

RESOURCE MANAGEMENT SYSTEM AND METHOD

FIELD

[0001] The present disclosure relate to the technical field of resource management, and in particular, to a resource management system and method.

BACKGROUND

[0002] With the rapid development of computer and internet technologies, containerization and microservice architectures have become the core paradigms supporting modern cloud-native applications. Container orchestration platforms, such as Kubernetes, manage application resources by defining resource requests and limits, striving to achieve a balance between stability and utilization. However, existing mainstream technical solutions have exposed a series of intractable contradictions in practice. The Horizontal Pod Autoscaler (HPA) handles load by adding or removing instances, but fails to solve per-instance resource bottlenecks and suffers from significant response latency. The Vertical Pod Autoscaler (VPA) can adjust the specification of a single container, but its "black box" decision-making mode that relies on historical monitoring data has inherent lag, and usually requires the evicting and restarting of Pods to take effect, resulting in service interruptions and state loss. The more fundamental problem lies in the fact that the infrastructure layer is completely oblivious to the critical intentions within the application (e.g., the upcoming big data sorting or model loading), and the resource supply often fails to match the actual requirement in terms of both time and space. Therefore, there is a dilemma that a large amount of resources may be wasted if peak resources are statically configured for stability, and concurrent multi-tasks can easily trigger node-level memory exhaustion and unexpected business crashes if oversubscription is set to improve utilization. Therefore, an effective solution is urgently needed to address the above issues.

SUMMARY

[0003] In view of this, a resource management system is provided according to an embodiment of the present disclosure. One or more embodiments of the present disclosure further relate to a resource management method, a computing device, a computer-readable storage medium, and a computer program product, to solve the technical deficiencies in the conventional art.

[0004] According to a first aspect of the embodiments of the present disclosure, a resource management system is provided, which includes a resource collaboration component, a resource controller and an agent node. The resource collaboration component is configured to determine a heavy-load task to be executed by a target application, calculate resource requirement information corresponding to the heavy-load task, and send a resource adjustment request to the resource controller according to the resource requirement information. The resource controller is configured to determine a target container associated with the target application in response to the resource adjustment request, and perform a patch update on a resource definition field of the target container according to the resource adjustment request. The agent node is configured to modify a control group configuration file of the target container upon detecting a patch update operation. The resource controller is configured to send, if it is determined that the modified control group configuration file has taken effect, resource adjustment completion information to the target application for the target application to execute the heavy-load task.

[0005] According to a second aspect of the embodiments of the present disclosure, a resource management method is provided, which is applied to a resource management system. The system includes a resource collaboration component, a resource controller and an agent node, and the method includes:

determining, by the resource collaboration component, a heavy-load task to be executed by a target application, calculating resource requirement information corresponding to the heavy-load task, and sending a resource adjustment request to the resource controller according to the resource requirement information;

determining, by the resource controller, a target container associated with the target application in response to the resource adjustment request, and performing a patch update on a resource definition field of the target container according to the resource adjustment request;

modifying, by the agent node, a control group configuration file of the target container upon detecting a patch update operation, ; and

sending, by the resource controller, resource adjustment completion information to the target application if it is determined that the modified control group configuration file has taken effect, for the target application to execute the heavy-load task.

[0006] According to a third aspect of the embodiments of the present disclosure, a computing device is provided, which includes a memory and a processor. The memory is configured to store computer executable instructions, the processor is configured to execute the computer executable instructions, where the computer executable instructions, when executed by the processor, perform the steps of the resource management method described above.

[0007] According to a fourth aspect of the embodiments of the present disclosure, a computer-readable storage medium is provided, which includes computer executable instructions stored thereon, where the computer executable instructions, when executed by a processor, perform the steps of the resource management method described above.

[0008] According to a fifth aspect of the embodiments of the present disclosure, a computer program product is provided, which includes computer programs or instructions, where the computer programs or instructions, when executed by a processor, perform the steps of the resource management method described above.

[0009] In the resource management system according to embodiments of the present disclosure, in order to achieve active and lossless resource scaling operation, the resource collaboration component may calculate resource requirement information corresponding to the heavy-load task upon determining a heavy-load task to be executed by a target application, and send a resource adjustment request to the resource controller based on the resource requirement information. The resource controller may determine a target container associated with the target application in response to the resource adjustment request, and perform a patch update on a resource definition field of the target container based on the resource adjustment request. The agent node may directly modify a control group configuration file of the target container upon detecting a patch update operation, thereby achieving resource allocation without rebuilding or restarting the container. Thereafter, the resource controller may send resource adjustment completion information to the target application upon determining that the modified control group configuration file has taken effect, so that the target application executes the heavy-load task. Therefore, through the standard Kubernetes API, the Pod resource definition is declaratively updated in a patch manner, and the new resource quotas are dynamically injected online into the container control mechanism of the operating system kernel relying on the cooperation of kubelet, thereby achieving seamless, instant resource

expansion, effectively avoiding system-wide crashes caused by resource contention, and providing agile, seamless, and business-transparent resource supply, thus significantly improving system stability.

BRIEF DESCRIPTION OF THE DRAWINGS

5 **[0010]** Figure 1 is a schematic structural diagram of a resource management system according to an embodiment of the present disclosure;

[0011] Figure 2 is a schematic diagram of a resource management system according to an embodiment of the present disclosure;

10 **[0012]** Figure 3 is a flow chart of a resource management method according to an embodiment of the present disclosure; and

[0013] Figure 4 is a structural block diagram of a computing device according to an embodiment of the present disclosure.

DETAILED DESCRIPTION OF EMBODIMENTS

15 **[0014]** Many specific details are set forth in the following description to facilitate a full understanding of the present disclosure. However, the present disclosure may be implemented in many other ways different from those described herein, and those skilled in the art may make similar generalizations without violating the connotations of the present disclosure, and thus the present disclosure is not limited by the specific embodiments described below.

20 **[0015]** Terms used in one or more embodiments of the present disclosure are used only for the purpose of describing a particular embodiment and are not intended to limit one or more embodiments of the present disclosure. The singular forms of "a", "an", and "the" used in one or more embodiments of the present disclosure and the claims are also intended to encompass the plural form, unless the context clearly indicates otherwise. It should also be understood that the term "and/or" used in one or more embodiments of the present disclosure refers to and
25 encompasses any or all possible combinations of one or more of the listed items in association.

[0016] It should be understood that while the terms "first", "second", etc. may be used in one or more embodiments of the present disclosure to describe various information, such information should not be limited to these terms. These terms are used only to distinguish the
30 same type of information from one another. For example, without departing from the scope of

the one or more embodiments of the present disclosure, "a first..." may also be referred to as "a second...", and similarly, "a second..." may be referred to as "a first...". Depending on the context, the phrase "if" as used herein may be interpreted as "at the time of" or "when" or "in response to a determination".

5 **[0017]** In addition, it should be noted that the user information (including but not limited to user device information, user personal information, etc.) and data (including but not limited to data used for analysis, stored data, displayed data, etc.) involved in one or more embodiments of the present disclosure are authorized by the user or sufficiently authorized by the parties, and that the collection, use, and processing of the relevant data are subject to relevant laws,
10 regulations, and regulations of relevant countries and regions, and a corresponding operation portal is provided for the user to choose to authorize or reject.

[0018] First, the terminologies involved in one or more embodiments of the present disclosure are explained below.

[0019] Operator refers to a custom control logic running in a Kubernetes environment, and
15 is used to automatically manage the lifecycle of specific applications or resources.

[0020] In-place Resizing refers to a technology that dynamically modifies the CPU or memory configuration of a Pod container without destroying or restarting the Pod container.

[0021] Request and Limit refer to two parameters in Kubernetes that define resource reservation (a guaranteed value) and upper limit (a maximum value).

20 **[0022]** Resource Oversubscription refers to a situation where the total amount of resources allocated in the cluster exceeds the actual loading capacity of the physical machine, posing a risk of system crash due to competition.

[0023] Memory Overflow Kill (OOM Killer) refers to a self-protection mechanism of the operating system that forcibly terminates high-memory-consuming processes when physical
25 memory is exhausted.

[0024] Pod refers to the smallest deployable and manageable computing unit in Kubernetes, and is an encapsulation collection of one or more containers that share network, storage, and runtime context (e.g., a logical host).

[0025] Container refers to a lightweight, executable software package that includes all
30 content needed to run an application including code, runtime, system tools, libraries, and

settings. In Kubernetes, the container runs inside the Pod.

[0026] Node refers to a worker machine in a Kubernetes cluster, which may be a physical machine or a virtual machine. Each node is managed by a control plane and runs the container runtime (e.g., Docker) and agent components (kubelet) needed to host Pods.

5 **[0027]** Control Plane refers to the "brain" of the cluster, and is responsible for managing global decisions of the cluster (e.g., scheduling) and detecting and responding to cluster events. Main components include API Server, Scheduler, Controller Manager, etc.

[0028] API Server refers to a front end of the Kubernetes control plane, and is the entry point and processing center for all RESTful API requests, and is the only component that
10 interacts directly with the cluster state store (etcd).

[0029] Kubelet refers to an agent running on each Node, and is responsible for ensuring that containers in Pod are in a running state. Kubelet receives Pod definitions issued via the API Server, manages the lifecycle (start, stop) of the container, and reports the status of Nodes and Pods to the control plane.

15 **[0030]** Cgroups (Control Groups) refers to a function of the Linux kernel, which is used to limit, record, and isolate the physical resources (CPU, memory, disk I/O, etc.) used by process groups (e.g., containers).

[0031] Request refers to the resource reservation declared to the scheduler. The scheduler decides whether a Pod can be scheduled onto a Node based on the sum of Requests of all
20 Pods.

[0032] Limit refers to the absolute upper limit of resources allowed to be used by the container during running. Exceeding this limit may result in the container being terminated or throttled.

[0033] Scheduler refers to a component of the control plane responsible for selecting an
25 appropriate Node for newly created Pods that have not yet been assigned a Node, the decisions of which are mainly based on Pod Requests and available resources of each Node.

[0034] VPA (Vertical Pod Autoscaler) refers to an automation tool of the Kubernetes for automatically adjusting Pod Request and Limit values. The standard workflow of the VPA typically requires eviction and restart of Pods to apply new resource configurations.

[0035] HPA (Horizontal Pod Autoscaler) refers to another automation tool of the Kubernetes that responds to load changes by increasing or decreasing the number of Pod replicas of the same application.

[0036] Role in the present solution refers to that the resource adjustment controller in the present solution is essentially a custom Operator that implements specific resource collaboration logic.

[0037] In the present disclosure, a resource management system is provided. One or more embodiments of the present disclosure further relate to a resource management method, a computing device, a computer-readable storage medium, and a computer program product, which are described in detail one by one in the following embodiments.

[0038] In practice, in container orchestration platforms, the problem of temporal and spatial mismatch between resource supply and demand has been identified. The infrastructure layer lacks awareness of critical internal intents of applications, leading to resource allocation that cannot synchronize with actual demand. Further, when a heavy-load task is executed, resource demand changes abruptly in the temporal dimension and exhibits localized characteristics in the spatial dimension, while existing horizontal scaling mechanisms can only respond by adjusting the number of instances, and single-instance resource bottlenecks cannot be eliminated. Vertical scaling mechanism relies on historical monitoring data for decision-making, incurring response latency, and resource adjustment requires eviction and restart of Pods, introducing service interruption. As a result, the contradiction between system stability and resource utilization is difficult to reconcile, and it is forced to choose between statically configuring peak resources to avoid interruptions and setting oversubscription to improve utilization—the former leads to resource idleness, while the latter increases the risk of node-level memory exhaustion.

[0039] For example, in a Kubernetes-based microservice architecture, when an application instance executes a large-scale data sorting task, memory resource demand is significantly increased. However, because the infrastructure cannot predict the task execution intent, resource supply cannot be adjusted in time. Further, the horizontal scaling mechanism cannot complete resource supplementation before the task starts due to response latency, and the vertical scaling mechanism requires Pod eviction, causing task execution interruption. Specifically, when memory demand exceeds the container limit, the node triggers the OOM

Killer mechanism, forcibly terminating the application process, causing service interruption and state loss. In addition, to avoid such interruptions, a higher memory resource is statically configured, but during a task idle period, resource idleness is created, reducing system resource utilization.

5 **[0040]** If the above problem is not solved, the risk of continuous service interruption is faced, and resource utilization remains low for a long time. Frequent business interruptions not only impair user experience but may also cause data consistency. Resource waste increases infrastructure costs, and in multi-task concurrent scenarios, node resource exhaustion may trigger cascading failures. Furthermore, this mismatch is more prominent in
10 dynamic load environment, severely restricting the reliability and scalability of cloud-native applications.

[0041] In the resource management system according to the embodiment, in order to achieve active and lossless resource scaling operation, the resource collaboration component can first calculate resource requirement information corresponding to the heavy-load task
15 upon determining a heavy-load task to be executed by a target application, and send a resource adjustment request to the resource controller according to the resource requirement information. The resource controller can determine a target container associated with the target application in response to the resource adjustment request, and perform a patch update on a resource definition field of the target container according to the resource adjustment
20 request. The agent node directly modifies the control group configuration file of the target container upon detecting the patch update operation, thereby completing resource allocation without rebuilding or restarting the container. Thereafter, the resource controller can send resource adjustment completion information to the target application upon determining that the modified control group configuration file has taken effect, so that the target application
25 executes the heavy-load task. Pod resource definitions are declaratively updated in a patch manner by using the standard Kubernetes API, and the new resource quotas are dynamically injected online into the container control mechanism of the operating system kernel relying on the cooperation of kubelet, thereby achieving seamless, instant resource expansion, effectively avoiding system-wide crashes caused by resource contention, and providing agile, seamless,
30 and business-transparent resource supply, thus significantly improving system stability.

[0042] Referring to Figure 1, Figure 1 is a schematic structural diagram of a resource

management system according to an embodiment of the present disclosure. The resource management system 100 includes a resource collaboration component 110, a resource controller 120, and an agent node 130.

[0043] The resource collaboration component 110 is configured to determine a heavy-load task to be executed by a target application, calculate resource requirement information of the heavy-load task, and send a resource adjustment request to the resource controller based on the resource requirement information.

[0044] The resource controller 120 is configured to determine a target container associated with the target application in response to the resource adjustment request, and perform a patch update to a resource definition field of the target container according to the resource adjustment request.

[0045] The agent node 130 is configured to modify a control group configuration file of the target container upon detecting a patch update operation.

[0046] The resource controller 120 is configured to send resource adjustment completion information to the target application upon determining that the modified control group configuration file has taken effect, for the target application to execute the heavy-load task.

[0047] The resource management system according to this embodiment may be applied to any container orchestration platform, such as K8s, Nomad, or cloud-native computing frameworks and the like. As long as the platform supports modifying cpu.shares and memory.limit without restarting the container through the control group (Cgroups) interface, all the operations can be implemented by the technical solution according to the embodiment.

[0048] Specifically, the resource collaboration component refers to a module in the resource management system that is responsible for identifying a heavy-load task to be executed by the target application, and calculating required resource information based on the task characteristic. This component aims to compensate for the lack of the infrastructure layer in perceiving the internal intentions of the application, and achieve the early identification of resource requirements. The resource controller refers to a core module in the resource management system that receives a resource adjustment request and is responsible for performing a resource adjustment operation. Its main functions include determining the target container, updating a resource definition field of the container, and coordinating with the

agent node to perform modification of the underlying resource configuration. The agent node refers to a module deployed on a computing node, which is responsible for monitoring the configuration change instructions from the resource controller and directly operating the control group (cgroup) configuration file at the operating system level to achieve resource isolation and limitation.

[0049] Correspondingly, the target application refers to an application instance running in a containerized environment that requires dynamic resource adjustment to perform specific tasks. The heavy-load task refers to a task in the target application that requires a large amount of computing, memory, or I/O resources to be executed efficiently, such as big data processing, complex model training, or high-concurrency request handling. The resource requirement information refers to a quantitative description of resources such as processors, memory, storage or network bandwidth that are needed by the heavy-load task during execution. The resource adjustment request refers to a request sent by the resource collaboration component to the resource controller, which includes the target application, the type of heavy-load task, and the calculated resource requirement information, for triggering a resource adjustment process. The target container refers to a container instance associated with the target application whose resource configuration needs to be dynamically adjusted. The resource definition field refers to a configuration item in a container orchestration platform (such as Kubernetes) used to specify container resource requests and limits. The patch update refers to an operation that locally modifies the resource definition field of the target container without recreating or restarting the container, aiming to ensure service continuity during resource adjustment. The control group configuration file refers to a configuration file of the cgroup mechanism in the Linux operating system that is used for resource isolation and limiting. By modifying these files, the system resources available to processes or containers can be directly adjusted. The resource adjustment completion information refers to a notification sent by the resource controller to the target application upon confirming that the resource adjustment has been successfully applied, indicating that the target application can safely start executing its heavy-load task.

[0050] Based on this, a resource management system is provided in the embodiment, aiming to address the problems of temporal and spatial mismatch between resource supply and demand, response delay, service interruption and resource waste in traditional resource management solutions when handling heavy-load tasks. The system achieves real-time,

dynamic and transparent resource adjustment for heavy-load tasks in the application by introducing a resource collaboration component, a resource controller and an agent node.

[0051] Specifically, the resource collaboration component is configured to determine a heavy-load task to be executed by the target application. In an embodiment, this component can identify potential heavy-load tasks by analyzing the deployment manifest file of the target application or preset task type labels. For example, when the deployment manifest contains specific labels such as "big data analysis" or "AI model training", the task is identified as a heavy-load task. In another embodiment, the resource collaboration component can maintain a task feature library and determine the heavy-load task by comparing the task description of the target application with the entries in the feature library. After identifying the heavy-load task, the resource collaboration component is configured to calculate resource requirement information corresponding to the heavy-load task. In an embodiment, the resource collaboration component can obtain the resource requirement information based on the predefined task type and resource requirement mapping table. For example, for a "big data analysis" task, it may be preset to require 8 processor cores and 16GB of memory. In another embodiment, the resource collaboration component may allow an operation and maintenance personnel to manually input or configure the resource requirement of specific task and use it as the resource requirement information. Subsequently, the resource collaboration component is configured to send a resource adjustment request to the resource controller based on the resource requirement information. This request may include the identification of the target application, the type of the heavy-load task, and the calculated resource requirement information. In an embodiment, the resource collaboration component may send the resource adjustment request to the resource controller through a standard API interface. In another embodiment, the resource collaboration component may encapsulate the resource adjustment request as a message and send it to the resource controller through a message queue mechanism.

[0052] Furthermore, the resource controller is configured to determine a target container associated with the target application in response to the resource adjustment request. In an embodiment, the resource controller may query and locate a running container instance associated with the application in metadata of the container orchestration platform based on the target application identifier in the resource adjustment request, and determine it as the target container. In another embodiment, the resource controller may maintain a mapping

table between applications and containers and determine the target container by consulting the table. After the target container is determined, the resource controller is configured to perform a patch update on the resource definition field of the target container according to the resource adjustment request. In an embodiment, the resource controller may directly invoke an API
5 interface provided by the container orchestration platform to modify the resource request value and resource limit value of the target container in a patch manner, and adjust them to the target resource values specified in the resource requirement information. For example, the CPU limit of the container is increased from 2 cores to 8 cores, and the memory limit is increased from 4GB to 16GB. In another embodiment, the resource controller may generate a
10 configuration file containing modifications to the resource definition field, and submit it to the container orchestration platform for updating.

[0053] Furthermore, the agent node is configured to modify a control group configuration file of the target container upon detecting a patch update operation. In an embodiment, the agent node may periodically poll the status of the container orchestration platform, detect
15 changes in the resource definition field of the target container, parse these changes, and directly modify the cgroup configuration file of the corresponding container in the Linux operating system, so that the new resource limits take effect immediately, e.g., modifying the `cpu.cfs_quota_us` and `memory.limit_in_bytes` files in cgroupfs. In another embodiment, the agent node may subscribe to notifications from the resource controller. After completing the
20 patch update, the resource controller can actively notify the agent node to execute the modification of the cgroup configuration file.

[0054] Based on this, the resource controller is configured to send resource adjustment completion information to the target application upon determining that the modified control group configuration file has taken effect. In an embodiment, the resource controller may set a
25 timer for a short time interval, wait for a period after the patch update operation is completed, and then confirm that the cgroup configuration file has been successfully modified and has taken effect through querying the agent node or the operating system status. Once the configuration file is confirmed to have taken effect, the resource adjustment completion information will be sent to the target application. In another embodiment, after completing the
30 modification of the cgroup configuration file, the agent node may send a confirmation message to the resource controller. Upon receiving this message, the resource controller sends the resource adjustment completion information to the target application. Finally, the target

application is configured to execute the heavy load task after receiving the resource adjustment completion information, enabling the target application to efficiently and stably complete its compute-intensive or memory-intensive tasks with sufficient resource guarantees.

5 **[0055]** During the specific implementation, referring to the schematic diagram shown in Figure 2, the resource management system according to the embodiment achieves resource management mainly based on a controller (Operator) cooperating with applications. The resource collaboration component (SDK/Sidecar) runs inside the business container or is deployed as a sidecar. It includes an intent recognition module and a communication interface,
10 which are used to capture resource demand pre-notification inside the application and receive scale-down instruction from the controller. The resource manager (Operator) serves as the control center of the system including: a cluster state monitoring module for obtaining real-time idle physical resources of nodes (not logical allocation); a scheduling event listening module for listening to binding signals from the cluster scheduler; a decision execution
15 module responsible for issuing resource adjustment instructions according to policies. The agent node (Kubelet/In-place Agent) is located on the computing node and is responsible for receiving instructions from the controller and modifying the control group (Cgroups) parameters of the kernel without restarting the containers. When performing resource management, the system mainly follows the principle of "Request = Limit", which can be
20 understood as: deterministic scheduling—by setting request values equal to limit values, the Kubernetes scheduler's occupancy statistics for node resources at any time are consistent with actual physical allocation, completely avoiding the OOM risk triggered when multiple containers reach their peaks simultaneously under the traditional oversubscription (Request < Limit) model; and millisecond feedback loop—because in-place adjustment technology is
25 adopted, the time span from resource "application" to "kernel effect" is shortened to milliseconds, and no business process restart is required, thus ensuring the continuity of long-connection services.

[0056] For example, assuming that in a cloud-native environment, a target application providing "data analysis service" is running, and its current container is configured with
30 baseline resources, e.g., 2 processor cores and 4 GB of memory. In this case, the "data analysis service" needs to execute a large "monthly report generation" heavy-load task, which involves complex aggregation and computation on massive historical data.

[0057] First, the resource collaboration component is configured to identify that the "data analysis service" is about to execute the "monthly report generation" task. For example, the task may be captured by the resource collaboration component through a specific API invoking or a message in a task queue. The resource collaboration component calculates that the "monthly report generation" task requires resource requirement information of 8 processor cores and 16 GB of memory according to a preset policy or a task type mapping table. Subsequently, the resource collaboration component constructs a resource adjustment request containing the identifier of the "data analysis service" and the resource requirements of 8 processor cores and 16 GB of memory, and sends the request to the resource controller. Upon receiving the resource adjustment request, the resource controller determines the running container instance associated with the service in the container orchestration platform as the target container according to the identifier of the "data analysis service". The resource controller then performs a patch update on the resource definition field of the target container according to the resource adjustment request. Specifically, the resource controller invokes the API of the container orchestration platform to update the CPU limit of the target container from 2 cores to 8 cores and the memory limit from 4 GB to 16 GB. This update operation is performed in a non-disruptive manner, avoiding container restart.

[0058] The agent node deployed on the computing node where the target container is located continuously monitors the status changes of the container orchestration platform. When the agent node detects that the resource definition field of the target container has been updated, it will parse these updates. The agent node then modifies the control group configuration file corresponding to the target container on the computing node.

[0059] For example, the agent node updates files such as `/sys/fs/cgroup/cpu/kubepods/.../cpu.cfs_quota_us`` and `/sys/fs/cgroup/memory/kubepods/.../memory.limit_in_bytes``, to immediately adjust the processor and memory resources available to the target container to 8 cores and 16 GB. After the agent node completes the modification of the control group configuration file, the resource controller performs confirmation. For example, the resource controller may wait for a preset short time and then confirm that the resource adjustment has taken effect by querying the agent node or directly checking the cgroup file system. Once it is confirmed that the modified control group configuration file has taken effect, the resource controller may send resource adjustment completion information to the target application of the "data analysis service."

[0060] Upon receiving the resource adjustment completion information, the target application of the "data analysis service" confirms that its runtime environment has sufficient resources, and immediately starts the "monthly report generation" heavy-load task. With the guarantee of 8 processor cores and 16 GB of memory, the task is executed efficiently and stably, avoiding performance bottlenecks or task failures caused by insufficient resources.

[0061] In summary, in the present embodiment, the problem that the infrastructure layer is unaware of critical internal intents of applications is solved through active identification of heavy-load tasks to be executed by the target application and calculation of resource requirements by the resource collaboration component. In the above example, the "monthly report generation" task of the "data analysis service" is identified and its resource requirements are calculated before execution, which is in sharp contrast to traditional solutions that passively adjust based on monitoring data after the task has started. Further, the patch update of the resource definition field of the target container by the resource controller and the real-time modification of the control group configuration file by the agent node achieve business continuity during resource adjustment. In the example, the resources of the target container are dynamically increased without service interruption, avoiding the common service interruption and state loss problems in traditional vertical scaling solutions. With this seamless resource adjustment mechanism, it is ensured that the "data analysis service" smoothly transits from baseline resources to the resources required for the heavy-load task. Thus, with the overall technical solution of the present embodiment, real-time, on-demand dynamic resource adjustment for heavy-load tasks can be achieved through close collaboration among components. This not only effectively solves the temporal and spatial mismatch between resource supply and demand, avoids response latency and service interruption, but also reduces resource idleness and waste caused by static configuration of peak resources through precise resource supply, improving overall resource utilization.

[0062] Further, calculating resource requirement information may lack precision and timeliness, failing to accurately perceive application internal intents, leading to delayed or mismatched resource adjustment. To this end, in the embodiment, the resource collaboration component is further configured to invoke an intent recognition module to calculate processor requirement information and memory requirement information corresponding to the heavy-load task, and use the processor requirement information and the memory requirement information as resource requirement information; and construct a resource adjustment request

according to the resource requirement information, and send the resource adjustment request to the resource controller.

[0063] Here, the intent recognition module is a software or hardware component whose core function is to analyze and understand the intrinsic nature and potential requirement of a task to be executed or being executed by the target application, which can extract key information from application behavior, code logic, configuration information, or preset rules, thereby inferring the expected resource consumption of the application. The module may be integrated inside the target application as part of the application runtime environment, identifying the start signal of a heavy-load task by monitoring internal function calls, data structure access patterns, or usage of specific APIs within the application, and predicting resource requirements according to the preset resource model or machine learning model. Alternatively, the module may be deployed as an independent agent service on the same compute node as the target application, inferring application intent by intercepting and analyzing the application's network communications, file I/O operations, or process state changes, and calculating corresponding resource requirements in combination with a predefined task feature library.

[0064] Correspondingly, the calculating the processor requirement information and memory requirement information corresponding to the heavy-load task aims to quantify the specific requirements of the heavy-load task on the processor and memory to be executed by the target application, thereby providing an accurate basis for subsequent resource adjustment. By directly calculating rather than relying on historical data, temporal and spatial mismatch between resource supply and actual demand can be effectively avoided. When identifying that a heavy-load task (e.g., large-scale data processing, complex model training, or high-concurrency request handling) is about to start, the intent recognition module can estimate in real time the number of processor cores, frequency, and memory capacity required to complete the task based on task type, input data scale, or preset performance indicators, through table lookup, formula calculation, or invocation of a pre-trained prediction model. Alternatively, the calculation process may be combined with a software development kit (SDK) inside the application, where the application developer explicitly declares the expected processor and memory requirements of the task through the SDK interface when writing the heavy-load task code. The intent recognition module is responsible for parsing these declarations and converting them into standardized requirement information.

[0065] In the step of using the processor requirement information and memory requirement information as resource requirement information, the specifically quantified processor and memory requirements are integrated into unified resource requirement information, forming a standardized data structure for subsequent system processing and transmission, which ensures the completeness and consistency of the resource adjustment request. The processor requirement information (e.g., number of CPU cores, CPU utilization percentage) and memory requirement information (e.g., memory size, memory bandwidth) may be encapsulated in a structured data object, such as JSON or YAML format, containing explicit fields to indicate different types of resource requirements. Alternatively, this information may be encoded in a specific message protocol, for example, including a resource type identifier in the message header and corresponding values in the message body, ensuring that the resource controller can correctly parse and understand these requirements.

[0066] In the step of constructing a resource adjustment request according to the resource requirement information, a resource adjustment request conforming to the system communication protocol is generated based on the previously accurately calculated and integrated resource requirement information. The request will serve as an instruction sent to the resource controller, triggering actual resource allocation or adjustment operations. The resource collaboration component may fill a predefined request template according to the received resource requirement information; the template may include the target container identifier, request type (e.g., scale-up, scale-down), and specific resource increments or target values. Alternatively, when constructing the request, a standard API invoking format such as a RESTful API request body may be used, with the resource requirement information included as part of the request parameters, ensuring clear semantics and easy parsing.

[0067] In the step of sending the resource adjustment request to the resource controller which is a key link in achieving resource adjustment, by reliably transmitting the constructed resource adjustment request to the resource controller, the entire resource adjustment process is started. The resource collaboration component may send the request to an API interface exposed by the resource controller via network communication protocols (e.g., HTTP/HTTPS, gRPC). To ensure reliability of the request, a message queue (e.g., Kafka, RabbitMQ) may be used for asynchronous communication, or a retry mechanism may be implemented. In some tightly coupled architectures, the resource collaboration component may directly invoke internal functions or methods provided by the resource controller to pass the request, reducing

network overhead and improving response speed.

[0068] For example, when a target application is preparing to execute a large-scale machine learning model training task, which is identified as a heavy-load task, an SDK integrated inside the target application may be invoked before the task starts. The SDK analyzes the size of the model to be loaded, the scale of the dataset, and the complexity of the training algorithm, and calculates the number of the central processing unit (CPU) cores (e.g., 8 CPU cores) and memory capacity (e.g., 64 GB) required for this training task. The calculated CPU cores and memory capacity constitute the processor requirement information and memory requirement information. Subsequently, the resource collaboration component integrates these processor requirement information and memory requirement information into a structured resource requirement information, e.g., a JSON object containing "cpu_request": "8" and "memory_request": "64Gi". Based on this precise resource requirement information, the resource collaboration component constructs a resource adjustment request. The request may be an HTTP POST request carrying the JSON-formatted resource requirement information in its request body, and specifying the identifier of the target container. Finally, the resource collaboration component sends the resource adjustment request to the resource controller via a network protocol (e.g., HTTP/2). Upon receiving the request, the resource controller initiates the subsequent resource adjustment process to allocate the required CPU and memory resources to the target container.

[0069] In summary, compared with the resource collaboration component only calculating generalized resource requirement information, this solution significantly improves the accuracy and timeliness of resource adjustment, effectively avoids temporal and spatial mismatch between resource supply and demand, and ensures that the target application can obtain sufficient and appropriate resources when executing heavy-load tasks, thereby guaranteeing smooth task execution and system stability.

[0070] Further, without checking available resources of the target node, resource allocation may fail or system instability may occur, this is because the resource demand may exceed the available capacity of the node, causing resource conflict or application interruption. To this end, in the embodiment, the resource controller is further configured to determine a target node corresponding to the target application in response to the resource adjustment request, and query available resource information corresponding to the target node; compare the

resource requirement information with the available resource information, and determine the target container associated with the target application upon determining that the target node meets a resource adjustment condition based on the comparison result; and invoke an orchestration platform interface to perform a patch update on the resource definition field of the target container, where the patch update is used to synchronously modify the resource request value and resource limit value of the target container to a target resource value.

[0071] Specifically, the "determining a target node corresponding to the target application in response to the resource adjustment request" means that after receiving the resource adjustment request, the resource controller needs to identify the specific compute node on which the target application is currently running, which may be achieved by querying the metadata service of the container orchestration platform. For example, in a Kubernetes environment, the resource controller may invoke the Kubernetes API to query the Node information where the Pod (i.e., the target container) associated with the target application is located; alternatively, the resource controller may maintain an internal mapping table between nodes and applications/containers and quickly locate the target node by querying the mapping table. The "querying the available resource information corresponding to the target node" means obtaining the current allocatable computing resources of the target node, which may be achieved in various ways. For example, the resource controller may send a request to an agent program (such as Kubelet in Kubernetes) running on the target node to obtain real-time resource usage and remaining available resources of the node; alternatively, the resource controller may obtain resource statistics of the target node from a centralized cluster resource monitoring system (such as Prometheus).

[0072] Correspondingly, the "comparing the resource requirement information with the available resource information" means comparing the amount of resources required for the heavy-load task of the target application with the amount of resources currently available on the target node. This comparison may be a numerical comparison for each resource, such as CPU and memory, for example, for determining whether the available CPU cores of the target node are greater than or equal to the CPU demand in the resource requirement information, and whether the available memory is greater than or equal to the memory demand in the resource requirement information. The "determining the target container associated with the target application upon determining that the target node meets the resource adjustment condition based on the comparison result" means that only when the target node has sufficient

available resources to meet the resource demand does the resource controller continue to perform subsequent resource adjustment operations. If the comparison result shows insufficient resources, this adjustment may be rejected or delayed. Once the condition is met, the resource controller identifies the specific container instance associated with the target application as the target for the resource patch update.

[0073] Correspondingly, the "invoking an orchestration platform interface to perform a patch update on the resource definition field of the target container" means that the resource controller uses a standardized interface provided by the container orchestration platform to modify the configuration of the target container. For example, in Kubernetes, this typically involves sending a PATCH request to the Kubernetes API Server to update the resource definition (the field `resources`) of a specific container in the target Pod. The "the patch update is used to synchronously modify the resource request value and resource limit value of the target container to a target resource value" means that during resource adjustment, not only the resource request value (requests) of the container is updated, but also its resource limit value (limits) is updated at the same time, and both are set to the same or similar "target resource value." This synchronous modification ensures that the resource demand and maximum available resources of the container remain consistent during scheduling and runtime, avoiding resource contention or waste caused by mismatch between request and limit values.

[0074] Based on this, first, the resource collaboration component determines a heavy-load task to be executed by the target application and calculates corresponding resource requirement information, and then sends a resource adjustment request to the resource controller. After receiving the request, the resource controller does not directly perform a patch update on the target container, but first accurately determines the specific target node on which the target application is currently running in response to the resource adjustment request. Then, the resource controller actively queries the current available resource information of the target node, obtains real-time remaining quantities of various resources such as CPU and memory. Subsequently, the resource controller strictly compares the resource requirement information with the queried available resource information of the target node. Only when the comparison result clearly shows that the target node can meet the resource adjustment condition, i.e., the target node has sufficient resources to bear the increased resource demand of the target application, does the resource controller further

determine the target container associated with the target application. On this basis, the resource controller invokes the interface provided by the container orchestration platform to perform a patch update on the resource definition field of the target container. The key of this patch update is that it synchronously modifies the resource request value and resource limit value of the target container to the pre-calculated target resource value, ensuring consistency of resource configuration. Once the patch update is completed, the agent node detects this operation and correspondingly modifies the control group configuration file of the target container to affect the new resource quota at the operating system level. Finally, after confirming that the modified control group configuration file has successfully taken effect, the resource controller sends resource adjustment completion information to the target application, allowing the target application to execute its heavy-load task under the new, more sufficient resource environment. By introducing a check of available resources of the target node before resource update, this solution effectively avoids system instability and resource conflicts caused by blind resource allocation, ensuring the reliability and success rate of resource adjustment.

[0075] For example, assuming that a cloud-native application runs in a Kubernetes cluster, and its resource collaboration component detects that the application is about to start a large-scale data analysis task, which is identified as a heavy-load task, and calculates that it requires an additional 4 CPU cores and 8 GB of memory as resource requirement information, the resource collaboration component then sends a resource adjustment request to the resource controller. Upon receiving the request, the resource controller first queries the Kubernetes API Server to determine that the Pod (target container) where the data analysis application resides is running on a node named "node-alpha" (target node). Next, the resource controller communicates with the Kubelet agent on "node-alpha" or queries the cluster monitoring system to obtain the available resource information of "node-alpha," which currently has 6 CPU cores and 12 GB of memory. The resource controller compares the requested 4 CPU cores and 8 GB of memory with the available resources of "node-alpha," and determines that the available resources are sufficient to meet the demand. Therefore, the resource controller determines that "node-alpha" meets the resource adjustment condition and confirms the target container. Subsequently, the resource controller constructs a Kubernetes API PATCH request, which sets the ``resources.requests.cpu`` and ``resources.limits.cpu`` of the data analysis container in the target Pod to "4", and sets ``resources.requests.memory`` and

`resources.limits.memory` to "8Gi" and then sends this PATCH request to the Kubernetes API Server. When the Kubelet (agent node) on "node-alpha" detects that the resource fields in the Pod definition have been updated, it immediately modifies the corresponding Linux Cgroups configuration files of that container (for example, updating
5 `/sys/fs/cgroup/cpu/kubepods/.../cpu.cfs\_quota\_us` and
`/sys/fs/cgroup/memory/kubepods/.../memory.limit\_in\_bytes`), so that the new resource quota takes effect immediately at the operating system level. Once the Kubelet confirms that the Cgroups has updated successfully, the resource controller receives a success feedback and sends resource adjustment completion information to the data analysis application. In this
10 case, the data analysis application can safely and efficiently execute its large-scale data analysis task with sufficient resource guarantees.

[0076] In summary, before the resource controller responds to the resource adjustment request and updates the resource definition field, a strict verification step of available resources of the target node is introduced, which effectively solves the problems in traditional
15 solutions, such as resource allocation failure, system instability, and resource conflicts, due to not checking node available resources. By accurately locating the target node, querying its available resources, and performing comparison before resource adjustment, this solution ensures that resource adjustment operations are executed only when the target node has sufficient capacity, thereby avoiding business interruptions and system crashes caused by
20 blind resource allocation. In addition, the resource request value and resource limit value are synchronously modified to the target resource value, further ensuring the uniformity and effectiveness of resource configuration and improving the reliability of resource management and the operational stability of the application, especially for target applications executing heavy-load tasks, which can obtain more stable and predictable resource guarantees.

[0077] Further, when a task binding event occurs, the lack of a mechanism to actively
25 release temporarily occupied resources may lead to resource waste or failure to execute new tasks in time due to insufficient resources, resulting in node-level resource exhaustion or service interruption. To this end, in the embodiment, the resource controller is further configured to determine, according to a priority policy, a resource task to be released that is in
30 a temporary resource adjustment state on the target node upon listening to a task binding event associated with the target node; send a resource release request through a coordination interface to an associated application associated with the resource task to be released; and

release resources corresponding to the resource task to be released on the target node in response to a resource release completion request fed back in response to the resource release request.

[0078] Specifically, the resource controller is the core component of the resource management system, which is responsible for resource allocation, scheduling, and management. Listening to a task binding event means that the resource controller can perceive in real time that a new task (e.g., a container instance or Pod) has been assigned by the scheduler to a specific compute node, but has not yet actually started or run. This listening mechanism is a trigger condition for proactive resource adjustment, ensuring that the system can respond promptly to changes in resource demand. In an embodiment, the resource controller may subscribe to or poll the API server of the container orchestration platform (e.g., Kubernetes) to obtain task binding events related to nodes. For example, by listening to Pod Binding events or Pod creation events in the Kubernetes API, events associated with a specific node is filtered out. In another embodiment, the resource controller may communicate with an agent program (e.g., kubelet) on the node, which actively notifies the resource controller when a task binding event is detected. For example, the agent program may monitor local Pod manifest files or send event notifications to the resource controller via protocols such as gRPC.

[0079] Correspondingly, the priority policy is a set of rules for evaluating and ranking the importance of different tasks, which guides the resource controller to decide which tasks can preferentially release their temporarily occupied resources when resources are tight. Tasks in a temporary resource adjustment state generally refer to those that have temporarily obtained additional resources to cope with short-term high loads, and these resources are not necessary for their long-term stable operation. The purpose of determining the resource task to be released is to efficiently reclaim resources without affecting core services. One priority policy may be based on predefined priority labels of tasks (e.g., critical business, core services, batch tasks, etc.), preferentially selecting tasks with lower priority for resource release. In addition, the system may maintain a task state table marking which tasks are currently in a "temporary resource adjustment state," i.e., they previously obtained additional releasable resources through some mechanism (e.g., intent-driven scaling). Another priority policy may combine the resource utilization of tasks and business importance, for example, preferentially selecting tasks whose current resource utilization is relatively low but are marked as "temporary

scaling," or selecting batch tasks that are insensitive to latency. The system may determine whether a task is in a "temporary resource adjustment state" based on the difference between its resource request value and actual usage value, and combine with priority configured by the business for screening.

5 **[0080]** Correspondingly, the coordination interface is a bridge for communication between the resource controller and the application running inside the task to be released. The purpose of sending a resource release request is to notify the application in a "friendly" and controlled manner, requesting it to actively release part of its temporarily occupied resources, rather than being forcibly terminated or deprived by the system. This coordination mechanism helps
10 avoid business interruptions or data loss caused by forced resource reclamation. One coordination interface may be an HTTP/HTTPS RESTful API. The application registers a callback address with the resource controller at startup, and the resource controller sends an HTTP request containing a resource release instruction to that address when it needs to release resources. Upon receiving the request, the application performs corresponding resource
15 cleanup operations. Another coordination interface may be based on a message queue (e.g., Kafka, RabbitMQ) or inter-process communication (IPC) mechanisms. The resource controller publishes the resource release request as a message to a specific topic or queue, and the application inside the task to be released subscribes to that topic or queue and processes the message. For example, within a container, communication with a sidecar container or the
20 main application process may be achieved via Unix Domain Sockets or shared memory.

[0081] Correspondingly, the resource releasing completion request refers to that a confirmation signal is sent by the application to the resource controller after successfully performing the resource release operation. After receiving this request, the resource controller formally executes the resource release operation corresponding to the task on the node, such
25 as updating resource quotas, adjusting cgroup configuration, etc. This response mechanism ensures the synchronization and reliability of resource release, preventing the resource controller from reclaiming resources before the application is ready, thereby ensuring business continuity. The resource controller may provide a callback API endpoint; after completing resource release, the application sends an HTTP POST request containing a task identifier and
30 release result to that endpoint, indicating successful resource release. Upon receiving this request, the resource controller may update its inside resource management state and trigger subsequent resource reclamation processes. The application may feed back the resource

release completion request by sending a specific message (e.g., via message queue or gRPC call) to the resource controller. The message may include the amount of released resources, task ID, and operation result. After receiving and verifying the message, the resource controller may execute corresponding resource release operation, such as adjusting container resource limits by calling the orchestration platform interface.

[0082] Based on this, by establishing a task binding event-driven resource release process, the problem of missing resource reclamation mechanisms is solved. When the resource controller listens to a task binding event associated with the target node, it indicates that a new task is about to be scheduled to that node and may require additional resources. The resource controller does not take immediate mandatory measures, but first intelligently identifies, according to a preset priority policy, those resource tasks to be released that are in a temporary resource adjustment state on the current node. These tasks typically temporarily obtained additional resources to cope with short-term load peaks, and these resources are not necessary for their long-term stable operation. Once the resource task to be released is determined, the resource controller sends a resource release request through a coordination interface to the associated applications of those tasks. This "courtesy before force" coordination mechanism allows the application to actively and safely perform resource release operations such as memory cleanup and pausing non-core threads upon receiving the request, thereby avoiding business exception or data loss that might be caused by forced system interruption. After completing resource release, the application feeds back a resource release completion request to the resource controller. After receiving and confirming the completion request, the resource controller formally executes the resource release operation for the corresponding task to be released on the target node, such as adjusting container resource quotas or cgroup configuration. This mechanism skillfully combines early warning of resource scheduling (task binding event) with flexible resource release at the application layer. By initiating the resource reclamation process immediately upon the occurrence of a task binding event, the system can free necessary resources for the upcoming new task, effectively alleviating resource tension on the node. In addition, with the guiding of the priority policy, stable operation of core services is ensured, and non-core or temporarily occupied resources are preferentially reclaimed. The proactive cooperative release by applications ensures smooth business transition and avoids the harshness and destructiveness of traditional resource reclamation methods. This cooperative working mode enables the resource management system to achieve

efficient resource reclamation and reuse under the premise of ensuring business continuity, thereby effectively avoiding node-level resource exhaustion or service interruption risks, thus significantly improving overall system resource utilization and stability.

[0083] For example, in a cloud-native environment based on Kubernetes, when the
5 Kubernetes scheduler decides to bind a new Pod (e.g., a high-priority online service Pod) to a certain node, the resource controller may be configured to listen to Pod Binding events from the Kubernetes API server. Once a Pod is bound to the target node, the resource controller immediately triggers the resource reclamation process. In this case, the resource controller queries its internally maintained task state database to identify which Pods on the target node
10 are currently in a "temporary resource adjustment state." For example, there may be a batch processing task Pod that, a few minutes ago, temporarily increased its memory limit from 2 GB to 8 GB to speed up data processing, and its current actual memory usage is far below 8 GB. The resource controller marks the batch processing task Pod as a resource task to be released according to a preset priority policy (e.g., batch tasks have lower priority than online
15 services). Subsequently, the resource controller sends a resource release request through a predefined HTTP RESTful API to the application running inside the batch processing task Pod. The request may contain a JSON payload instructing the application to adjust memory usage back to the baseline value of 2 GB. Upon receiving the request, the batch processing application initiates internal memory cleanup mechanism, such as clearing unnecessary
20 caches and releasing temporary data structures. Once the application confirms that memory usage has been reduced to the target level, it sends an HTTP POST request containing the task ID and a status code of "resource release completed" to the callback API endpoint provided by the resource controller. After receiving the completion request, the resource controller invokes the Kubernetes API to perform a patch update on the resource definition field of the batch
25 processing task Pod, reverting its memory limit and request values back to the baseline value of 2 GB. In this way, the 6 GB of memory resources temporarily occupied by the batch processing task are officially released and made available for the newly bound high-priority online service Pod, ensuring smooth startup and operation of the new task.

[0084] In summary, by immediately initiating the resource reclamation process upon
30 listening to a task binding event associated with the target node, the system can capture the demand signal of new tasks in real time, ensuring timely triggering of resource release and freeing necessary resources for upcoming new tasks. By intelligently selecting releasable

tasks according to priority policies and prioritizing non-core or low-priority tasks, resource allocation efficiency is optimized, avoiding interference with critical services. By communicating directly with the application layer through coordination interfaces, the application performs release operations safely and proactively, avoiding business exceptions or data loss caused by forced interruptions. By formally releasing resources only in response to the resource release completion request fed back by the application, synchronization and reliability of the release operation can be ensured, preventing data inconsistency caused by premature resource reclamation. These measures work in coordination so that the resource management system can achieve efficient resource reclamation and reuse under the premise of ensuring business continuity, thereby effectively avoiding node-level resource exhaustion or service interruption risks, and significantly improving overall system resource utilization and stability.

[0085] On this basis, the associated application lacks specific and efficient resource release mechanism, which may cause delays in the resource release process or trigger business interruption, for example, inability to quickly clean memory or pause non-critical computing threads, thereby affecting the timeliness of resource reclamation and system stability. To this end, in the present embodiment, the associated application is configured to perform a memory release operation and a processor release operation in response to the resource release request, where the memory release operation is used to clean memory buffers or temporary caches, and the processor release operation is used to pause non-core computing threads; feed back a resource release completion request to the resource controller according to operation results corresponding to the memory release operation and the processor release operation respectively.

[0086] Specifically, the associated application refers to an application instance in the resource management system that is associated with the target application and capable of receiving and responding to resource release requests, which serves as the execution subject of resource release and is responsible for proactively adjusting its own resource occupation according to system instructions. The associated application may be enabled to receive and process resource release requests by integrating a specific SDK or library inside the application; alternatively, a proxy or sidecar container is deployed external to the application, which is responsible for communicating with the resource controller and coordinating resource release by the application. The performing memory release operation and processor

release operation in response to the resource release request aims to ensure that the associated application can proactively and timely respond to release instructions from the resource controller and initiate internal resource optimization process. This can be achieved by registering callback functions or event listeners inside the application; upon receiving a resource release request, preset memory and processor release logic is immediately triggered. Alternatively, the resource controller notifies the associated application via IPC mechanisms or API calls, and upon receiving the notification, the main thread or a dedicated resource management thread of the associated application asynchronously starts the resource release task.

[0087] Here, the memory release operation is used to clean memory buffers or temporary caches, with the effect of quickly reclaiming non-core memory resources occupied by the application, reducing memory pressure, while avoiding interruption to core business logic. This operation may manifest as the application actively calling memory management interface (for example, the function `'free()'` in C/C++ or the Linux system invocation `'munmap()'`) to release dynamically allocated memory that is no longer in use, especially buffers for temporary data, logs, or non-critical computation results. Alternatively, the application internally maintains a cleanable cache pool and evicts or clears part of the cached data according to a preset policy (e.g., LRU, LFU, etc.) upon receiving a memory release request. The processor release operation is used to pause non-core computing threads, which aims to reduce the CPU occupancy of the application, free processor time for other high-priority tasks or system resources, while maintaining continuity of a core service. This can be achieved by designing the application to divide computing tasks into core and non-core threads, and upon receiving a processor release request, pausing or reducing the priority of non-core computing threads via thread management APIs (e.g., `'pthread_suspend()'`, `'std::thread::yield()'`, or a custom semaphore/condition variable mechanism). Alternatively, the application internally maintains a scheduler managing all threads; upon receiving the release request, the scheduler adjusts the running strategy of non-core threads, such as reducing their time slice allocation or putting them into a waiting state, until resource pressure is relieved.

[0088] Correspondingly, according to the operation results corresponding to the memory release operation and the processor release operation respectively, a resource release completion request is fed back to the resource controller, which aims to provide the resource controller with the execution status and result of the resource release operation, so that the

resource controller can confirm successful resource release and perform subsequent resource reclamation or scheduling decisions. After completing the memory release and processor release operations, the associated application may encapsulate the respective operation statuses (e.g., success, failure, partial success, etc.) into a structured message and send it to the resource controller via a predefined communication protocol (e.g., HTTP/gRPC API call). Alternatively, the associated application internally maintains a resource release state machine, and triggers the logic to send a resource release completion request to the resource controller when both the memory release and processor release operations reach a "completed" state, where the request may include the amount of released resources or a confirmation of successful release.

[0089] Based on this, through the above mechanism, after the resource controller listens to a task binding event associated with the target node and determines a resource task to be released that is in a temporary resource adjustment state, the resource controller can send a resource release request through a coordination interface to the associated application of the task to be released. Upon receiving the request, the associated application no longer passively waits for system resource reclamation but actively initiates an internal "slimming" procedure. The associated application first performs the memory release operation, cleans memory buffers or temporary caches to quickly release non-core memory resources, avoiding interference with core business. Subsequently, the associated application performs the processor release operation, pauses non-core computing threads to reduce CPU occupancy, freeing computing capacity for other tasks. After completing these operations, the associated application feeds back a resource release completion request to the resource controller according to the result of each operation. This proactive and fine-grained resource release process ensures the timeliness and effectiveness of resource reclamation while minimizing impact on the application's own operation. After receiving the completion request, the resource controller can accurately confirm that resources have been released, and then execute subsequent resource release operation, such as calling an in-place resource adjustment interface to revert the resource request value and resource limit value of the task to be released back to baseline resource values, thereby improving the overall responsiveness and scheduling efficiency of the resource management system in the face of resource pressure.

[0090] For example, assuming that the associated application is a big data processing service, such as an Apache Spark Executor, when the resource controller detects resource

pressure on the target node and sends a resource release request to the Spark Executor, the Spark Executor, as the associated application, immediately responds. In terms of memory release operation, the Spark Executor can proactively clean its internal shuffle caches, local copies of broadcast variables, or release user-defined temporary data structures, these data are typically stored in off-heap memory or the Java heap and are released by calling corresponding memory management APIs. In terms of processor release operation, the Spark Executor can pause non-critical statistical analysis threads, log compression threads, or reduce the priority of minor garbage collection threads to reduce CPU usage. After completing these operations, the Spark Executor sends a JSON-formatted resource release completion request to the resource controller via its built-in communication module, e.g., through a RESTful API interface, which may include information such as `"memory_released": true` and `"cpu_threads_paused": ["thread_id_1", "thread_id_2"]`, to indicate that memory and some processor resources have been successfully released.

[0091] In summary, the associated application can actively and efficiently perform resource reclamation without interruption, solving the problems of resource release delay and service interruption in traditional solutions. By cleaning memory buffers or temporary caches, non-persistent memory resources are directly released without restarting containers or interrupting core business, thereby improving memory reclamation efficiency. In addition, by pausing the non-core computing thread, CPU resources are released while core threads are kept running, maintaining business continuity and optimizing processor utilization. This fine-grained internal resource adjustment mechanism enables the resource controller to confirm resource release status timely and accurately, thereby facilitating more effective subsequent resource scheduling and management, significantly improving resource utilization and stability of the entire resource management system, and effectively avoiding system crashes or performance degradation caused by insufficient resources.

[0092] Furthermore, the callback operation of resource request values and resource limit values after resource release may lack a synchronization mechanism, causing delayed configuration updates or incoherent adjustment operations, thereby affecting the efficiency and responsiveness of resource management. To this end, in the present embodiment, the resource controller is further configured to invoke an in-place resource adjustment interface to execute a resource release operation in response to the resource release completion request, for synchronously reverting the resource request value and resource limit value corresponding

to the resource task to be released to baseline resource values. The agent node is further configured to update the control group configuration file according to a callback result upon detecting a callback operation. The resource controller is further configured to execute a resource adjustment operation for the task binding event.

5 **[0093]** Specifically, the resource controller is the core coordination unit of the entire resource management system, which can trigger the resource reclamation process in response to the resource release completion request fed back by the associated application. In this process, the resource controller invokes the in-place resource adjustment interface to execute the resource release operation, which allows dynamic modification of the resource configuration of the target application without interrupting its operation. The key of this operation is to synchronously revert the resource request value and resource limit value, which were previously increased for the task to be released, back to preset baseline resource values, thereby achieving immediate resource return. The in-place resource adjustment interface may be implemented in ways including but not limited to: interacting with the API server of the container orchestration platform (e.g., Kubernetes) and sending a patch request to update the resource fields in the Pod definition of the target container; or by the resource controller communicating directly with the Kubelet or other container runtime agent deployed on the node to instruct execution of modification of resource configuration.

20 **[0094]** Correspondingly, the agent node, as a component deployed on each compute node, is responsible for monitoring and responding to changes in the resource configuration. When the resource controller performs the above callback operation, the agent node can detect this operation. Upon detecting the callback operation, the agent node updates the control group configuration file of the target container according to the callback result, i.e., the new baseline resource values. The control group configuration file is a key mechanism at the operating system level for limiting and isolating process resources (e.g., CPU, memory). The agent node may detect the callback operation in ways including but not limited to: continuously listening for resource object change events on the container orchestration platform API server, and parsing the new resource values when detecting that the resource definition of the target container has been modified; or the agent node receiving a specific notification message from the resource controller that explicitly indicates the occurrence of the resource callback and the new resource configuration. The way to update the control group configuration file may include: directly modifying corresponding parameters in the Linux cgroups file system, such

as ``cpu.shares``, ``memory.limit_in_bytes`` and the like; or, invoking resource management tools or APIs provided by the operating system to complete the configuration update.

[0095] In addition, the resource controller is further configured to execute resource adjustment operation for the task binding event. A task binding event generally refers to a new task being scheduled to a certain node, or an existing task requiring additional resources. By responding to such events, the resource controller can assess the current resource status of the node in a timely manner and trigger further resource allocation or reclamation processes according to a preset policy (e.g., priority, resource utilization, etc.), ensuring that new tasks can obtain required resources or allocation of existing resources can be optimized. The resource controller may detect task binding events by: subscribing to Pod binding events issued by the scheduler of the container orchestration platform, and evaluating the resource status on that node when a new Pod is scheduled to a certain node; or the resource controller periodically checking resource usage and pending task queues of each node in the cluster, and proactively initiating the resource adjustment operation upon determining that a node is under resource pressure or a high-priority task is waiting.

[0096] Based on this, through close collaboration among the resource controller, the agent node and the associated application, an efficient, real-time resource reclamation and redistribution closed loop is formed. When the associated application completes internal cleanup and sends a resource release completion request to the resource controller, the resource controller immediately responds and synchronously revert the resource request value and limit value of the task to be released back to baseline values by using the in-place resource adjustment interface. This operation avoids container restart and ensures business continuity. Subsequently, the agent node quickly detects this callback operation and updates the control group configuration file at the operating system level in a timely manner, ensuring consistency between physical resource quota and logical configuration. This synchronous configuration update mechanism solves the problem in traditional solutions of delayed or incoherent configuration update after resource release. Furthermore, the resource controller can execute resource adjustment operation for a task binding event, meaning that once resources are effectively released and updated, the system can quickly perceive and allocate them to a new high-priority task, greatly improving resource utilization and system responsiveness. This mechanism ensures a seamless connection from resource release to reallocation, effectively avoiding resource idleness and waste and business interruption due to

insufficient resources.

[0097] For example, in a cloud-native environment, a big data analytics application (as the task to be released) completes its compute-intensive phase and cleans memory buffers and pauses non-core computing threads according to the memory release operation and processor release operation in the above solution. After completing the cleanup, the big data analytics application sends a resource release completion request to the resource controller. Upon receiving the request, the resource controller immediately calls the Patch interface of the Kubernetes API to synchronously modify the fields `requests` and `limits` of the CPU and memory of the Pod corresponding to the big data analytics application from the previously increased values for heavy-load tasks back to their initially defined baseline values. The Kubelet (as the agent node) deployed on the worker node where the Pod resides continuously listens to Pod state changes on the Kubernetes API server. Once detecting that the resource definition of the Pod has been patch-updated, the Kubelet parses the new baseline resource values and immediately modifies the corresponding configuration in the Linux cgroups file system for that Pod, such as updating files `/sys/fs/cgroup/cpu/kubepods.slice/.../cpu.shares`` and `/sys/fs/cgroup/memory/kubepods.slice/.../memory.limit_in_bytes``, thereby officially reclaiming these resource quotas at the operating system level. Almost simultaneously, a new high-priority machine learning training task is scheduled to the same worker node, and a task binding event is triggered. The resource controller detects this task binding event and can quickly allocate resources to the machine learning task or confirm that existing resources are sufficient to meet its needs in combination with the just-released resource information, thereby ensuring timely startup and efficient operation of the high-priority task.

[0098] In summary, the resource controller responds to the resource release completion request and invokes the in-place resource adjustment interface for synchronous callback, and the agent node detects the callback operation and update the control group configuration file, which ensures the immediacy and consistency of resource configuration. This not only avoids inefficiency and incoherence caused by delayed resource configuration but also enables released resources to be rapidly and effectively redistributed through the resource controller executing resource adjustment operations for task binding events, thereby significantly improving resource management efficiency, system responsiveness, and overall resource utilization, while ensuring business continuity and stability.

[0099] In addition, if the associated application fails to feed back the resource release completion request in time, the resource release operation will be delayed, causing resources not to be reclaimed promptly, affecting system efficiency and resource utilization. To this end, in the present embodiment, the resource controller is further configured to execute a resource forced release operation if a resource release completion request fed back by the associated application is not received within a set time interval.

[0100] Specifically, the technical feature "if a resource release completion request fed back by the associated application is not received within a set time interval" means that after sending the resource release request to the associated application, the resource controller starts a timer to monitor the response of the associated application. This time interval may be preconfigured, for example, set to hundreds of milliseconds to several seconds based on network latency, application processing capability, and other factors. In an embodiment, the resource controller may internally maintain a timer module, and start an independent timer for each sent resource release request, and check whether a corresponding feedback has been received when the timer times out. In another embodiment, the resource controller may adopt an asynchronous callback mechanism, record a timestamp when sending the request and registering a timeout callback function; and trigger the callback function to determine that no feedback has been received when the timeout is reached and no feedback has been received.

[0101] Correspondingly, the technical feature "executing a resource forced release operation" means that after detecting that the associated application has timed out without responding, the resource controller no longer waits for proactive cooperation of the application, but directly takes measures to forcibly reclaim or adjust resources. In an embodiment, the resource controller may directly invoke an underlying container orchestration platform API or system interface to forcibly modify the resource configuration of the target container, for example, directly lowering its resource limit value and request value to baseline values or lower, thereby achieving forced release of resources. In another embodiment, the resource controller may send a forced release instruction to the agent node, and the agent node directly modifies the control group configuration file of the target container to achieve forced resource release.

[0102] Based on this, by introducing a resource forced release mechanism, the problem of resource release delay is solved, ensuring timely resource reclamation and improving overall

reliability and efficiency. In the above solution, after listening to a task binding event associated with the target node, the resource controller determines a resource task to be released according to a priority policy and sends a resource release request through a coordination interface to the associated application. This aims to achieve smooth resource release through proactive cooperation at the application level, for example, through memory release operations and processor release operations. However, if the associated application fails to respond in time and feed back a resource release completion request, resources may not be released as expected, affecting timely scheduling and execution of high-priority tasks. On this basis, in the present embodiment, after sending the resource release request, the resource controller is further configured to start a monitoring mechanism. Once the preset time interval has been exceeded without receiving the resource release completion request fed back by the associated application, the resource controller no longer relies on the cooperation of the associated application, but proactively executes a resource forced release operation. This mechanism ensures that even when the associated application is "uncooperative" or "unresponsive," the system can reclaim resources in a timely manner, avoiding long-term resource occupation, thereby ensuring that high-priority tasks can quickly obtain necessary physical resources and start smoothly. This not only improves the robustness of the system to abnormal situations but also significantly enhances the determinism and efficiency of resource scheduling.

[0103] For example, assuming that a high-priority machine learning training task needs to be started on a node, but an existing low-priority data processing task is occupying a large amount of memory, the resource controller sends a resource release request to the data processing application according to a priority policy, expecting it to proactively clean memory buffers or pause non-core computing threads and feed back a resource release completion request. However, if the data processing application becomes unresponsive due to internal faults or excessive load, and fails to feed back the resource release completion request within a set time interval of 500 milliseconds, the resource controller will no longer wait. In this case, the resource controller directly executes a resource forced release operation, for example, by invoking the container orchestration platform interface to forcibly lower the memory limit value and request value of the container where the data processing application resides back to baseline values, or by sending an instruction to the agent node to directly modify the control group configuration file of the container, thereby forcibly reclaiming the occupied memory

resources, ensuring that the high-priority machine learning training task can immediately obtain the necessary physical resources and start smoothly.

[0104] In summary, through the above technical solution, a time threshold is set to monitor the response status of the associated application. When the time interval is exceeded without receiving feedback, this indicates that the associated application may be unable to respond in time or has failed, a subsequent operation is triggered, thereby avoiding indefinite waiting by the system and enhancing the proactivity and controllability of resource management. Executing the resource forced release operation allows the resource controller to directly force release resources upon detecting a timeout, without relying on feedback from the associated application, ensuring timely resource reclamation, preventing resource waste and system blocking, and improving resource utilization efficiency and system stability. Overall, the technical solution of the present embodiment effectively solves the delay problem in resource release by combining time monitoring and forced release mechanisms, strengthening the robustness of the system.

[0105] In specific implementation, the resource management system according to this embodiment is mainly divided into an initialization phase, a resource application phase, a decision evaluation phase, an in-place change phase, an effect notification phase, and a resource retreat phase. In the initialization phase, the business Pod starts, Request = Limit = baseline value (e.g., 4 GB) is set, ensuring the Pod is in a high-stability guarantee level. In the resource application phase, the application senses a high-load task (e.g., model loading) and sends a request ApplyResource (Target_CPU, Target_Mem) to the controller through the SDK. In the decision evaluation phase, the controller queries in real time the idle space of the physical machine on that node (calculation formula: total physical capacity - sum of Requests of all containers). In the in-place change phase, after confirming sufficient space, the controller initiates a Patch Pod operation. The Kubernetes API triggers the in-place adjustment logic, to synchronously raise the Request and Limit of that Pod to the target values. In the effect notification phase, after the node agent (Kubelet) modifies the Cgroups instruction and modification has taken effect, the controller returns a "success" signal to the application, and the application begins to execute a memory-intensive operation. In the resource retreat phase, when perceiving that a high-priority Pod is to be scheduled on the node, the controller forcibly executes the reverse process of the above, shrinking resources in milliseconds.

[0106] It should be noted that when implementing interaction between the application and the resource controller in this embodiment, a sidecar mode may also be used. Specifically, instead of directly integrating an SDK, a lightweight sidecar container is deployed in the same Pod. The sidecar container monitors logs, performance metrics (such as memory watermarks), or specific communication ports of the application, and initiates resource application to the controller (Operator) on behalf of the application. Alternatively, implementation may be based on environment variables or file listening. Specifically, the application does not proactively report, but the controller periodically writes current cluster resource watermarks to a specific shared volume, and the application reads files or listens to environment variable changes to dynamically adjust its internal business logic (e.g., adjusting the size of caches).

[0107] On the other hand, resource adjustment triggering logic may also be implemented through predictive adjustment, introducing an AI prediction module. In this embodiment, the controller does not wait for application requests, but predicts future resource demands based on historical load data, and pre-allocates resources in advance using the In-place Resizing interface. In addition, trigger may be based on specific middleware. Specifically, if the application runs on specific middleware (e.g., Spark, Flink), the controller can directly interface with the resource manager of the middleware to achieve coordinated adjustment of cluster resources and job demands.

[0108] Furthermore, resource management may also be implemented through kernel-level direct control. Specifically, as an alternative to API invoking, the controller may directly modify resource quotas of container processes using eBPF or kernel modules through privileged agents deployed on the node with bypassing the orchestration layer API, thereby achieving flexibility in resource management.

[0109] In the resource management system according to the present embodiment, in order to achieve active and lossless resource scaling operations, the resource collaboration component can first calculate resource requirement information corresponding to the heavy-load task upon determining a heavy-load task to be executed by a target application, and send a resource adjustment request to the resource controller according to the resource requirement information. The resource controller can determine a target container associated with the target application in response to the resource adjustment request, and perform a patch update on a resource definition field of the target container according to the resource

adjustment request. The agent node directly modifies the control group configuration file corresponding to the target container upon detecting the patch update operation, thereby completing resource allocation without rebuilding or restarting the container. Thereafter, the resource controller can send resource adjustment completion information to the target application upon determining that the modified control group configuration file has taken effect, so that the target application executes the heavy-load task. By using the standard Kubernetes API to declaratively update Pod resource definitions in a patch manner, and relying on the cooperation of kubelet, the new resource quotas are dynamically injected online into the container control mechanism of the operating system kernel, thereby achieving seamless, instant resource expansion, effectively avoiding system-wide crashes caused by resource contention, and providing agile, seamless, and business-transparent resource supply, thus significantly improving system stability.

[0110] Corresponding to the above system embodiments, a resource management method is further provided according to an embodiment of the present disclosure. Figure 3 shows a flowchart of a resource management method according to an embodiment of the present disclosure. As shown in Figure 3, the method is applied to a resource management system, which includes a resource collaboration component, a resource controller, and an agent node. The method includes the following steps S302 to S308.

[0111] In step S302, the resource collaboration component determines a heavy-load task to be executed by a target application, calculates resource requirement information corresponding to the heavy-load task, and sends a resource adjustment request to the resource controller according to the resource requirement information.

[0112] In step S304, the resource controller determines a target container associated with the target application in response to the resource adjustment request, and performs a patch update on a resource definition field of the target container according to the resource adjustment request.

[0113] In step S306, upon detecting a patch update operation, the agent node modifies a control group configuration file corresponding to the target container.

[0114] In step S308, if it is determined that the modified control group configuration file has taken effect, the resource controller sends resource adjustment completion information to the target application for the target application to execute the heavy-load task.

[0115] In an embodiment, the resource collaboration component is further configured to invoke an intent recognition module to calculate processor requirement information and memory requirement information corresponding to the heavy-load task, and use the processor requirement information and the memory requirement information as resource requirement information; construct a resource adjustment request according to the resource requirement information, and send the resource adjustment request to the resource controller.

[0116] In an embodiment, the resource controller is further configured to determine a target node corresponding to the target application in response to the resource adjustment request, and query available resource information corresponding to the target node; compare the resource requirement information with the available resource information, and determine the target container associated with the target application if the comparison result determines that the target node meets a resource adjustment condition; invoke an orchestration platform interface to perform a patch update on the resource definition field of the target container. The patch update is used to synchronously modify the resource request value and resource limit value of the target container to a target resource value.

[0117] In an embodiment, the resource controller is further configured to determine, according to a priority policy, a resource task to be released that is in a temporary resource adjustment state on the target node upon listening to a task binding event associated with the target node; send a resource release request through a coordination interface to an associated application associated with the resource task to be released; and release resources corresponding to the resource task to be released on the target node in response to a resource release completion request fed back for the resource release request.

[0118] In an embodiment, the associated application is configured to perform a memory release operation and a processor release operation in response to the resource release request, where the memory release operation is used to clean memory buffers or temporary caches, and the processor release operation is used to pause non-core computing threads; and feed back a resource release completion request to the resource controller according to operation results corresponding to the memory release operation and the processor release operation respectively.

[0119] In an embodiment, the resource controller is further configured to invoke an in-place resource adjustment interface to execute a resource release operation in response to the

resource release completion request, for synchronously reverting the resource request value and resource limit value corresponding to the resource task to be released to baseline resource values. The agent node is further configured to update the control group configuration file according to a callback result upon detecting a callback operation. The resource controller is further configured to execute a resource adjustment operation for the task binding event.

[0120] In an embodiment, the resource controller is further configured to execute a resource forced release operation if a resource release completion request fed back by the associated application is not received within a set time interval.

[0121] In the resource management method according to the present embodiment, in order to achieve active and lossless resource scaling operations, the resource collaboration component can first calculate resource requirement information corresponding to the heavy-load task upon determining a heavy-load task to be executed by a target application, and send a resource adjustment request to the resource controller according to the resource requirement information. The resource controller can determine a target container associated with the target application in response to the resource adjustment request, and perform a patch update on a resource definition field of the target container according to the resource adjustment request. The agent node directly modifies the control group configuration file corresponding to the target container upon detecting the patch update operation, thereby completing resource allocation without rebuilding or restarting the container. Thereafter, the resource controller can send resource adjustment completion information to the target application upon determining that the modified control group configuration file has taken effect, so that the target application executes the heavy-load task. By using the standard Kubernetes API to declaratively update Pod resource definitions in a patch manner, and relying on the cooperation of kubelet, the new resource quotas are dynamically injected online into the container control mechanism of the operating system kernel, thereby achieving seamless, instant resource expansion, effectively avoiding system-wide crashes caused by resource contention, and providing agile, seamless, and business-transparent resource supply, thus significantly improving system stability.

[0122] The above is a schematic solution of the resource management method of the present embodiment. It should be noted that the technical solution of the resource management method belongs to the same concept as the technical solution of the resource

management system described above, and details not described in detail in the technical solution of the resource management method can be referred to the description of the technical solution of the resource management system.

[0123] Figure 4 illustrates a structural block diagram of a computing device 400 according to an embodiment of the present disclosure. The components of the computing device 400 include, but are not limited to, a memory 410 and a processor 420. The processor 420 is connected to the memory 410 via a bus 430, and a database 450 is used to store data.

[0124] The computing device 400 further includes an access device 440, the access device 440 enables the computing device 400 to communicate via one or more networks 460.

Examples of these networks include a public switched telephone network (PSTN), a local area network (LAN), a wide area network (WAN), a personal area network (PAN), or a combination of communication networks such as the Internet. The access device 440 may include one or more of any types of wired or wireless network interfaces (e.g., a network interface controller (NIC), such as an IEEE802.11 Wireless Local Area Network (WLAN) wireless interface, Worldwide Interoperability for Microwave Access (Wi-MAX) interface, Ethernet interface, Universal Serial Bus (USB) interface, cellular network interface, Bluetooth interface, Near Field Communication (NFC).

[0125] In an embodiment of the present disclosure, the above components of the computing device 400, as well as other components not shown in Figure 4, may also be connected to each other, such as via a bus. It should be understood that the block diagram of the structure of the computing device shown in Figure 4 is for exemplary purposes only and is not a limitation on the scope of the present disclosure. Those skilled in the art may add or replace other components as desired.

[0126] The computing device 400 may be any type of stationary or mobile computing device, including a mobile computer or a mobile computing device (e.g., a tablet computer, a personal digital assistant, a laptop computer, a notebook computer, a netbook, etc.), a cell phone (e.g., a smartphone), a wearable computing device (e.g., a smartwatch, smartglasses, etc.), or other types of mobile devices, or a stationary computing device such as desktop computer or a personal computer (PC). The computing device 400 may also be a mobile or stationary server.

[0127] The processor 420 is used to execute computer executable instructions, which, when

executed by the processor, perform the steps of the above-described resource management method.

[0128] A schematic solution of the computing device of the embodiment is described above. It should be noted that the technical solution of the computing device belong to the same concept as the technical solution of the resource management method described above, and the details not described in detail in the technical solution of the computing device may be found in the description of the resource management method described above.

[0129] A computer-readable storage medium is further provided according to an embodiment of the present disclosure, which stores computer-executable instructions, the computer-executable instructions, when executed by a processor, perform the steps of the above-described resource management method.

[0130] A schematic solution of the computer-readable storage medium of the embodiment is described above. It should be noted that the technical solution of the computer-readable storage medium belong to the same concept as the technical solution of the resource management method described above, and the details not described in detail in the technical solution of the storage medium may be referred to the description of the technical solution of the resource management method described above.

[0131] A computer program product is further provided according to an embodiment of the present disclosure, which includes computer programs or instructions, the computer programs or instructions, when executed by a processor, perform the steps of the above-described resource management method.

[0132] A schematic solution of the computer program product of the embodiment is described above. It should be noted that the technical solution of the computer program product belong to the same concept as the technical solution of the resource management method described above, and the details not described in detail in the technical solution of the computer program product may be referred to the description of the technical solution of the resource management method described above.

[0133] Preferred embodiments of the present disclosure are described above. Other embodiments are within the scope of the claims. In some cases, the actions or steps described in the claims may be performed in a different order than in the embodiments and still achieve

the desired results. In addition, the processes depicted in the drawings do not necessarily require the particular order or consecutive order shown to achieve the desired results. In some embodiments, multi-tasking and parallel processing is also possible or may be advantageous.

[0134] The computer instruction includes computer program codes, the computer program code may be in the form of source code, in the form of object code, in the form of an executable file or in some intermediate form or the like. The computer readable medium may include: any entity or device capable of carrying the computer program code, a recording medium, a USB disk, a removable hard disk, a diskette, an optical disk, a computer memory, a Read-Only Memory (ROM), a Random Access Memory (RAM), an electric carrier signal, a telecommunications signal, and software distribution medium, etc. It should be noted that the content of the computer-readable medium may be increased or decreased according to the requirements of the patent practice, for example, in some regions, according to the patent practice, the computer-readable medium does not include carrier signals and telecommunication signals.

[0135] It should be noted that, for each of the foregoing method embodiments, for the sake of simplicity of description, they are all expressed as a series of combinations of actions, but those skilled in the art should know that the embodiments of the present disclosure are not limited by the order of the described actions, because according to the embodiments of the present disclosure, certain steps may be performed in other orders or simultaneously. Secondly, those skilled in the art should also be aware that the embodiments described in the disclosure are preferred embodiments, and the actions and modules involved are not necessarily necessary for the embodiments of the present disclosure.

[0136] In the above-described embodiments, the description of each embodiment has its own focus, and the part not described in detail in a certain embodiment can be referred to the relevant description of other embodiments.

[0137] The above preferred embodiments of the present disclosure are intended only to aid in the description of the present disclosure. The preferred embodiments are not an exhaustive recitation of all details, nor do they limit the invention to the specific embodiments described only. Apparently, many modifications and variations may be made in accordance with the contents of the embodiments of the present disclosure. These embodiments are selected and specifically described in the present disclosure to better explain the principles and practical

applications of the embodiments of the present disclosure so that those skilled in the art to which they belong will be able to well understand and utilize the present disclosure.

CLAIMS

1. A resource management system, comprising:

a resource collaboration component, configured to determine a heavy-load task to be
5 executed by a target application, calculate resource requirement information corresponding to
the heavy-load task, and send a resource adjustment request to a resource controller based on
the resource requirement information;

a resource controller, configured to determine a target container associated with the
target application in response to the resource adjustment request, and perform a patch update
10 on a resource definition field of the target container according to the resource adjustment
request; and

an agent node, configured to modify a control group configuration file of the target
container upon detecting a patch update operation, wherein

the resource controller is further configured to send resource adjustment completion
15 information to the target application if it is determined that the modified control group
configuration file has taken effect, for the target application to execute the heavy-load task.

2. The resource management system according to claim 1, wherein the resource
collaboration component is further configured to:

invoke an intent recognition module to calculate processor requirement information and
20 memory requirement information corresponding to the heavy-load task, and use the processor
requirement information and memory requirement information as the resource requirement
information; and

construct a resource adjustment request based on the resource requirement information
and send the resource adjustment request to the resource controller.

3. The resource management system according to claim 1, wherein the resource
25 controller is further configured to:

determine a target node corresponding to the target application in response to the
resource adjustment request, and query available resource information corresponding to the
target node;

compare the resource requirement information with the available resource information, and determine the target container associated with the target application if it is determined that the target node meets a resource adjustment condition based on a comparison result; and

5 invoke an orchestration platform interface to perform the patch update on the resource definition field of the target container, wherein

the patch update is used to synchronously modify a resource request value and a resource limit value of the target container to a target resource value.

4. The resource management system according to claim 3, wherein the resource controller is further configured to:

10 determine, if a task binding event associated with the target node is detected, a resource task to be released that is in a temporary resource adjustment state on the target node according to a priority strategy;

send, through a coordination interface, a resource release request to an associated application associated with the resource task to be released; and

15 release, in response to a resource release completion request fed back for the resource release request, resources corresponding to the resource task to be released on the target node.

5. The resource management system according to claim 4, wherein the associated application is configured to:

20 perform a memory release operation and a processor release operation in response to the resource release request, wherein the memory release operation is used to clear a memory buffer or a temporary cache, and the processor release operation is used to suspend a non-core computing thread; and

25 send the resource release completion request to the resource controller based on operation results respectively corresponding to the memory release operation and the processor release operation.

6. The resource management system according to claim 5, wherein the resource controller is further configured to invoke an in-place resource adjustment interface to perform a resource release operation in response to the resource release completion request, to synchronously callback the resource request value and resource limit value corresponding to

the resource task to be released to a base resource value;

the agent node is further configured to update, if a callback operation is detected, the control group configuration file based on a callback result; and

the resource controller is further configured to perform a resource adjustment operation
5 for the task binding event.

7. The resource management system according to any one of claims 4 to 6, wherein the resource controller is further configured to perform a forced resource release operation if the resource release completion request fed back by the associated application is not received within a set time interval.

10 8. A resource management method, applied to a resource management system, wherein the resource management system comprises a resource collaboration component, a resource controller and an agent node, the method comprises:

determining, by the resource collaboration component, a heavy-load task to be executed
by a target application, calculating, by the resource collaboration component, resource
15 requirement information corresponding to the heavy-load task, and sending, by the resource
collaboration component, a resource adjustment request to the resource controller according
to the resource requirement information;

determining, by the resource controller, a target container associated with the target
application in response to the resource adjustment request, and performing, by the resource
20 controller, a patch update on a resource definition field of the target container according to the
resource adjustment request;

modifying, by the agent node, a control group configuration file of the target container
upon detecting a patch update operation; and

sending, by the resource controller, resource adjustment completion information to the
25 target application if it is determined that the modified control group configuration file has
taken effect, for the target application to execute the heavy-load task.

9. A computing device, comprising:

a memory, configured to store computer-executable instructions; and

a processor, configured to execute the computer-executable instructions, wherein the

computer executable instructions, when executed by the processor, perform the steps of the method according to claim 8.

10. A computer-readable storage medium, comprising computer executable instructions stored thereon, wherein the computer executable instructions, when executed by a processor,
5 perform the steps of the method according to claim 8.

11. A computer program product, comprising computer programs or instructions, wherein the computer programs or instructions, when executed by a processor, perform the steps of the method according to claim 8.