should be
10 understood that embodiments of the present application are based on the situation where the physical device has already created multiple virtual devices, or at least one virtual device. Here, the virtual device may create a user mode in the physical device, wherein the user mode is a privilege level in the operating system (e.g., Linux operating system), and in the user mode, programs therein are not allowed to perform operations that require a privileged
15 mode in the processor (e.g., CPU, GPU, etc.) in order to avoid crashes of the operating system.

Specifically, acquiring task information of a current task to be executed includes: acquiring the task information according to a storage address carried by the current task to be executed.

20 Here, the task information describes the current task to be executed, e.g., an execution instruction for the current task to be executed.

The storage address is an address for the storage of the task information, such as a buffer address of a buffer or a cache address of a cache.

It should be noted that, in addition to carrying the storage address, the task to be executed may also has other information, such as that indicating which processing device is to perform task processing (carrying the identifier of the processing device), and the like.

For example, the task scheduler in the cloud server receives the current task to be
5 executed a (hereafter also referred to as task a), reads the current task to be executed a, and
acquires from it a buffer address of a command buffer that it carries. The task scheduler
reads this buffer address and reads the corresponding execution instruction. That is, as shown
in FIG. 3, step 301 is executed: reading information in the command buffer.

It should be noted that this buffer address may be the start address of the corresponding
10 buffer and the length of the buffer, which enable the task scheduler to read the execution
instruction from the corresponding buffer location in the corresponding buffer. It may also
be the head and tail pointers of the buffer, which enable the task scheduler to read the
execution instruction from the corresponding buffer location in the corresponding buffer. If
there are multiple buffers, it may also carry information such as the identifier of the buffer,
15 which enable the task scheduler to find the corresponding buffer.

202: determining an actual execution time for the current task to be executed according
to the task information.

Here, the actual execution time is the execution time for executing this task to be
executed and, in particular, the actual execution time for executing this task to be executed
20 by the processing device.

Determining an actual execution time for the current task to be executed according to the
task information includes: determining an actual execution time for executing the task
information as the actual execution time for the current task to be executed. That is, the
actual execution time obtained by executing the task information is used as the actual
25 execution time for executing this task to be executed.

Here, the actual execution time may be determined in a manner as follows.

1). Determining an actual execution time for executing the task information includes: inputting the task information into a preset model, and determining an output of the preset model as the actual execution time.

5 Here, the preset model is a model for predicting the actual execution time for the task information. The model may be a neural network model, and it may be generated through training. Before the training, the corresponding training data, such as multiple pieces of task information (e.g., multiple execution instructions) and the actual execution time for the corresponding processing device, may be acquired. During the training process, this preset
10 model is generated by analyzing the task information to acquire its task information features, and then performing training according to the corresponding actual execution time.

 It should be noted that since processing devices may be of different types, such as CPUs, GPUs, and NPUs, multiple corresponding preset models may be generated according to the different types of processing devices for use in predicting the actual execution time for each
15 of the processing devices, respectively.

 Once the model is generated, the prediction can be performed. The task scheduler in the cloud server may input the task information into a corresponding preset model, for example, if the execution instruction is for a GPU processing device, the execution instruction is input into a preset model corresponding to the GPU to obtain an output of that preset model for
20 use as the actual execution time for the GPU to execute the execution instruction. That is, as shown in FIG. 3, step 302 is implemented: determining the actual execution time for the processing device.

2). Determining an actual execution time for the current task to be executed according to the task information includes: matching the task information with multiple pieces of preset

task information, and selecting a preset execution time corresponding to matched preset task information according to a matching result as the actual execution time.

In addition, the cloud server may also acquire multiple execution instructions as well as actual execution times corresponding to different processing devices, and store such information to a local preset storage region, or a database server, and after the task scheduler in the cloud server acquires the task information, such as an execution instruction, it compares or matches that execution instruction with the acquired multiple execution instructions, and determines an execution instruction which has a similarity within a certain threshold or which is the same, and acquires the corresponding actual execution time according to the determined execution instruction.

It should be understood that the actual execution time may be different for different processing devices, so an execution instruction may store multiple actual execution times, and different processing devices correspond to different actual execution times, and a corresponding actual execution time is determined for the task scheduler based on the processing device required for the task information.

203: comparing the actual execution time with a preset scheduling time corresponding to the current task to be executed, and adjusting an actual scheduling time corresponding to a next task to be executed according to the comparison result so as to determine whether to schedule the next task to be executed.

Here, the preset scheduling time, which may also be referred to as a preset time slice, is the time used to occupy hardware resources, which is usually a period of time, such as 10 ms (milliseconds). Illustration will be provided below using an example of a processor. For example, for CPUs, multiple time slices may be divided for each CPU and assigned to different containers, so that the CPU can process the to-be-processed tasks from the containers according to the time slices, thus enabling the CPU to process multiple tasks over

a period of time. In addition, for a multi-core CPU, multiple time slices may be divided for each core and assigned to different containers, so that the CPU core can process the to-be-processed tasks from the containers according to the time slices, thus enabling the CPU to process multiple tasks over a period of time. The situation is similar for GPUs and NPUs, and will not be elaborated here. This may also be used as a basis for the division of other hardware resources. Therefore, the cloud server may divide time slices for different virtual devices, such as containers. The preset time slices for the same container are the same, i.e., the preset scheduling times are the same. Then, the preset time slices for different tasks to be executed from the same container are also the same. The time slices may be different or the same for different containers. The preset scheduling time may be pre-divided by the task scheduler.

Corresponding to the preset scheduling time, there is also an actual scheduling time. The actual scheduling time refers to the amount of time that the processing device can actually have to allow the processing device to perform task processing on the tasks to be executed, given the preset scheduling time.

Although a corresponding preset scheduling time is set for each container, the execution time for the task to be executed in a hardware resource such as a processing device may exceed the preset scheduling time, resulting in occupation of the preset scheduling time for a task to be executed that is later submitted. For example, for a GPU1 processing device, the task scheduler may first schedule task 1 from container 1 to GPU1 for task processing, but since GPU1 takes too much time to process this task 1, which exceeds the preset scheduling time, the processing time for a subsequent task is compressed. For example, the task waiting for GPU1 to process after task 1 at this moment is task 2 from container 2, since the scheduling time for task 2 is occupied, it cannot fully utilize its own preset scheduling time, which in turn affects the quality of service (QoS) of container 2.

Therefore, an actual scheduling time is set here to quantify how much time when each task in each container can actually occupy the processing device, i.e., how much time is actually available for each task.

For example, as described above, the task scheduler in the cloud server, upon receiving task a sent by container A, determines that this task a is targeted to GPU1 and also determines the actual execution time of the processing of task a through the GPU. The task scheduler compares the actual execution time of 15 ms with the preset scheduling time of 10 ms corresponding to container A and finds that the actual execution time is greater than the preset scheduling time, then it reduces the actual scheduling time for the next task to be executed from container A. That is, as shown in FIG. 3, step 303 is executed: reducing the next actual scheduling time. In this way, the scheduling time for the next task to be executed from container A is used to compensate for the scheduling time for a task to be processed from other containers.

In addition, the task scheduler compares the actual execution time of 7 ms with the preset scheduling time of 10 ms corresponding to container A and finds that the actual execution time is less than the preset scheduling time, then in this case, it may idle the corresponding processing device, or it may compensate for the actual scheduling time for the next task to be executed from container A. As shown in FIG. 3, step 305 is executed: determining an idle time or compensating for the next actual scheduling time.

It should be noted that if the task scheduler finds that the actual execution time is equal to the preset scheduling time corresponding to container A, it can either not perform adjustment, or maintain the current scheduling time that has been reduced (i.e., the actual scheduling time).

In addition, for the current task to be executed, if it is during the container initialization phase, that is, the current task to be executed sent by the container is the first task to be

executed from that container, the actual scheduling time is the preset scheduling time and may continue to be used as the actual scheduling time for the second task to be executed from that container for use in subsequent updates. At this point, before performing step 302, the task scheduler may perform step 306: submitting the task to the processing device for task processing by the processing device. Upon receiving task a from container A, the task scheduler acquires the execution instruction and then directly submits task a to the corresponding GPU for task processing. However, if the container has been running for a period of time, the task scheduler, upon receiving task a, may first detect whether the actual scheduling time for container A to which the current task a belongs is 0, and if so, it will not perform any processing and will not execute step 306, and will need to wait for the arrival of the next preset scheduling time before performing step 306 and the step of adjusting the actual scheduling time.

For the current task to be executed, after completing the adjustment of the actual scheduling time corresponding to the next task to be executed from the same source, the task scheduler may continue to schedule this next task to be executed and adjust the actual scheduling time. That is, step 304 is executed: performing the next task scheduling.

The specific implementation of step 203 is as follows.

Here, adjusting an actual scheduling time corresponding to a next task to be executed according to the comparison result includes: determining an excess time when the comparison result is that the actual execution time is greater than the preset scheduling time; and adjusting the actual scheduling time corresponding to the next task to be executed from the same source according to the excess time to obtain an updated actual scheduling time, and continuing to use it as an actual scheduling time for a subsequent task to be executed from the same source until a subsequent updated actual scheduling time is reduced to 0, wherein when the current task to be executed is an initial task to be executed from the same

source, an actual scheduling time corresponding to the current task to be executed is the same as the actual scheduling time corresponding to the next task to be executed from the same source, and is the preset scheduling time.

For example, as described above, the task scheduler receives task a (i.e., the current task
5 to be executed) sent from container A and acquires its execution instruction, and thereby determines that the corresponding actual execution time is 15 ms, and compares it with the preset scheduling time of 10 ms for container A, and determines that the excess time is 5 ms. Then, the next time the task scheduler receives a task sent by container A, such as task b, its actual scheduling time is $10 - 5 = 5$ ms, i.e., changing from the original 10 ms (in this case,
10 the preset scheduling time may be considered as the actual scheduling time in the case where there is no excess time, for example, if task a is the first task from container A, or if none of the previous tasks from container A has excess time or idle time, then the actual scheduling time for task a can be the preset scheduling time and used as the actual scheduling time for task b) to 5 ms, then the actual scheduling time for task b is updated to 5 ms and is used as
15 the actual scheduling time for the next task c from container A. However, since the actual scheduling time corresponding to task a is the preset scheduling time and is greater than 0, the task scheduler will send task a to the corresponding processing device, such as GPU1, before calculating the actual scheduling time of task b, and GPU1 can acquire the execution instruction from the storage address in task a so as to perform task processing. Since the
20 actual scheduling time for task b is updated to 5 ms, but is also greater than 0, task b may also be scheduled by the task scheduler and processed by the corresponding processing device. If the actual execution time for task b is also 15 ms, it is greater than the preset scheduling time of 10 ms, and the excess time is also 5 ms. Then for a further next task to be executed from container A, such as task c, its actual scheduling time will be updated from

5 ms to 0 ms, i.e., $5 - 5 = 0$. At this point, after receiving task c, the task scheduler does not schedule this task c upon the arrival of its preset scheduling time, but needs to wait.

It should be understood that the subsequent task to be executed may refer to a task that follows the current task to be executed or a task that follows the next task to be executed.

5 It should be noted that the adjustment of the actual scheduling time makes the scheduling time for other virtual devices such as other containers not to be occupied forcibly, thus ensuring the fairness of scheduling while meeting the quality of service (QoS) requirement, and also avoiding malicious occupation of time slices (i.e., the preset scheduling time) of other virtual devices such as other containers by some programs, thus ensuring the security
10 of the scheduling of the containers.

Based on the excess time, the actual scheduling time is adjusted, and the specific process is as follows.

Specifically, the method 200 further includes: reducing, for the current task to be executed, the excess time to the actual scheduling time corresponding to the current task to
15 be executed so as to update the corresponding actual scheduling time, and using the updated actual scheduling time as the actual scheduling time for the next task to be executed.

Since this has already been described above, it will not be elaborated here.

The waiting process described above may be specifically as follows: the method 200 further includes: when the subsequent updated actual scheduling time is 0, determining, upon
20 the arrival of a preset scheduling time corresponding to the subsequent task to be executed, not to schedule the subsequent task to be executed.

For example, as described above, after the task scheduler receives task c, its actual scheduling time is updated to 0, then the task scheduler waits for the arrival of the preset scheduling time corresponding to container A to which task c belongs and will not schedule

that task c anymore, that is, it will not send task c through the device driver to the corresponding processing device such as GPU1 for task processing.

However, if the actual scheduling time for task c is not 0 after updated, or if the actual scheduling time for a subsequent task is not 0 after updated, scheduling can be performed.

5 Specifically, the method 200 further includes: when the actual scheduling time corresponding to the subsequent task to be executed is not 0 after updated, determining, upon the arrival of a corresponding preset scheduling time, to schedule the subsequent task to be executed according to the actual execution time.

For example, as described above, after the task scheduler receives task b, as the actual
10 scheduling time is updated to 5 ms, then after receiving task b, the task scheduler sends, after waiting until the preset scheduling time for the container A arrives, task b through the driver device of the cloud server to the corresponding processing device such as GPU1 for task processing. Meanwhile, the actual execution time for GPU1 to process task b is 15 ms as predicted above.

15 It should be noted that since the actual execution time for task b is 15 ms, the task scheduler may access the processing device before the 15 ms by means of polling to determine whether it has finished the processing of task b, and send the next task when determining that it has finished the processing. Under normal circumstances, the processing device can finish the processing of the task at the moment of 15 ms or, alternatively, can
20 finish the processing of the task within a margin of error for 15 ms.

After the scheduling of the current task to be executed is completed, it can be determined whether to schedule the next task to be executed, and after it is determined that scheduling is to be performed, task scheduling is performed.

Specifically, the method 200 further includes: after determining to schedule the next task
25 to be executed, receiving the next task to be executed, and waiting for the arrival of the

corresponding preset scheduling time, and then sending the next task to be executed to a corresponding processing device so as to perform task processing according to the corresponding actual execution time.

Since this has already been described above, it will not be elaborated here. In the case of
5 not scheduling the subsequent task to be executed, its preset scheduling time may be compensated to tasks from other sources, thus allowing the scheduling time for the tasks to be balanced.

Specifically, the method 200 further includes: receiving the subsequent task to be executed and, upon the arrival of the corresponding preset scheduling time, sending a task
10 to be executed from other sources to a corresponding processing device so as to perform task processing according to a corresponding actual execution time.

For example, as described above, the task scheduler does not schedule task c upon the arrival of the corresponding preset scheduling time, but may schedule other tasks, such as a task from other containers that follows that task c, such as task m from container B. This
15 enables scheduling of subsequent schedulable tasks from other containers before the arrival of the next preset scheduling time for container A, thus sharing the current scheduling time for container A with other containers for use.

Since the scheduling of the tasks to be executed from other sources by the task scheduler is the same as the scheduling process described above, it will not be elaborated here. Just for
20 the sake of illustration, the tasks from whichever container can all be scheduled using the scheduling method of the embodiments of the present application. For the tasks from other containers, there are also cases of updating the actual scheduling time and of waiting for tasks to be scheduled, which will not be discussed in detail here.

In addition, since there are multiple containers in the cloud server, the task scheduler
25 may receive multiple tasks to be executed, and these tasks need to be queued for processing,

in which case after the arrival of the preset scheduling times for the containers they belong to, and in this queued order, the tasks are processed sequentially.

In the case where a task is not scheduled in time by the task scheduler but is determined to wait to be scheduled, after waiting until it can be scheduled, its specific scheduling process may be as follows: the method 200 further includes: waiting for the arrival of a corresponding next preset scheduling time, and then sending the subsequent task to be executed to a corresponding processing device so as to perform task processing according to a corresponding actual execution time, wherein upon the arrival of the corresponding next preset scheduling time, the actual scheduling time corresponding to the subsequent task to be executed is equal to the preset scheduling time.

For example, as described above, the task scheduler does not schedule task c, but instead, after determining that task c is in the waiting state, wait for the arrival of the next preset scheduling time for the container A to which it belongs

It should be noted that in the embodiments of the present application, although a task in a container may occupy the scheduling time for a task from another container, the scheduling time for the another container will be compensated. However, after the scheduling time for the another container is occupied, the task from the another container may not be scheduled to the corresponding processing device exactly on time. That is, the task from the another container cannot be scheduled immediately after the arrival of the preset scheduling time, but there will be a delay of some time. Nevertheless, the task can only be scheduled after the corresponding preset scheduling time has arrived, and the task can be scheduled to the corresponding processing device only after the task scheduler has polled and found that the corresponding processing device has completed the current task.

In addition to the actual execution time being greater than the preset scheduling time, there is also a possibility that the actual execution time is less than the preset scheduling time, in which case the actual scheduling time is adjusted as follows.

Specifically, adjusting an actual scheduling time corresponding to a next task to be
5 executed according to the comparison result includes: determining an idle time when the
comparison result is that the actual execution time is less than the preset scheduling time;
and adjusting the actual scheduling time corresponding to the next task to be executed from
the same source according to the idle time to obtain an updated actual scheduling time, and
continuing to use it as an actual scheduling time for a subsequent task to be executed from
10 the same source.

For example, as described above, the task scheduler receives task a (i.e., the current task
to be executed) sent from container A and acquires its execution instruction, and thereby
determines that the corresponding actual execution time is 7 ms, and compares it with the
preset scheduling time of 10 ms for container A, and determines that the idle time is 3 ms.
15 Then, the next time the task scheduler receives a task sent by container A, such as task b, its
actual scheduling time is $10 + 3 = 13$ ms, i.e., changing from the original 10 ms (in this case,
the preset scheduling time may be considered as the actual scheduling time in the case where
there is no excess time, for example, if task a is the first task from container A, or if none of
the previous tasks from container A has excess time or idle time, then the actual scheduling
20 time for task a can be the preset scheduling time and used as the actual scheduling time for
task b) to 13 ms, then the actual scheduling time for task b is updated to 13 ms and is used
as the actual scheduling time for the next task c from container A.

It should be noted that, in addition to giving its idle time to a task from the same container,
it is also possible to idle that idle time directly, thus enabling the corresponding processing

device to be idle. In addition, it is also possible to distribute the idled idle time to tasks from other containers to provide compensation or help, and the like.

After determining that there is idle time, the task can be scheduled.

Specifically, the method 200 further includes: determining to schedule the next task to
5 be executed when the comparison result is that the actual execution time is less than or equal to the preset scheduling time.

For example, as described above, if the task scheduler determines that task a has an idle time of 3 ms, the actual scheduling time for task b is updated to 13 ms, which is greater than 0, then it can schedule task b upon receiving task b.

10 It should be understood that if the actual execution time for task a is 10 ms, which is equal to the preset scheduling time, then the actual scheduling time for task b will not be updated, but will remain at 10 ms, or some other value greater than 0 (if it is a value equal to 0, then task a is not schedulable, so the premise here is that task a is schedulable and is not in a waiting state), and task b may be scheduled.

15 Here, the specific process of adjusting the actual scheduling time based on the idle time is as follows.

Specifically, the method 200 further includes: increasing, for the current task to be executed, the idle time to the actual scheduling time corresponding to the current task to be executed so as to update the corresponding actual scheduling time, and using the updated
20 actual scheduling time as the actual scheduling time for the next task to be executed.

Since this has already been described above, it will not be elaborated here.

After the actual scheduling time is updated to 0, since the corresponding task is not schedulable for the time being, it needs to wait for the arrival of the next preset scheduling time before it can be scheduled. At that point, the task owes no more scheduling time to tasks

from other containers, so its actual scheduling time may also be reset to the preset scheduling time. The specific process is as follows.

Specifically, the method 200 further includes: if the updated actual scheduling time is 0, the actual scheduling time corresponding to the next task to be executed is equal to the preset
5 scheduling time in the case where it is determined that the next task to be executed can be scheduled.

For example, as described above, after the actual scheduling time for task c is updated to 0, the arrival of the next preset scheduling time for container A is waited for, and upon arrival, task c can be scheduled. Then its actual scheduling time may also be reset to 10 ms,
10 which is equal to the preset scheduling time, and used as the actual scheduling time for the next task to be executed from container A, such as task d, such that this actual scheduling time of 10 ms will be updated when the task scheduler subsequently receives this task d. For the specific update process, reference may be made to that described above.

It should be noted that the subsequent tasks that appear in the embodiments of the present
15 application, such as tasks b, c, d, etc., are intended only to allow a clear description of the ongoing example but not to indicate that these tasks are identified and that the present solution can only be executed if the identifiers are complied with; rather, these identifiers simply indicate that these tasks are subsequent tasks from the same container.

In addition, since in the process of adjusting the actual scheduling time, it may not be
20 updated exactly to 0 directly, but may be updated from 5 ms to -1 ms, in which case the adjustment process is as follows.

Specifically, the method 200 further includes: when the actual scheduling time corresponding to a subsequent task to be executed from the same source is directly reduced from a positive number to a negative number, determining a difference value between this
25 negative number and 0; adjusting the actual scheduling time corresponding to the subsequent

task to be executed from the same source to 0; receiving the subsequent task to be executed from the same source and, upon the arrival of the corresponding preset scheduling time, sending a task to be executed from other sources to a corresponding processing device; and waiting for the arrival of a corresponding next preset scheduling time, and then sending the subsequent task to be executed from the same source to a corresponding processing device so as to perform task processing according to a corresponding actual execution time, wherein upon the arrival of the corresponding next preset scheduling time, the actual scheduling time corresponding to the subsequent task to be executed from the same source is equal to the difference between the preset scheduling time and the difference value.

For example, as described above, the task scheduler receives task a (i.e., the current task to be executed) sent from container A and acquires its execution instruction, and thereby determines that the corresponding actual execution time is 16 ms, and compares it with the preset scheduling time of 10 ms for container A, and determines that the excess time is 6 ms. Then the next time the task scheduler receives a task sent by container A, such as task b, its actual scheduling time is $10 - 6 = 4$ ms, i.e., changing from the original 10 ms (in this case, the preset scheduling time may be considered as the actual scheduling time in the case where there is no excess time, for example, if task a is the first task from container A, or if none of the previous tasks from container A has excess time or idle time, then the actual scheduling time for task a can be the preset scheduling time and used as the actual scheduling time for task b) to 4 ms, then the actual scheduling time for task b is updated to 4 ms and is used as the actual scheduling time for the next task c from container A. After receiving task b, the task scheduler determines that its actual execution time is 16 ms and its excess time is 6 ms, then it updates its actual scheduling time to $4 - 6 = -2$ ms. At this point, the task scheduler may then directly update the actual scheduling time to 0 ms and use it as the actual scheduling time for task c, and at the same time determines 2 ms as the difference value. The

task scheduler may schedule task a and task b, but may not schedule task c for the time being but needs to wait. After the arrival of the corresponding preset scheduling time, task c is not scheduled for the time being, and a subsequent task from other containers may be scheduled, and the arrival of the next preset scheduling time is waited for, then task c may be scheduled.

- 5 Since at this time, the actual scheduling time for task c is 0, but in the case of rescheduling, its actual scheduling time can be reset to 10 ms, which is equal to the preset scheduling time, the actual scheduling time for the next task d from this container A can be updated to $10\text{ ms} - 2 = 8\text{ ms}$.

It should be noted that task scheduling in the embodiments of the present application is
10 independent of whether the device driver is open source or not, and can be easily applied to processing devices with open-source or non-open source device drivers and thus has a wider range of applications.

Based on the same inventive concept described above, FIG. 4 illustrates a flow chart of a scheduling method for container tasks provided by another exemplary embodiment of the
15 present application. The method 400 provided by the embodiments of the present application is executed by the above-mentioned cloud server and, more specifically, a task scheduler provided in the cloud server. As shown in FIG. 4, the method 400 includes the following steps:

401: receiving a current task to be executed sent by any container, and acquiring task
20 information of the current task to be executed, the task information describing the current task to be executed;

402: determining an actual execution time for the current task to be executed according to the task information; and

403: comparing the actual execution time with a preset scheduling time corresponding
25 to the current task to be executed, and adjusting an actual scheduling time corresponding to

a next task to be executed from the same container according to the comparison result so as to determine whether to schedule the next task to be executed from the same container.

Since the specific implementation of steps 401-403 has been described in detail above, it will not be elaborated here.

5 Here, receiving a current task to be executed sent by any container includes: receiving the current task to be executed according to a communication interface corresponding to the container; and the method 400 further includes: determining a container identifier according to the communication interface so as to determine a task to be executed from the same container according to the container identifier.

10 Since this has already been described above, it will not be elaborated here.

In addition, for the contents not described in detail in this method 400, reference may be made to the steps in the method 200 described above.

FIG. 5 is a structural framework schematic diagram of a scheduling apparatus for tasks provided in an exemplary embodiment of the present application. The apparatus 500 may be
15 applied to a cloud server and, more specifically, a task scheduler in the cloud server. The apparatus 500 includes acquisition module 501, determination module 502, and adjustment module 503. A detailed description of the functions of each module will be given below.

Acquisition module 501 is used for acquiring task information of a current task to be executed, the task information describing the current task to be executed.

20 Determination module 502 is used for determining an actual execution time for the current task to be executed according to the task information.

Adjustment module 503 is used for comparing the actual execution time with a preset scheduling time corresponding to the current task to be executed, and adjusting an actual scheduling time corresponding to a next task to be executed according to the comparison
25 result so as to determine whether to schedule the next task to be executed.

Specifically, acquisition module 501 is used for acquiring the task information according to a storage address carried by the current task to be executed.

Specifically, determination module 502 is used for determining an actual execution time for executing the task information as the actual execution time for the current task to be
5 executed.

Specifically, determination module 502 is used for inputting the task information into a preset model, and determining an output of the preset model as the actual execution time.

Specifically, determination module 502 is used for matching the task information with multiple pieces of preset task information, and selecting a preset execution time
10 corresponding to matched preset task information according to a matching result as the actual execution time.

Specifically, adjustment module 503 includes: a determination unit for determining an excess time when the comparison result is that the actual execution time is greater than the preset scheduling time; and an adjustment unit for adjusting the actual scheduling time
15 corresponding to the next task to be executed from the same source according to the excess time to obtain an updated actual scheduling time, and continuing to use it as an actual scheduling time for a subsequent task to be executed from the same source until a subsequent updated actual scheduling time is reduced to 0, wherein when the current task to be executed is an initial task to be executed from the same source, an actual scheduling time
20 corresponding to the current task to be executed is the same as the actual scheduling time corresponding to the next task to be executed from the same source, and is the preset scheduling time.

In addition, the apparatus 500 further includes: a scheduling module for, when the subsequent updated actual scheduling time is 0, determining, upon the arrival of a preset

scheduling time corresponding to the subsequent task to be executed, not to schedule the subsequent task to be executed.

In addition, the scheduling module is further used for: when the actual scheduling time corresponding to the subsequent task to be executed is not 0 after updated, determining, upon
5 the arrival of a corresponding preset scheduling time, to schedule the subsequent task to be executed according to the actual execution time.

In addition, the apparatus 500 further includes: a sending module for: after determining to schedule the next task to be executed, receiving the next task to be executed, and waiting for the arrival of the corresponding preset scheduling time, and then sending the next task to
10 be executed to a corresponding processing device so as to perform task processing according to the corresponding actual execution time.

In addition, the sending module is further used for: receiving the subsequent task to be executed and, upon the arrival of the corresponding preset scheduling time, sending a task to be executed from other sources to a corresponding processing device so as to perform task
15 processing according to a corresponding actual execution time.

In addition, the sending module is further used for: waiting for the arrival of a corresponding next preset scheduling time, and then sending the subsequent task to be executed to a corresponding processing device so as to perform task processing according to a corresponding actual execution time, wherein upon the arrival of the corresponding next
20 preset scheduling time, the actual scheduling time corresponding to the subsequent task to be executed is equal to the preset scheduling time.

In addition, the determination unit is further used for: determining an idle time when the comparison result is that the actual execution time is less than the preset scheduling time; and the adjustment unit is further used for: adjusting the actual scheduling time
25 corresponding to the next task to be executed from the same source according to the idle

time to obtain an updated actual scheduling time, and continuing to use it as an actual scheduling time for a subsequent task to be executed from the same source.

In addition, the determination unit is further used for: determining to schedule the next task to be executed when the comparison result is that the actual execution time is less than
5 or equal to the preset scheduling time.

In addition, the adjustment unit is further used for: increasing, for the current task to be executed, the idle time to the actual scheduling time corresponding to the current task to be executed so as to update the corresponding actual scheduling time, and using the updated actual scheduling time as the actual scheduling time for the next task to be executed.

10 In addition, the adjustment unit is further used for: reducing, for the current task to be executed, the excess time to the actual scheduling time corresponding to the current task to be executed so as to update the corresponding actual scheduling time, and using the updated actual scheduling time as the actual scheduling time for the next task to be executed.

In addition, the adjustment unit is further used for: if the updated actual scheduling time
15 is 0, the actual scheduling time corresponding to the next task to be executed is equal to the preset scheduling time in the case where it is determined that the next task to be executed can be scheduled.

Here, preset scheduling times corresponding to tasks to be executed from the same source are the same.

20 In addition, determination module 502 is further used for: when the actual scheduling time corresponding to a subsequent task to be executed from the same source is directly reduced from a positive number to a negative number, determining a difference value between this negative number and 0; adjustment module 503 is further used for: adjusting the actual scheduling time corresponding to the subsequent task to be executed from the
25 same source to 0; the sending module is further used for: receiving the subsequent task to be

executed from the same source and, upon the arrival of the corresponding preset scheduling time, sending a task to be executed from other sources to a corresponding processing device; and the sending module is further used for: waiting for the arrival of a corresponding next preset scheduling time, and then sending the subsequent task to be executed from the same source to a corresponding processing device so as to perform task processing according to a corresponding actual execution time, wherein upon the arrival of the corresponding next preset scheduling time, the actual scheduling time corresponding to the subsequent task to be executed from the same source is equal to the difference between the preset scheduling time and the difference value.

Specifically, acquisition module 501 is used for receiving, via a communication interface, the current task to be executed sent by a virtual device.

FIG. 6 is a structural framework schematic diagram of a scheduling apparatus for container tasks provided in an exemplary embodiment of the present application. The apparatus 600 may be applied to a cloud server and, more specifically, may be a task scheduler provided in the cloud server. The apparatus 600 includes receiving module 601, determination module 602, and adjustment module 603. A detailed description of the functions of each module will be given below.

Receiving module 601 is used for receiving a current task to be executed sent by any container, and acquiring task information of the current task to be executed, the task information describing the current task to be executed.

Determination module 602 is used for determining an actual execution time for the current task to be executed according to the task information.

Adjustment module 603 is used for comparing the actual execution time with a preset scheduling time corresponding to the current task to be executed, and adjusting an actual scheduling time corresponding to a next task to be executed from the same container

according to the comparison result so as to determine whether to schedule the next task to be executed from the same container.

Specifically, receiving module 601 is specifically used for: receiving the current task to be executed according to a communication interface corresponding to the container; and
5 determination module 602 is used for: determining a container identifier according to the communication interface so as to determine a task to be executed from the same container according to the container identifier.

It should be noted that for the parts of apparatus 600 that are not mentioned above, reference can be made to the contents of apparatus 500 described above.

10 The internal functions and structure of apparatus 500 shown in FIG. 5 are described above, and in a possible design, the structure of apparatus 500 shown in FIG. 5 may be implemented as a computing device, e.g., a server. As shown in FIG. 7, the device 700 may include: memory 701 and processor 702.

Memory 701 is used for storing a computer program.

15 Processor 702 is used for executing the computer program for: acquiring task information of a current task to be executed, the task information describing the current task to be executed; determining an actual execution time for the current task to be executed according to the task information; and comparing the actual execution time with a preset scheduling time corresponding to the current task to be executed, and adjusting an actual
20 scheduling time corresponding to a next task to be executed according to the comparison result so as to determine whether to schedule the next task to be executed.

Specifically, processor 702 is specifically used for: acquiring the task information according to a storage address carried by the current task to be executed.

Specifically, processor 702 is specifically used for: determining an actual execution time for executing the task information as the actual execution time for the current task to be executed.

Specifically, processor 702 is specifically used for: inputting the task information into a
5 preset model, and determining an output of the preset model as the actual execution time.

Specifically, processor 702 is specifically used for: matching the task information with multiple pieces of preset task information, and selecting a preset execution time according to a matching result as the actual execution time.

Specifically, processor 702 is specifically used for: determining an excess time when the
10 comparison result is that the actual execution time is greater than the preset scheduling time; and adjusting the actual scheduling time corresponding to the next task to be executed from the same source according to the excess time to obtain an updated actual scheduling time, and continuing to use it as an actual scheduling time for a subsequent task to be executed from the same source until a subsequent updated actual scheduling time is reduced to 0,
15 wherein when the current task to be executed is an initial task to be executed from the same source, an actual scheduling time corresponding to the current task to be executed is the same as the actual scheduling time corresponding to the next task to be executed from the same source, and is the preset scheduling time.

In addition, processor 702 is further used for: when the subsequent updated actual
20 scheduling time is 0, determining, upon the arrival of a preset scheduling time corresponding to the subsequent task to be executed, not to schedule the subsequent task to be executed.

In addition, processor 702 is further used for: when the actual scheduling time corresponding to the subsequent task to be executed is not 0 after updated, determining, upon the arrival of a corresponding preset scheduling time, to schedule the subsequent task to be
25 executed according to the actual execution time.

In addition, processor 702 is further used for: after determining to schedule the next task to be executed, receiving the next task to be executed, and waiting for the arrival of the corresponding preset scheduling time, and then sending the next task to be executed to a corresponding processing device so as to perform task processing according to the corresponding actual execution time.

In addition, processor 702 is further used for: receiving the subsequent task to be executed and, upon the arrival of the corresponding preset scheduling time, sending a task to be executed from other sources to a corresponding processing device so as to perform task processing according to a corresponding actual execution time.

In addition, processor 702 is further used for: waiting for the arrival of a corresponding next preset scheduling time, and then sending the subsequent task to be executed to a corresponding processing device so as to perform task processing according to a corresponding actual execution time, wherein upon the arrival of the corresponding next preset scheduling time, the actual scheduling time corresponding to the subsequent task to be executed is equal to the preset scheduling time.

In addition, processor 702 is further used for: determining an idle time when the comparison result is that the actual execution time is less than the preset scheduling time; and adjusting the actual scheduling time corresponding to the next task to be executed from the same source according to the idle time to obtain an updated actual scheduling time, and continuing to use it as an actual scheduling time for a subsequent task to be executed from the same source.

In addition, processor 702 is further used for: determining to schedule the next task to be executed when the comparison result is that the actual execution time is less than or equal to the preset scheduling time.

In addition, processor 702 is further used for: increasing, for the current task to be executed, the idle time to the actual scheduling time corresponding to the current task to be executed so as to update the corresponding actual scheduling time, and using the updated actual scheduling time as the actual scheduling time for the next task to be executed.

5 In addition, processor 702 is further used for: reducing, for the current task to be executed, the excess time to the actual scheduling time corresponding to the current task to be executed so as to update the corresponding actual scheduling time, and using the updated actual scheduling time as the actual scheduling time for the next task to be executed.

In addition, processor 702 is further used for: if the updated actual scheduling time is 0,
10 the actual scheduling time corresponding to the next task to be executed is equal to the preset scheduling time in the case where it is determined that the next task to be executed can be scheduled.

Here, preset scheduling times corresponding to tasks to be executed from the same source are the same.

15 In addition, processor 702 is further used for: when the actual scheduling time corresponding to a subsequent task to be executed from the same source is directly reduced from a positive number to a negative number, determining a difference value between this negative number and 0; adjusting the actual scheduling time corresponding to the subsequent task to be executed from the same source to 0; receiving the subsequent task to be executed
20 from the same source and, upon the arrival of the corresponding preset scheduling time, sending a task to be executed from other sources to a corresponding processing device; and the sending module is further used for: waiting for the arrival of a corresponding next preset scheduling time, and then sending the subsequent task to be executed from the same source to a corresponding processing device so as to perform task processing according to a
25 corresponding actual execution time, wherein upon the arrival of the corresponding next

preset scheduling time, the actual scheduling time corresponding to the subsequent task to be executed from the same source is equal to the difference between the preset scheduling time and the difference value.

Specifically, processor 702 is specifically used for: receiving, via a communication
5 interface, the current task to be executed sent by a virtual device.

In addition, the embodiments of the present invention provide a computer storage medium, wherein a computer program, when executed by one or more processors, causes the one or more processors to implement steps of a scheduling method for tasks of the method embodiment of FIG. 2.

10 The internal functions and structure of apparatus 600 shown in FIG. 6 are described above, and in a possible design, the structure of apparatus 600 shown in FIG. 6 may be implemented as a computing device, e.g., a server. As shown in FIG. 8, the device 800 may include: memory 801 and processor 802.

Memory 801 is used for storing a computer program.

15 Processor 802 is used for executing the computer program for: receiving a current task to be executed sent by any container, and acquiring task information of the current task to be executed, the task information describing the current task to be executed; determining an actual execution time for the current task to be executed according to the task information; and comparing the actual execution time with a preset scheduling time corresponding to the
20 current task to be executed, and adjusting an actual scheduling time corresponding to a next task to be executed from the same container according to the comparison result so as to determine whether to schedule the next task to be executed from the same container.

Specifically, processor 802 is specifically used for: receiving the current task to be executed according to a communication interface corresponding to the container; and
25 processor 802 is further used for: determining a container identifier according to the

communication interface so as to determine a task to be executed from the same container according to the container identifier.

It should be noted that for the parts of the device 800 that are not mentioned above, reference can be made to the contents of device 700 described above.

5 In addition, the embodiments of the present invention provide a computer storage medium, wherein a computer program, when executed by one or more processors, causes the one or more processors to implement steps of a scheduling method for container tasks of the method embodiment of FIG. 4.

Further, some of the processes described in the above embodiments and the
10 accompanying drawings contain multiple operations appearing in a particular order, but it should be clearly understood that these operations may be executed not in the particular order in which they appear herein or in parallel, and that the serial numbers of the operations, such as 201, 202, 203, etc., are used only to distinguish the various different operations, and the serial numbers themselves do not represent any order of execution. Alternatively, these
15 processes may include more or fewer operations, and these operations may be executed sequentially or in parallel. It should be noted that the descriptions "first" and "second" herein are used to distinguish different messages, devices, modules, etc. and do not represent sequential order, nor do they limit the "first" and "second" to be different types.

The apparatus embodiments described above are merely schematic, where the units
20 described as separate components may or may not be physically separated, and the components displayed as units may or may not be physical units, i.e., may be located in one place or may be distributed to multiple network units. Part or all of the modules may be selected according to actual needs to achieve the purpose of the solution of the present embodiment. Those of ordinary skill in the art may understand and implement the solution
25 of the present embodiment without creative effort.

From the description of the above implementations, those skilled in the art may clearly understand that the various implementations may be implemented by means of software plus a necessary general hardware platform, and definitely may also be implemented by a combination of hardware and software. Based on such understanding, the above technical solution essentially or the part contributing to the prior art may be embodied in the form of a computer product. The present invention can be in the form of a computer program product implemented on one or more computer-usable storage media (including, but not limited to, a magnetic disk memory, a CD-ROM, an optical memory, and the like) including computer usable program code.

The present invention is described with reference to flow charts and/or block diagrams of methods, devices (systems), and computer program products according to embodiments of the present invention. It should be understood that each flow and/or block in the flow charts and/or block diagrams and a combination of flows and/or blocks in the flow charts and/or block diagrams may be implemented by computer program instructions. These computer program instructions can be provided to a processor of a general-purpose computer, a special-purpose computer, an embedded processor, or another programmable multimedia data processing device to produce a machine, so that the instructions executed by the processor of the computer or another programmable multimedia data processing device produce an apparatus for realizing the functions specified in one or more flows of the flow charts and/or one or more blocks of the block diagrams.

These computer program instructions may also be stored in a computer-readable memory that can direct a computer or another programmable multimedia data processing device to work in a specific manner, so that the instructions stored in this computer-readable memory produce an article of manufacture including an instruction apparatus which implements the

functions specified in one or more flows of the flow charts and/or one or more blocks of the block diagrams.

These computer program instructions can also be loaded onto a computer or another programmable multimedia data processing device, so that a series of operation steps are performed on the computer or another programmable device to produce computer-implemented processing, so that the instructions executed on the computer or another programmable device provide steps for implementing the functions specified in one or more flows of the flow charts and/or one or more blocks of the block diagrams.

In a typical configuration, the computing device includes one or more processors (CPUs), an input/output interface, a network interface, and a memory.

The memory may include a volatile memory, a random access memory (RAM), and/or a non-volatile memory in a computer-readable medium, e.g., a read-only memory (ROM) or a flash RAM. The memory is an example of the computer-readable medium.

The computer-readable medium includes permanent and non-permanent, removable and non-removable media, which may implement storage of information by using any method or technology. The information may be a computer-readable instruction, a data structure, a module of a program, or other data. Examples of computer storage media include, but are not limited to, a phase change memory (PRAM), a static random access memory (SRAM), a dynamic random access memory (DRAM), other type of random access memory (RAM), a read-only memory (ROM), an electrically erasable programmable read-only memory (EEPROM), a flash memory or other memory technology, a compact disc read-only memory (CD-ROM), a digital versatile disc (DVD) or other optical storage, a cassette magnetic tape, tape and disk storage or other magnetic storage device, or any other non-transmission media, which may be configured to store information accessible by a computing device. As defined

herein, the computer-readable medium does not include computer-readable transitory media, such as a modulated data signal and a carrier.

Finally, it should be noted that the above embodiments are only used to illustrate the technical solutions of the present invention, and are not intended to limit the present invention. Although the present invention has been described in detail with reference to the foregoing embodiments, those of ordinary skill in the art should understand that the technical solutions recorded in the foregoing various embodiments can still be modified, or some of the technical features thereof may be equivalently replaced. These modifications or replacements do not cause the essence of the corresponding technical solutions to deviate from the spirit and scope of the technical solutions of the embodiments of the present invention.

CLAIMS

1. A scheduling method for tasks, comprising:

acquiring task information of a current task to be executed, the task information describing the current task to be executed;

5 determining an actual execution time for the current task to be executed according to the task information; and

comparing the actual execution time with a preset scheduling time corresponding to the current task to be executed, and adjusting an actual scheduling time corresponding to a next task to be executed according to the comparison result so as to determine whether to schedule the next
10 task to be executed.

2. The method according to claim 1, wherein said acquiring task information of a current task to be executed comprises:

acquiring the task information according to a storage address carried by the current task to be executed.

15 3. The method according to claim 1, wherein said determining an actual execution time for the current task to be executed according to the task information comprises:

determining an actual execution time for executing the task information as the actual execution time for the current task to be executed.

20 4. The method according to claim 3, wherein said determining an actual execution time for executing the task information comprises:

inputting the task information into a preset model, and determining an output of the preset model as the actual execution time.

5. The method according to claim 1, wherein said determining an actual execution time for the current task to be executed according to the task information comprises:

matching the task information with multiple pieces of preset task information, and selecting a preset execution time corresponding to matching preset task information according to the matching result as the actual execution time.

6. The method according to claim 1, wherein said adjusting an actual scheduling time
5 corresponding to a next task to be executed according to the comparison result comprises:

determining an excess time when the comparison result is that the actual execution time is greater than the preset scheduling time; and

adjusting the actual scheduling time corresponding to the next task to be executed from the same source according to the excess time to obtain an updated actual scheduling time, and
10 continuing to use it as an actual scheduling time for a subsequent task to be executed from the same source until a subsequent updated actual scheduling time is reduced to 0;

wherein when the current task to be executed is an initial task to be executed from the same source, an actual scheduling time corresponding to the current task to be executed is the same as the actual scheduling time corresponding to the next task to be executed from the same source, and
15 is the preset scheduling time.

7. The method according to claim 6, further comprising:

when the subsequent updated actual scheduling time is 0, determining, upon the arrival of a preset scheduling time corresponding to the subsequent task to be executed, not to schedule the subsequent task to be executed.

20 8. The method according to claim 1, further comprising:

when the actual scheduling time corresponding to the subsequent task to be executed is not 0 after updated, determining, upon the arrival of a corresponding preset scheduling time, to schedule the subsequent task to be executed according to the actual execution time.

9. The method according to claim 1 or 8, further comprising:

25 after determining to schedule the next task to be executed, receiving the next task to be executed, and waiting for the arrival of a corresponding preset scheduling time, and then sending

the next task to be executed to a corresponding processing device so as to perform task processing according to the corresponding actual execution time.

10. The method according to claim 7, further comprising:

receiving the subsequent task to be executed and, upon the arrival of the corresponding preset
5 scheduling time, sending a task to be executed from other sources to a corresponding processing device so as to perform task processing according to a corresponding actual execution time.

11. The method according to claim 10, further comprising:

waiting for the arrival of a corresponding next preset scheduling time, and then sending the
subsequent task to be executed to a corresponding processing device so as to perform task
10 processing according to a corresponding actual execution time,

wherein upon the arrival of the corresponding next preset scheduling time, the actual scheduling time corresponding to the subsequent task to be executed is equal to the preset scheduling time.

12. The method according to claim 1, wherein said adjusting an actual scheduling time
15 corresponding to a next task to be executed according to the comparison result comprises:

determining an idle time when the comparison result is that the actual execution time is less than the preset scheduling time; and

adjusting the actual scheduling time corresponding to the next task to be executed from the same source according to the idle time to obtain an updated actual scheduling time, and continuing
20 to use it as an actual scheduling time for a subsequent task to be executed from the same source.

13. The method according to claim 12, further comprising:

determining to schedule the next task to be executed when the comparison result is that the actual execution time is less than or equal to the preset scheduling time.

14. The method according to claim 12, further comprising:

25 adding, for the current task to be executed, the idle time to the actual scheduling time corresponding to the current task to be executed so as to update the corresponding actual scheduling

time, and using the updated actual scheduling time as the actual scheduling time for the next task to be executed.

15. The method according to any one of claims 6-12, wherein preset scheduling times corresponding to tasks to be executed from the same source are the same.

5 16. The method according to claim 1, wherein said acquiring task information of a current task to be executed comprises:

receiving, via a communication interface, the current task to be executed sent by a virtual device.

17. A scheduling method for container tasks, comprising:

10 receiving a current task to be executed sent by any container, and acquiring task information of the current task to be executed, the task information describing the current task to be executed;

determining an actual execution time for the current task to be executed according to the task information; and

15 comparing the actual execution time with a preset scheduling time corresponding to the current task to be executed, and adjusting an actual scheduling time corresponding to a next task to be executed from the same container according to the comparison result so as to determine whether to schedule the next task to be executed from the same container.

18. The method according to claim 17, wherein said receiving a current task to be executed sent by any container comprises:

20 receiving the current task to be executed according to a communication interface corresponding to the container; and

the method further comprises:

determining a container identifier according to the communication interface so as to determine a task to be executed from the same container according to the container identifier.

25 19. A scheduling apparatus for tasks, comprising:

an acquisition module for acquiring task information of a current task to be executed, the task information describing the current task to be executed;

a determination module for determining an actual execution time for the current task to be executed according to the task information; and

5 an adjustment module for comparing the actual execution time with a preset scheduling time corresponding to the current task to be executed, and adjusting an actual scheduling time corresponding to a next task to be executed according to the comparison result so as to determine whether to schedule the next task to be executed.

20. A scheduling apparatus for container tasks, comprising:

10 a receiving module for receiving a current task to be executed sent by any container, and acquiring task information of the current task to be executed, the task information describing the current task to be executed;

a determination module for determining an actual execution time for the current task to be executed according to the task information; and

15 an adjustment module for comparing the actual execution time with a preset scheduling time corresponding to the current task to be executed, and adjusting an actual scheduling time corresponding to a next task to be executed from the same container according to the comparison result so as to determine whether to schedule the next task to be executed from the same container.

21. A computing device comprising: a memory and a processor, wherein

20 the memory is used for storing a computer program; and

the processor is used for executing the computer program for:

acquiring task information of a current task to be executed, the task information describing the current task to be executed;

determining an actual execution time for the current task to be executed according to the task
25 information; and

comparing the actual execution time with a preset scheduling time corresponding to the current task to be executed, and adjusting an actual scheduling time corresponding to a next task to be executed according to the comparison result so as to determine whether to schedule the next task to be executed.

5 22. A computing device comprising: a memory and a processor, wherein

the memory is used for storing a computer program; and

the processor is used for executing the computer program for:

receiving a current task to be executed sent by any container, and acquiring task information of the current task to be executed, the task information describing the current task to be executed;

10 determining an actual execution time for the current task to be executed according to the task information; and

comparing the actual execution time with a preset scheduling time corresponding to the current task to be executed, and adjusting an actual scheduling time corresponding to a next task to be executed from the same container according to the comparison result so as to determine whether to schedule the next task to be executed from the same container.

15

23. A computer-readable storage medium storing a computer program, characterized in that the computer program, when executed by one or more processors, causes the one or more processors to implement steps in the method of any one of claims 1-18.