

MODEL MANAGEMENT SYSTEM AND METHOD

FIELD

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

BACKGROUND

[0002] With the development of computer and internet technologies, large language models (LLMs) have become a core driving force in the field of artificial intelligence. However, providing efficient and low-cost inference services remains a key challenge for their commercial deployment. Cloud-native technologies, especially Kubernetes (K8s), have emerged as the preferred infrastructure for deploying AI inference services due to their powerful container orchestration and automated management capabilities. In addition, high-performance inference frameworks, e.g. vLLM, leverage the PagedAttention algorithm to efficiently manage the massive KV Cache in LLM inference, so as to improve request throughput and resource utilization on a single GPU. In terms of deployment paradigms, serverless architecture has become an ideal choice because it allows users to focus solely on code logic while the platform automatically manages underlying resources. However, in order to integrate serverless with large model inference, it is required to resolve the fundamental conflict between resource utilization and latency. In the conventional art, although cloud-native solutions based on dynamic scaling are common, they face the trade-off between the high TTFT introduced by cold starts and the design of scaling-down thresholds. The K8s Operators or Ray-based integration deployment solution may optimize deployment processes and single-instance throughput but fail to address issues such as long-time process initialization, inability to reuse runtime instances, and lack of intelligent dynamic preemption and fast data transfer mechanisms. Therefore, an effective solution is urgently needed to address the above problems.

SUMMARY

[0003] In view of this, a model management system is provided according to an

embodiment of the present disclosure. One or more embodiments of the present disclosure further relate to a model management method, a computing device, a computer-readable storage medium, and a computer program product, to solve the technical deficiencies in the conventional art.

5 **[0004]** According to a first aspect of the embodiments of the present disclosure, a model management system is provided, which includes a model service gateway and a model controller. The model service gateway is configured to modify, in response to an invocation request for a target business model, an expected state of a model resource object of the target business model to an awakened state upon determining that the target business model is in a
10 dormant state. The model controller is configured to allocate target resources to the target business model upon detecting that the expected state of the model resource object is the awakened state, send an awakening operation to a model instance of the target business model, and modify an actual state of the model resource object to the awakened state according to a resource allocation result and an awakening operation result. The model service gateway is
15 configured to execute a model processing task corresponding to the invocation request by using the target business model upon detecting that the actual state of the model resource object is the awakened state.

[0005] According to a second aspect of the embodiments of the present disclosure, a model management method is provided, which is applied to a model management system. The model
20 management system includes a model service gateway and a model controller and the method includes:

 modifying, by the model service gateway in response to an invocation request for a target business model, an expected state of a model resource object of the target business model to an awakened state upon determining that the target business model is in a dormant state;

25 allocating, by the model controller, target resources to the target business model upon detecting that the expected state of the model resource object is the awakened state, sending, by the model controller, an awakening operation to a model instance of the target business model, and modifying, by the model controller, an actual state of the model resource object to the awakened state according to a resource allocation result and an awakening operation result;
30 and

 executing, by the model service gateway, a model processing task corresponding to the

invocation request using the target business model upon detecting that the actual state of the model resource object is the awakened state.

[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 model 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 model 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 model management method described above.

[0009] The model management system according to this embodiment includes a model service gateway and a model controller. In a case that the model service gateway determines, in response to an invocation request for a target business model, that the target business model is in a dormant state, the model service gateway may modify an expected state of a model resource object of the target business model to an awakened state. Then, upon detecting that the expected state of the model resource object is the awakened state, the model controller allocates target resources to the target business model and sends an awakening operation to a model instance of the target business model, and modifies, according to a resource allocation result and an awakening operation result, the actual state of the model resource object to the awakened state. Finally, in a case that the model service gateway detects that the actual state of the model resource object is the awakened state, the model service gateway executes a model processing task corresponding to the invocation request by using the target business model. Therefore, the state of the model and resource allocation is dynamically managed through the model service gateway and the model controller, and the expected state is modified, resources are allocated and the model is awakened upon detecting a dormant state,

thereby reducing model cold-start latency and improving GPU resource utilization.

BRIEF DESCRIPTION OF THE DRAWINGS

- 5 **[0010]** Figure 1 is a schematic structural diagram of a model management system according to an embodiment of the present disclosure;
- [0011]** Figure 2 is a schematic diagram of a system architecture of a model management system according to an embodiment of the present disclosure;
- [0012]** Figure 3 is a flowchart of a model management method according to an embodiment of the present disclosure; and
- 10 **[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", "the", 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.

30 **[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 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] TTFT (Time to First Token) refers to a latency of the first token, which is a
15 performance metric that measures duration from the time when a user submits a request to the time when the model generates and returns the first output token. A low TTFT is crucial for improving user experience as it directly affects the response speed perceived by the user.

[0020] vLLM refers to a high-performance inference framework for large language models, which is an open-source library that aims to significantly improve the throughput and
20 efficiency of inference of large language models by efficiently managing GPU memory through advanced algorithms such as PagedAttention.

[0021] K8s Operator (Kubernetes Operator) refers to an extension software for Kubernetes that leverages Custom Resource Definitions (CRDs) and controller patterns to automate, package, and manage complex Kubernetes applications, encoding the expertise of human
25 operators into software.

[0022] Serverless refers to a Serverless architecture. In this cloud computing execution model, the cloud service provider is responsible for provisioning, managing, and scaling the underlying infrastructure, while users do not need to manage servers and only pay for the computing resources they actually use.

30 **[0023]** RDMA (Remote Direct Memory Access) refers to a technology that allows directly

exchanging data between the main memories of networked computers, bypassing the CPUs and caches of the operating systems on both sides, thereby achieving high-throughput and low-latency data transmission.

5 [0024] LLM (Large Language Model) refers to deep learning models with a large number of parameters (typically billions or more) and capable of understanding and generating human language text, which is widely used in natural language processing tasks.

[0025] KV Cache refers to Key-Value Cache. In the self-attention mechanism of large language models, this cache is used to store the Key and Value representations of previously generated tokens so that they can be reused during subsequent token generation, thereby
10 improving inference efficiency.

[0026] K8s (Kubernetes) refers to an open-source platform for automating the deployment, scaling, and management of containerized applications (container orchestration).

[0027] HPA (Horizontal Pod Autoscaler) refers to a mechanism in Kubernetes that automatically adjusts the number of Pod replicas based on CPU utilization or custom metrics.

15 [0028] CR (Custom Resource) refers to an extension of the Kubernetes API that allows users to store and manage custom structured data.

[0029] CRD (Custom Resource Definition) refers to an API object that defines a new type of custom resource in a Kubernetes cluster.

20 [0030] TP (Tensor Parallelism) refers to a distributed training/inference technique that splits the tensors (data) of a model across multiple devices for parallel processing.

[0031] PP (Pipeline Parallelism) refers to another distributed technique that assigns different layers/stages of a model to different devices, forming a processing pipeline.

[0032] CUDA IPC (Inter-Process Communication) refers to a high-efficiency transport mechanism that allows different processes on the same machine to directly share GPU
25 memory resources and events, which is used for fast copying (warm-up operation) of model data within the same node (i.e., the same physical machine). By employing CUDA-IPC to achieve efficient data copying, the model loading time is significantly reduced, thereby lowering the time to first token (TTFT).

[0033] In the present disclosure, a model management system is provided. One or more

embodiments of the present disclosure further relate to a model 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.

5 [0034] In practice, in the conventional art, a strategy of "dynamic scaling to zero replicas" is typically adopted to save GPU resources. When a new request arrives, a complete model loading process needs to be performed. This process not only involves the time required to load model weights from storage (such as disk, network storage, or cache) into GPU memory, but also necessitates a series of time-consuming initialization steps, such as creating a new container instance, initializing process context, establishing network communication channel, 10 configuring environment variable, and profiling the PagedAttention KV Cache. These additional overheads, which are not directly part of the model loading itself, collectively account for a significant proportion of the total time, resulting in a prolonged TTFT (Time to First Token) for the user, severely impairing the real-time response experience of the Serverless service.

15 [0035] Furthermore, to avoid excessively frequent scaling, existing solutions typically set a fixed "idle timeout interval" (e.g., 5 minutes). A model is scaled to zero replicas only when the model has received no traffic continuously for that fixed interval. This scaling strategy based on static thresholds is highly inefficient when deploying multiple models at scale. During the timeout waiting period, a large amount of expensive GPU memory remains 20 occupied by idle model instances and cannot be released to other services in need, leading to low utilization of the entire GPU cluster and increased operational costs. Therefore, an effective solution is urgently needed to address the above issues.

[0036] The model management system according to this embodiment includes a model service gateway and a model controller. In a case that the model service gateway determines, 25 in response to an invocation request for a target business model, that the target business model is in a dormant state, the model service gateway may modify an expected state of a model resource object of the target business model to an awakened state. Then, upon detecting that the expected state of the model resource object is the awakened state, the model controller allocates target resources to the target business model and sends an awakening operation to a 30 model instance of the target business model, and modifies, according to a resource allocation result and an awakening operation result, the actual state of the model resource object to the

awakened state. Finally, in a case that the model service gateway detects that the actual state of the model resource object is the awakened state, the model service gateway executes a model processing task corresponding to the invocation request by using the target business model. Therefore, the state of the model and resource allocation is dynamically managed through the model service gateway and the model controller, and the expected state is modified, resources are allocated and the model is awakened upon detecting a dormant state, thereby reducing model cold-start latency and improving GPU resource utilization. In other words, with the model management system according to this embodiment, goals of low latency and high GPU resource utilization can be both achieved in sparse traffic scenarios.

10 **[0037]** Referring to Figure 1, Figure 1 is a schematic structural diagram of a model management system according to an embodiment of the present disclosure. The model management system 100 includes a model service gateway 110 and a model controller 120.

[0038] The model service gateway 110 is configured to modify, in response to an invocation request for a target business model, an expected state of a model resource object of the target business model to an awakened state upon determining that the target business model is in a dormant state. The model controller 120 is configured to allocate target resources to the target business model upon detecting that the expected state of the model resource object is the awakened state, and send an awakening operation to a model instance of the target business model; and modify an actual state of the model resource object to the awakened state according to a resource allocation result and an awakening operation result. The model service gateway 110 is configured to execute a model processing task corresponding to the invocation request by the target business model upon detecting that the actual state of the model resource object is the awakened state.

25 **[0039]** When deploying large language model inference services in a cloud-native environment, due to the sparse nature of request traffic, model instances are scaled down to a dormant state during idle periods. When a new request arrives, it is required to reload the model to a running state. This process involves operations such as process initialization, GPU resource allocation, and communication setup, which increases the Time to First Token. In addition, a scaling-down mechanism based on a fixed time interval cannot dynamically adapt to traffic changes, causing GPU resources to be occupied inefficiently during idle periods and reducing overall resource utilization. The time consumption of process initialization for

awakening a model from a dormant state becomes a key factor affecting system response speed, while the asynchronous execution of resource allocation and model activation further exacerbates the issue of latency.

5 **[0040]** For example, in an online customer service application deployed based on a Kubernetes cluster, the vLLM inference framework is adopted to provide large language model services. During the low-traffic period at night, the frequency of user queries decreases, and the system scales down the model instances to a dormant state. When a new user session request arrives, the model service gateway detects that the target model is in a dormant state and triggers an awakening process. This process includes reallocating GPU resources, 10 initializing the model process and establishing communication channels, leading to an increase in the TTFT at the user end. Furthermore, after the peak daytime traffic ends, because the scaling-down threshold is set to a fixed duration, the model instances remain in a running state during low-traffic periods, causing GPU computing units to remain idle continuously and failing to release resources for scheduling by other tasks. As a result, the system faces an 15 inherent contradiction between resource allocation efficiency and response latency in sparse traffic scenarios.

[0041] If the above problem is not solved, the TTFT of the inference service may remain at a high level, negatively impacting the user interaction experience. In addition, the inefficient use of GPU resources may limit the overall throughput capacity of the cluster and increase 20 infrastructure costs. In scenarios with large traffic fluctuations, the system may not be able to respond promptly to burst requests, reducing service availability. Specifically, the serial processing mechanism of process initialization and resource allocation may prolong the awakening process, while the static scaling-down strategy cannot meet the dynamic traffic demand, resulting in widespread resource idleness.

25 **[0042]** In view of this, the model management system according to this embodiment includes a model service gateway and a model controller. Upon determining, in response to an invocation request for a target business model, that the target business model is in a dormant state, the model service gateway may modify an expected state of a model resource object of the target business model to an awakened state. Then, upon detecting that the expected state 30 of the model resource object is the awakened state, the model controller allocates target resources to the target business model and sends an awakening operation to a model instance

of the target business model, and modify, according to a resource allocation result and an awakening operation result, the actual state of the model resource object to the awakened state. Finally, upon detecting that the actual state of the model resource object is the awakened state, the model service gateway executes a model processing task corresponding to the invocation request by using the target business model. Therefore, the state of the model and resource allocation is dynamically managed through the model service gateway and the model controller, and the expected state is modified, resources are allocated and the model is awakened upon detecting a dormant state, thereby reducing model cold-start latency and improving GPU resource utilization.

[0043] Specifically, the model management system is responsible for full lifecycle management of a business model, including model registration, deployment, awakening, dormancy and resource allocation, which aims to optimize the performance and resource utilization efficiency of model services. The model service gateway, serving as a unified entry for model services, is responsible for receiving external invocation requests and routing or triggering corresponding state transition processes based on the current state of the model. The model service gateway functions to shield the complexity of the underlying models and provide a unified access interface. The model controller is a core component of the model management system, and is responsible for monitoring state changes of model resource objects and executes specific operations such as resource allocation and model instance awakening according to the expected state, so that the model can provide services on demand. The target business model refers to a specific business model that needs to be managed and provides services, e.g. a large language model or an image recognition model.

[0044] Accordingly, the invocation request refers to a request sent by an external system or user to the model service gateway, which requires the target business model to execute a specific processing task. The dormant state refers to a state where the target business model is currently not activated, and its occupied computing resources (such as GPU memory, CPU cores and the like) have been released or are in a minimized occupation state to save system resources. The model resource object refers to a logical entity used in the model management system for abstracting and managing the target business model and its related resources. This object contains the expected state, actual state, and other metadata information of the model. The expected state refers to a target state that the model resource object should achieve under system design or user intent. For example, if there is an invocation request, the expected state

of the model may be set to the awakened state. The awakened state refers to a state where the target business model has been activated, its required computing resources have been allocated, model data has been loaded, and it is ready to receive and process invocation requests. Target resources refer to specific computing resources allocated to awaken the target business model, such as one or more GPU devices, CPU cores, memory space. The model instance refers to a specific process or containerized deployment unit of the target business model during running, which carries the actual computational logic and data of the model. The awakening operation refers to an instruction sent by the model controller to the model instance, which is used to trigger the model instance to load model data, initialize the running environment and the like, transitioning the model from a dormant state to an awakened state. The actual state refers to a state in which the model resource object actually remains at present in the system, which reflects the actual running status and resource occupation of the model instance. The model processing task refers to a specific computational process in which the target business model processes input data according to its internal logic and generates an output result in response to an invocation request.

[0045] Based on this, the model management system according to the embodiment aims to optimize the performance of large models in sparse traffic scenarios. The system achieves fine-grained management of the model lifecycle by introducing a model service gateway and a model controller. First, the model service gateway is configured to respond to invocation requests for a target business model. Upon receiving such a request, the model service gateway detects the current state of the target business model. For example, the model service gateway may obtain its state information by querying a centralized state memory or by communicating with the model instance. If the model service gateway detects that the target business model is in a dormant state, i.e., the target business model is currently not activated and its resources are in a released or minimized occupation state, the model service gateway modifies the expected state of the corresponding model resource object of the target business model to the awakened state. This modification may be achieved by updating a field of the model resource object in a database, or by sending a state update message to a state management service.

[0046] In addition, the model controller is configured to continuously detect the expected state of the model resource object. Once the model controller detects that the expected state of a model resource object has been modified to the awakened state, the model controller

initiates a series of operations to transit the target business model from the dormant state to the awakened state. Specifically, the model controller allocates target resources to the target business model. For example, the model controller may select an available computing node from a preset resource pool and allocate the required GPU memory, CPU cores, or memory to the target business model. After resource allocation is completed, the model controller may send an awakening operation to the model instance of the target business model. This awakening operation may be a Remote Procedure Call (RPC) instructing the model instance to load its pre-trained model data and perform necessary initialization.

[0047] Further, the model controller modifies the actual state of the model resource object to the awakened state according to a resource allocation result and an awakening operation result. For example, if resources are successfully allocated and the model instance successfully responds to the awakening operation, the model controller updates the actual state field in the model resource object to "awakened". This step ensures that the actual state of the model resource object is consistent with the true running state of the model.

[0048] Finally, the model service gateway is configured to execute, upon detecting that the actual state of the model resource object is the awakened state, the model processing task corresponding to the invocation request by using the target business model. For example, the model service gateway may continuously query the actual state of the model resource object, or receive a state update notification from the model controller. Once the model service gateway confirms that the model has been fully woken up and is ready, the model service gateway routes the previously received invocation request to the awakened model instance, which completes the specific model inference or processing task.

[0049] Referring to the schematic diagram shown in Figure 2, the model management system according to this embodiment mainly includes a Gateway (Model Service Gateway), which serves as the overall traffic entry and exit for the system to primarily provide model registration and deletion capabilities to the upper layer, and a model invocation interface for the user side to interact with the Model CRD and Model Operator at the lower layer to complete model registration and deletion and obtain corresponding model instance addresses and other functions. The model management system further includes a Model Operator (Model Controller), which is a model controller and GPU resource allocator implemented based on the K8s Operator, and mainly completes the scheduling and creation of model

instances based on the CR of the model. The Model Controller mainly includes three sub-modules, namely, Scheduler, ModelExecutor, and Dynamic Evict.

[0050] The Scheduler (Resource Manager) is responsible for GPU resource scheduling of model instances, and selects nodes based on deployment information of the model, including TP and PP information and the like of the model. After the decision is made, the current model may be scheduled to one or more specific nodes.

[0051] The ModelExecutor (Instance Manager) is a main execution module with several functions. First, the instance manager handles model creation, after receiving the scheduling result, the instance manager performs the actual creation of Pods and Services in K8s and tracks the status of the corresponding instances and provide services externally if the instance is in the Ready state. Second, the instance manager is responsible for executing model deletion, if the model has no traffic, the instance manager may delete the corresponding Pods and Services. Third, for already created model instances, the instance manager executes swap-in and swap-out operations by invoking the corresponding interfaces of the underlying vLLM instance. Swap-in means swapping the current model instance into the GPU, which may actually occupy GPU cards. Swap-out means releasing the model-related data from the GPU, freeing up the GPU for use by other model instances.

[0052] The Dynamic Evict is a dynamic eviction module, which is primarily responsible for evicting models without traffic. If a node where a model instance is located has no idle GPU cards, this module may attempt to evict models without traffic. Eviction means swapping the model out of the GPU. After successful eviction, idle GPU resources may be used for subsequent resource scheduling logic.

[0053] In addition, the model management system further includes vLLM Instances, which are K8s POD running vLLM on each GPU node. Each vLLM Pod carries all or part of the weights of the model. The current vLLM version has been modified to support fast model loading from the Model Cache Service or model registering with the Model Cache Service, which is crucial for reducing TTFT. The model management system further includes a Model Cache Service, which is a distributed model caching service used to provide fast model registration and loading functionality for vLLM instances. All weights of the relevant model are cached in pinned memory. This service supports both IPC and RDMA communication manners. In order to register a model, the Cache Node where the vLLM instance is located

may be preferentially selected for registration, further improving model loading speed.

5 **[0054]** For example, it is assumed that a large language model is deployed in a cloud-native environment as a target business model to handle text generation request from a user A, to save resources, this target business model is set to a dormant state if there are no requests, and its corresponding model instance is in an inactive state and does not occupy GPU resources. If the user A sends a text generation invocation request to this large language model through a client, the request is first received by the model service gateway. After receiving the request, the model service gateway queries the status of the model resource object corresponding to the target business model. For example, the model service gateway can query a shared configuration center or database and find that the current actual state of the target business model is the dormant state. In view of this, the model service gateway determines that the model needs to be woken up to respond to the request, and modifies the expected state of the model resource object to the awakened state. This operation may be a simple state field update. The model controller continuously monitors the expected states of all model resource objects. Once the model controller detects that the expected state of the target business model has changed to the awakened state, the model controller begins executing the awakening process. Specifically, the model controller first allocates target resources for the target business model. For example, the model controller may determine an available computing node from the cluster and allocate a GPU device along with corresponding memory and CPU resources to the node. After resources are successfully allocated, the model controller sends an awakening operation to the model instance of the target business model. This awakening operation may include instructions requiring the model instance to load its pre-trained model weights and necessary runtime libraries into the allocated GPU memory and complete process initialization.

25 **[0055]** Further, after the model instance successfully loads the model data and completes initialization, the model instance may return an awakening success response to the model controller. Upon receiving feedback indicating successful resource allocation and awakening operation, the model controller may modify the actual state of the model resource object of the target business model to the awakened state. This state update may also be reflected in the shared configuration center or database. In this case, the model service gateway may again detect the actual state of the model resource object of the target business model. If the model service gateway detects that the actual state has changed to the awakened state, the model

30

service gateway confirms that the target business model is fully ready. As a result, the model service gateway routes the text generation invocation request originally sent by the user A to the awakened model instance. Upon receiving the request, the model instance executes the text generation task by using the loaded model data and initialized runtime environment and returns the generated result to the user A. Through the above process, the model management system can dynamically awaken the dormant large language model if the user A sends a request, thereby effectively responding to the user request while ensuring efficient resource utilization.

[0056] In other words, in conventional solutions, e.g. a cloud-native solution based on dynamic scaling, the model must go through complete time-consuming steps like process initialization, communication setup, and model weights loading into GPU memory if the model is cold-started from zero replicas, resulting in high Time to First Token (TTFT). In contrast, in this embodiment, a state-driven dynamic awakening mechanism is achieved by introducing collaborative work between the model service gateway and the model controller. In a case that the model service gateway detects a dormant state and modifies the expected state to the awakened state, the model controller may intervene in advance, allocate target resources, and send an awakening operation. This mechanism allows the resource preparation and initialization process of the model instance to start before the actual processing task, thereby effectively shortening the cold start latency perceived by the user.

[0057] Furthermore, in the conventional art, the conventional scaling-down strategy based on fixed time intervals are often not intelligent enough, causing idle models to continue occupying valuable GPU resources while waiting for the scaling-down interval to end, resulting in low overall GPU utilization. In this embodiment, fine-grained control over the model lifecycle is achieved through the management of the expected state and actual state of the model resource object. The model is only triggered to awaken if an invocation request is received and it is in a dormant state, and is switched back to a dormant state after completing tasks or after a long period without traffic. This on-demand awakening strategy avoids the model from continuously occupying resources in an idle state, thereby significantly improving the overall utilization efficiency of computing resources.

[0058] In summary, in this embodiment, an intelligent and efficient large model management solution is constructed through the collaborative cooperation between the model

service gateway and the model controller, as well as effective management of the desired and actual states of model resource object. With this solution, not only cold start latency in sparse traffic scenarios is effectively reduced but also computing resource utilization is significantly improved, providing important technical support for the commercial deployment of large models.

[0059] Furthermore, the conventional model management system lacks efficient resource allocation and release mechanisms during the model registration process, resulting in low resource utilization and potential startup delay. Therefore, the model controller according to the embodiment includes a resource manager and an instance manager. The model service gateway is further configured to create a model resource object corresponding to the target business model in response to a model registration request submitted for the target business model. The instance manager is configured to send a resource allocation request to the resource manager upon detecting the model resource object. The resource manager is configured to allocate resources to be released for the target business model in response to the resource allocation request. The instance manager is further configured to create a model instance for the target business model after resource allocation, and perform a resource release operation on the resources to be released; and switch the target business model to a dormant state if resource release is completed.

[0060] Specifically, the model controller is a core component responsible for managing the model lifecycle and resource scheduling. The model controller may be an independent microservice or an Operator integrated into an existing platform such as Kubernetes. The model controller may be implemented using various technologies, for example, based on the Kubernetes Operator pattern, by managing model resources through Custom Resource Definitions (CRDs) and a controller, or receiving and processing instructions from the model service gateway by adopting an asynchronous processing mechanism based on a message queue. Accordingly, the resource manager is a component of the model controller, and focuses on the scheduling and allocation of computing resources. The resource manager may select appropriate computing nodes and resource types (e.g., GPU, CPU memory) for allocation based on a preset scheduling policy (e.g., based on available resource capacity, model priority, load balancing, etc.), with implementations including: maintaining a real-time view of the cluster resource pool, obtaining and allocating resources by interacting with the underlying infrastructure (e.g., a Kubernetes scheduler, cloud platform API); or predicting resource

demands and optimizing allocation by adopting an AI-based intelligent scheduling algorithm.

[0061] Correspondingly, the instance manager is a component of the model controller, which is responsible for creation, destruction, and state management of model instances. The instance manager may create or destroy model instances (e.g., Pods) by invoking the API of a container orchestration platform such as Kubernetes based on the resource information allocated by the resource manager, with implementations including: directly interacting with the container engine using a Container Runtime Interface (CRI); or managing the network traffic and lifecycle of model instances through a service mesh. Accordingly, the model service gateway is the entry of the system, which is responsible for receiving external requests and interacting with internal services. The model service gateway may be implemented in the form of an API Gateway, reverse proxy, or load balancer, with implementations including: performing custom development based on open-source components such as Nginx, Envoy; or utilizing an API Gateway service provided by a cloud service provider. Accordingly, a model registration request is a request initiated by a user or the system, which is used to introduce a new business model into the model management system. The request may contain metadata of the model (e.g., model name, version, required resource configuration, deployment parameters, etc.). The model registration request may be submitted via a RESTful API, gRPC interface, or message queue. A model resource object is an abstract representation of the model within the system, containing configuration information, current state, expected state, etc., of the model. The model resource object may be stored in Kubernetes as a Custom Resource, or stored in a configuration center or database.

[0062] Correspondingly, a resource allocation request is a request sent by the instance manager to the resource manager, which is used to obtain the computing resources required to run the model instance. The request may include information such as a model ID, required resource type and quantity. The resource allocation request may be communicated via internal API invocation, message queues, or shared memory. Resources to be released refer to resources reserved or allocated by the resource manager for the target business model but not yet actually occupied by the model instance. This resource allocation strategy can improve resource utilization and avoid resource idleness. The resources to be released may be GPU memory, CPU cores, or memory blocks reserved on the computing node. Accordingly, a model instance is a specific copy of the target business model during running, which is responsible for executing model inference task. The model instance is typically deployed on a

computing node in the form of a container (e.g., a Docker container). Accordingly, a resource release operation refers to an operation of returning computing resources no longer needed by the model instance to the resource pool. The resource release operation may include: deleting the container or Pod corresponding to the model instance; or notifying the underlying
5 infrastructure to reclaim GPU memory, CPU, and memory. A dormant state refers to a state in which the model instance does not occupy computing resources, but its metadata and configuration information are retained. In the dormant state, the model may be quickly woken up, avoiding a complete cold start process. Switching the model to the dormant state may be achieved by unloading the model instance from GPU memory, stopping its running container,
10 but not deleting its configuration, etc.

[0063] Based on this, the model controller is further divided into a resource manager and an instance manager, optimization of model registration and resource management is achieved. If the model service gateway receives a model registration request for a target business model, the model service gateway creates a corresponding model resource object. Upon detecting this
15 model resource object, the instance manager sends a resource allocation request to the resource manager. The resource manager allocates to-be-released resources for the target business model in response to this request. Subsequently, the instance manager creates a model instance using the allocated resources, and simultaneously performs a resource release operation to unload the model data from the GPU memory. After the resource release is
20 completed, the target business model is switched to the dormant state. This mechanism enables the model to be in a "ready but dormant" state after registration, i.e. resources have been reserved but not actually occupied, thereby avoiding the immediate consumption of a large amount of resources upon model registration, while still allowing for rapid awakening when needed. This coordinated task division and pre-allocation strategy enhances the
25 efficiency of model registration, improves resource utilization, and lays the foundation for fast awakening operation.

[0064] For example, a user invokes the model registration interface of the model service gateway, specifies parameters such as model name, TP/PP, to register a large language model to the model service gateway. After receiving the registration request, the model service
30 gateway writes, via the Kubernetes client, a model custom resource (Model CR) named after the model name into Kubernetes, and sets the current state to "initializing", where the Model CR contains information related to model deployment. After the resource manager (Scheduler)

in the model controller detects a newly added Model CR, the resource manager may allocate GPU resources to the model based on deployment information (such as TP/PP) of the model, and schedules the model onto an appropriate computing node. Then, the instance manager (ModelExecutor) in the model controller creates a corresponding model instance (e.g. Pods and Service) in Kubernetes according to the scheduling result. After the vLLM instance successfully loads the model data from NFS, the instance manager performs a swap-out operation to release the model data from the GPU memory and registers the model with the model cache service, where the model is in a dormant state. If a user queries status of the model through the model service gateway, the model service gateway may display "Ready".

10 [0065] In summary, through the above processing, the model management system achieves pre-allocation of resources and dormancy of the model at the model registration stage, effectively solving the problem of low resource utilization after model registration in conventional solutions. This mechanism enables the model to enter a wakeable state quickly after registration, thereby avoiding immediate occupation of large GPU resources and
15 improving overall resource utilization. Further, as resource reservation and data loading have already been completed while the model is dormant, the model can be wakened quickly in response to a subsequent invocation request, thereby significantly reducing the Time-to-First-Token (TTFT), thus enhancing user experience.

[0066] Furthermore, when performing state detection, the gateway does not dynamically
20 adjust the processing flow based on the real-time state of the target business model. As a result, when processing an invocation request, unnecessary state modification steps may be performed or task execution may be delayed, resulting in additional processing latency and resource overhead. In view of this, according to the embodiment, the model service gateway is further configured to detect an actual state in the model resource object of the target
25 business model in response to an invocation request for the target business model, execute a step of modifying the expected state of the model resource object of the target business model to the awakened state if the actual state is the dormant state, and execute a step of executing a model processing task corresponding to the invocation request using the target business model if the actual state is the awakened state.

30 [0067] Specifically, the "in response to an invocation request for the target business model" means that the model service gateway receives a request initiated by an external system or a

user for processing using a specific business model. The request may be an API invocation sent via the HTTP/HTTPS protocol, a message event received through a message queue (such as Kafka, RabbitMQ), or a service request received through a remote procedure call (RPC) framework (such as gRPC). The "detecting an actual state in the model resource object of the target business model" means that the model service gateway obtains the current actual running state of the target business model in the system, which may be achieved by querying a centralized state registry, reading the status field of a corresponding model Custom Resource Definition (CRD) in Kubernetes (K8s), or accessing cached state information maintained internally by the model service gateway. The "executing a step of modifying the expected state of the model resource object of the target business model to the awakened state" means that upon detecting that the model is in a dormant state, the model service gateway issues an instruction to the system indicating that the model is desired to be activated, which may be achieved by updating the field 'spec.desiredState' of the corresponding model CRD in K8s to "awakened", or by invoking an API interface provided by the model controller to trigger an awakening flow for the model. The "executing a step of executing a model processing task corresponding to the invocation request using the target business model" means that if the model is in the awakened state, the model service gateway forwards the received invocation request to an activated and ready model instance of the target business model to perform actual model inference, data processing, or other business logic, which may be achieved by routing the request to the access address of the model instance via an internal load balancer, or by directly invoking an inference service interface provided by the model instance.

[0068] Based on this, a state detection and conditional execution mechanism are introduced, the model service gateway can intelligently process the invocation request according to the current actual state of the target business model. If the model service gateway receives an invocation request for a target business model, the model service gateway first detects the actual state in the model resource object of the target business model. This detection step ensures that the model service gateway can make decisions based on real-time state information, avoiding blindly performing state modification operations. If the actual state is detected to be a dormant state, the model service gateway executes a step of modifying the expected state of the model resource object of the target business model to an awakened state, thereby triggering a model controller (e.g. the model controller described above) to perform

resource allocation and awakening operations for the model instance. This process is performed in coordination with the model controller in the above-described model management system to ensure that the model is correctly awakened from the dormant state. However, if the actual state is detected to already be the awakened state, the model service gateway skips the state modification step and directly executes a step of processing a model processing task corresponding to the invocation request using the target business model. With this design, the model service gateway can dynamically switch between processing paths, avoiding unnecessary modification of the expected state if the model is already in the awakened state, thereby reducing redundant system interactions and potential latency. In this way, on the basis of the above-described model management system, the solution of this embodiment further optimizes the request processing flow and improves the response efficiency of system to changes of the model state.

[0069] For example, it is assumed that a user specifies a name of a model as "Model A" and sends a Chat message through a Chat interface provided by the model service gateway. Upon receiving the request, the model service gateway immediately queries the field 'status.actualState' of a custom resource (CRD) corresponding to "Model A" in K8s to obtain its current actual state. If the query result indicates that 'status.actualState' is "awakened", the model service gateway directly routes the Chat request to an access address of a vLLM instance corresponding to "Model A" for result generation. If the query result indicates that 'status.actualState' is "dormant", the model service gateway modifies the field 'spec.desiredState' of the CRD corresponding to "Model A" in K8s to "awakened", thereby triggering subsequent model awakening processing operation.

[0070] In summary, through the above processing, the model service gateway can dynamically adjust the request processing flow according to the real-time actual state of the target business model, which avoids performing an unnecessary expected state modification step if the model is already in the awakened state, thereby significantly reducing the latency of request processing, particularly providing a faster response speed for frequently invoked hot models. In addition, unnecessary internal system state update operations are reduced, so that the load on the model controller and related components is lowered, improving the resource utilization efficiency and stability of the entire model management system.

[0071] In addition, resource allocation and model loading may be insufficiently efficient,

resulting in increased awakening latency. To address this, according to the embodiment, the resource manager is configured to detect whether a computing node corresponding to the target business model has available resources, determine target resources from the available resources if the computing node has available resources and allocate the target resources to the target business model. The instance manager is configured to send a model awakening request to the model instance of the target business model, where the model awakening request carries resource information corresponding to the target resources. The model instance is configured to load model data corresponding to the target business model in the target resources in response to the model awakening request according to the resource information; and modify the actual state of the model resource object to the awakened state according to a data loading result.

[0072] Specifically, the resource manager is a module or service responsible for managing and allocating computing resources, which functions to determine and allocate appropriate resources on available computing nodes according to the requirement of a model. For example, the resource manager may be an independent microservice that interacts with the model controller via an API interface. Alternatively, the resource manager may also be a component in the model controller that directly manages a resource pool of computing nodes. The resource manager detects whether a computing node corresponding to the target business mode has available resources in various ways. For example, the resource manager may periodically query a resource monitoring agent on the computing node to obtain real-time usage and idle status of resources such as CPU, GPU, and memory; alternatively, the computing node may actively report its resource status to the resource manager. If available resources exist, the resource manager determines a target resource from these available resources and allocates the target resource to the target business model. The target resource may be determined based on a preset scheduling policy, for example, selecting a resource on a computing node with the lowest load, or selecting a resource that best matches the type of the target business model. Allocating the target resource may be implemented by updating a resource status database, marking that the resource has been occupied, and generating a resource allocation credential.

[0073] Correspondingly, the instance manager is a module or service responsible for managing the lifecycle of a model instance, which functions to perform operations such as creating, awakening, and hibernating on a model instance according to the expected state of

the model resource object. For example, the instance manager may be a Kubernetes Operator that manages pods of a model instance by monitoring changes of a custom resource (CR). Alternatively, the instance manager may also be an independent process that operates the model instance by invoking an underlying container runtime interface. The instance manager
5 sends a model awakening request to the model instance of the target business model. This request is an instruction that triggers the model instance to load model data and enter a working state. The sending may be performed via remote procedure call (RPC), a message queue (such as Kafka, RabbitMQ), or by directly invoking a RESTful API provided by the model instance. The model awakening request carries resource information corresponding to
10 the target resource, which is necessary for the model instance to load the model data, such as the ID of the allocated GPU device, GPU memory size, and network configuration. This information ensures that the model instance can accurately identify and use the allocated resources.

[0074] Correspondingly, the model instance is a carrier on which the target business model
15 actually runs on the computing node. For example, the model instance may be a vLLM service process running in a Docker container, or an independent Python process. The model instance responding to the model awakening request means that it receives and parses the awakening instruction from the instance manager. The step of loading the model data corresponding to the target business model into the target resource according to the resource
20 information is a key step for the model to transition from a dormant state to a working state. For example, the model instance loads model weight files from a storage medium (such as disk) into a specified GPU memory according to the GPU ID carried in the request, and initializes a necessary KV Cache structure. The step of modifying the actual state of the model resource object to the awakened state based on the data loading result ensures that the state of
25 the model resource object is consistent with the actual running state of the model instance. For example, after successfully loading the model data, the model instance sends a state update message to the model controller to notify the model controller that the model has been successfully awakened.

[0075] Based on this, the collaborative process of resource allocation and model loading is
30 optimized, the latency of model awakening is effectively reduced. Specifically, before allocating resources, the resource manager first detects whether a computing node corresponding to the target business model has available resources, which ensures the

effectiveness of resource allocation and avoids invalid operations if resources are unavailable, thereby saving unnecessary waiting time. If available resources exist, target resources are determined and allocated therefrom. This process optimizes the efficiency of resource selection by intelligently selecting the most suitable resources, reducing the time required for resource searching and allocation. If the instance manager sends a model awakening request to the model instance of the target business model, the instance manager synchronously carries resource information corresponding to the target resources, so that the model instance directly obtains the detailed configuration required for loading without additional query or negotiation steps, thus simplifying the model loading process. Upon receiving the awakening request, the model instance can immediately load model data in the specified target resources according to the provided resource information. This direct loading reduces configuration and initialization time. Finally, the actual state of the model resource object is modified to the awakened state according to the data loading result, ensuring accurate and timely updating of the model state and reflecting the real situation that the model is ready and available. Through the above tightly coordinated process, the solution of this embodiment significantly improves the efficiency of transitioning the model from a dormant to an awakened state, effectively addressing the awakening delay problem caused by inefficient resource allocation and model loading in conventional solutions.

[0076] For example, if the model service gateway detects that the model resource object of the target business model is in a dormant state and there is an invocation request that requires awakening the model, the resource manager in the model controller first checks whether the computing node where the target business model is located has sufficient GPU resources. For example, the resource manager may query a centralized resource scheduler that maintains the GPU usage of all computing nodes. If available GPU resources exist, the resource manager determines a target GPU based on a preset scheduling policy (e.g., selecting the GPU with the highest idle rate) and allocates the target GPU to the target business model. Subsequently, the instance manager sends a model awakening request to the model instance of the target business model (e.g., a container running a vLLM service). This request includes the ID of the target GPU and other necessary resource configuration information. Upon receiving the request, the model instance immediately utilizes the allocated GPU resources to load the weight data of the target business model from memory into GPU memory and initializes the KV Cache required by vLLM. Once the model data loading is complete, the model instance

reports to the model controller that it has been successfully awakened, and the model controller updates the actual state of the model resource object to the awakened state.

[0077] In summary, through the above processing, in this embodiment, efficient resource allocation and rapid model loading can be achieved for the target business model, significantly reducing the transition latency of the model from the dormant state to the awakened state. This enables the system to respond to user requests more quickly, improving user experience and optimizing the utilization efficiency of computing resources, thus avoiding prolonged idle waiting of resources.

[0078] Based on this, if there is no available resource in the computing node, resource allocation cannot be performed, the awakening of the target business model is fail or delayed, affecting system performance and resource utilization. To address this, according to the embodiment, the model controller further includes a dynamic eviction module. The resource manager is further configured to send a resource eviction request to the dynamic eviction module if the computing node corresponding to the target business model has no available resources. The dynamic eviction module is configured to detect traffic fields in candidate model resource objects corresponding to at least one candidate model in response to the resource eviction request; determine, according to a detection result, a target candidate model with a no-traffic identifier among the at least one candidate model; perform a resource eviction task on the target candidate model, and execute a step of allocating target resources to the target business model upon determining that an amount of available resources in the computing node meets an allocation condition according to a task execution result.

[0079] Specifically, the dynamic eviction module is a functional unit of the model controller, which is mainly responsible for intelligently identifying and releasing resources that are occupied but underutilized (such as no traffic) on the current computing node if system resources are tight, for other models with higher priority or urgent resource demands to use these resources. This module may be an independent software service or process that communicates with the resource manager through internal APIs or a message queue; alternatively, the module may be a submodule or a set of functions in the model controller that directly invokes the interface of the resource manager for resource operation. The module provides an active resource reclamation mechanism for the system, avoiding business interruption or delay caused by insufficient resources. If the resource manager detects that

there is no available resource on the computing node corresponding to the target business model, the resource manager sends a resource eviction request to the dynamic eviction module, which serves as a signal to trigger the dynamic eviction process and indicates that the current system needs to reclaim existing resources in order to meet new resource allocation demands. The resource manager may send the request (containing information of the target computing node and the amount of required resources) by invoking an RPC interface (such as gRPC, RESTful API) provided by the dynamic eviction module; alternatively, the resource manager may encapsulate the resource eviction request as a message and publish the message to a message queue (such as Kafka, RabbitMQ), and the dynamic eviction module subscribes to the message queue for processing.

[0080] Correspondingly, in response to a resource eviction request, the dynamic eviction module detects a traffic field in a candidate model resource object corresponding to at least one candidate model, which is a core step for the dynamic eviction module to identify models that can be evicted. By detecting the traffic field, the system can determine which models are currently in an idle or low-load state, thereby avoiding evicting active models that are providing services. The dynamic eviction module may query a centralized model state database or configuration service that stores the real-time state of all model resource objects, including the traffic field; alternatively, the dynamic eviction module may directly query the model service gateway or the model instance via an API interface for its current traffic statistics information, and evaluate the information as the traffic field. Based on the detection result, a target candidate model with a no-traffic indicator is determined among the at least one candidate model. This step is performed to precisely filter out candidate models that are not currently processing any requests or have extremely low traffic based on the detection result of the traffic field, such models are prioritized for eviction. The dynamic eviction module may set a traffic threshold, and models with a value below this threshold are marked as having a no-traffic indicator; alternatively, the system may maintain a model activity list, dynamically update the traffic field based on the request volume or connection count of the model over a period of time, and determine the no-traffic indicator accordingly. A resource eviction task is executed for the target candidate model, which is the actual operation of releasing resources, where by executing the eviction task, computing resources (such as GPU memory, CPU, memory) occupied by the target candidate model are reclaimed, thereby freeing up space for the target business model. The eviction task may include sending a

hibernation instruction to the model instance of the target candidate model, causing it to release GPU memory and computing resources, and updating the state of its model resource object; alternatively, the eviction task may directly terminating the container or process corresponding to the target candidate model, and the resource scheduling system reclaims the resources occupied by the target candidate model.

[0081] Correspondingly, if it is determined based on the task execution result that the amount of available resources on the computing node meets the allocation condition, the step of allocating target resources for the target business model is performed. This step ensures the effectiveness of the resource eviction, only when the eviction task has successfully freed up sufficient resources and the computing node thus meets the allocation requirement of the target business model, subsequent resource allocation may be performed, thereby avoiding futile allocation attempts. After completing the eviction task, the dynamic eviction module re-queries the amount of available resources on the computing node and compares the amount of available resources with the amount of resources required by the target business model. Alternatively, the execution result of the eviction task may be directly returned to the resource manager, which then determines whether the allocation condition is met based on this result and proceeds with the allocation operation.

[0082] Based on this, existing model management system often fail to effectively allocate resources if computing node resources are insufficient, leading to model awakening failures or delays. In this embodiment, a dynamic eviction module is provided that works in coordination with the resource manager to provide an intelligent resource scheduling mechanism. If the model service gateway determines that the target business model is in a dormant state in response to an invocation request, and instructs the model controller to perform the awakening, the resource manager in the model controller attempts to allocate target resources for the target business model. If the resource manager detects that there are no available resources on the computing node corresponding to the target business model, the resource manager no longer simply reports a resource shortage but instead proactively sends a resource eviction request to the dynamic eviction module. Upon receiving this request, the dynamic eviction module immediately initiates a resource reclamation process. The dynamic eviction module detects the traffic field in the model resource object corresponding to at least one candidate model on the current computing node. By analyzing these traffic fields, the dynamic eviction module can intelligently identify target candidate models with a "no-traffic indicator"

that currently have no traffic or extremely low traffic. These models are typically idle or have very low utilization and are prioritized for eviction. Once the target candidate models are identified, the dynamic eviction module executes resource eviction tasks for these models. This task aims to safely and efficiently release the computing resources currently occupied by the target candidate models, such as GPU memory, CPU, or memory. The execution result of the resource eviction task is crucial. The dynamic eviction module determines, based on this result, whether the amount of available resources on the computing node has met the allocation condition for the target business model. Only when the resource amount meets the allocation condition may the system proceed to execute the step of allocating target resources for the target business model. Through the above mechanism, the solution of this embodiment enables the model management system to proactively and intelligently optimize resources when facing resource constraints, rather than passively waiting. The close collaboration between the resource manager and the dynamic eviction module forms a closed-loop resource scheduling and reclamation system. This not only solves the problem of model awakening failures or delays if there are no available resources on the computing node, but also minimizes the impact on active services by prioritizing the eviction of models with no traffic, thereby significantly improving the overall system resource utilization and response speed. The combination of this dynamic eviction strategy with the model awakening process allows the system to achieve more efficient resource management and lower TTFT (Time To First Token) latency in large model sparse traffic scenarios, thus optimizing the user experience.

[0083] For example, if the model service gateway receives an invocation request for a model A and finds that the model A is in a dormant state, the model service gateway notifies the model controller to perform awakening. The resource manager in the model controller attempts to allocate the required GPU resources for the model A on a computing node X. If the resource manager detects that there are currently insufficient available GPU resources on computing node X, the resource manager sends a resource eviction request to the dynamic eviction module in the model controller, which includes the identifier of the computing node X and the amount of GPU resources required by the model A. Upon receiving the request, the dynamic eviction module queries the model resource objects corresponding to all candidate models currently running on the computing node X (e.g., a model B, a model C, a model D), and detects the traffic fields in these model resource objects, which may record the request QPS (queries per second) or connection count of the model over a past period of time. If the

traffic fields of the model B and the model C show their QPS is 0, while the model D has a high QPS, the dynamic eviction module determines that the model B and the model C are target candidate models with a no-traffic indicator in the traffic field. Subsequently, the dynamic eviction module executes resource eviction tasks for the model B and the model C, which may include invoking the hibernation interface of the model instances of the model B and the model C, instructing the model B and the model C to release the GPU memory and computing resources they occupy and update their states to the dormant state. After the resources of the model B and the model C are successfully released, the dynamic eviction module re-evaluates the amount of available GPU resources on the computing node X. If the available GPU resources on the computing node X have now met the allocation conditions for the model A, the dynamic eviction module notifies the resource manager, which then proceeds to execute the step of allocating target GPU resources for the model A.

[0084] In summary, through the above processing, this embodiment effectively solves the problem that resource allocation cannot be performed if there is no available resource on a computing node, causing the awakening of the target business model to fail or be delayed. Specifically, if the resource manager detects insufficient resources, the resource manager no longer passively waits but actively triggers the dynamic eviction module to reclaim resources. The dynamic eviction module can intelligently identify and preferentially evict idle models with no traffic, thereby avoiding impact on active business. This mechanism ensures that, even under resource constraints, the system can still allocate the required resources for the target business model in a timely manner, significantly reducing the cold-start latency (TTFT) of model awakening and improving the overall resource utilization and response speed of the system. In addition, models with no traffic for eviction are precisely identifies, so that service interruption that may result from blind eviction can be avoided, ensuring system stability and user experience.

[0085] Furthermore, under a sparse traffic scenario, existing model management systems often face the problem of inefficient or untimely resource release if new business models need to be allocated resources but existing resources are insufficient, which may lead to a situation where the resources are not explicitly returned to the resource manager or the actual availability of the resources is not ensured after resource eviction, affecting the function of the resource manager to quickly reallocate resources, ultimately reducing system response speed and resource utilization. To address this, according to the embodiment, the dynamic eviction

module is further configured to invoke a hibernation interface of a candidate model instance of the target candidate model, switch the target candidate model to a dormant state based on the hibernation interface, and return allocated resources of the target candidate model to the resource manager. The resource manager is further configured to execute the step of allocating target resources to the target business model upon determining that the amount of available resources in the computing node meets the allocation condition according to a resource return result.

[0086] Specifically, if resources need to be released, the dynamic eviction module no longer simply terminates model instances, but instead achieves controlled hibernation of the target candidate model by invoking the hibernation interface of the candidate model instance corresponding to the target candidate model. This hibernation interface may be a predefined API endpoint that allows external systems to send instructions to the model instance to instruct the model instance to enter a low-power or inactive state. For example, this interface may be an HTTP/RPC interface, upon receiving the hibernation instruction, the model instance executes a series of resource release operations, such as unloading the model weights and KV Cache from GPU memory, and releasing CPU cores and memory resources. In another implementation, this hibernation interface may also be triggered by updating the status field of a Kubernetes Custom Resource (CR) corresponding to the model instance, with a controller within the model instance monitoring this status change and executing the corresponding hibernation logic.

[0087] Correspondingly, switching the target candidate model to a dormant state based on the hibernation interface means that after receiving the hibernation instruction, the model instance proactively releases the computing resources it occupies, but its process or container may still be alive, albeit in an inactive state. This state transition ensures that resources are released in an orderly and complete manner, avoiding potential resource leaks or state inconsistencies that may arise from forced termination. For example, the model instance can persist its current state to facilitate quick awakening later.

[0088] Correspondingly, the dynamic eviction module returns the allocated resources of the target candidate model to the resource manager. This step is a critical part of resource reclamation, which ensures that the resource manager can promptly perceive the available resources. This can be achieved by the dynamic eviction module sending a resource release

notification message to the resource manager, which contains detailed information about the released resources, such as GPU ID, memory size, number of CPU cores. Alternatively, the dynamic eviction module may directly update the resource inventory maintained by the resource manager, marking the released resources as available.

5 **[0089]** Correspondingly, after receiving the resource return results, the resource manager updates its internal resource availability view based on this information and determines whether the amount of available resources on the computing node meets the allocation condition. The allocation condition here may include the minimum GPU memory, number of CPU cores, memory size, etc. required by the target business model. Only when it is
10 confirmed that the amount of available resources meets the allocation requirement of the target business model does the resource manager proceed to execute the step of allocating target resources for the target business model. This ensures the accuracy and effectiveness of resource allocation, avoiding ineffective allocation if resources are insufficient.

[0090] Based on this, if the model controller needs to allocate resources for the target
15 business model but the computing node has insufficient resources, the dynamic eviction module selects a target candidate model for eviction based on a preset policy (e.g., identifying a target candidate model with a no-traffic indicator based on a traffic field). Unlike directly terminating the model instance, the dynamic eviction module invokes the hibernation interface of the candidate model instance corresponding to the target candidate model,
20 prompting the model instance to proactively and gracefully release the resources it occupies and enter a dormant state. Once the resources are released, the dynamic eviction module explicitly returns the information about these released resources to the resource manager. Upon receiving these resource return results, the resource manager immediately updates its resource pool status and determines whether the amount of available resources on the current
25 computing node has met the allocation condition based on the resource requirement of the target business model. If the allocation condition is met, the resource manager rapidly allocates the target resources to the target business model. This series of coordinated operations makes the resource transfer process from idle models to new models more efficient and controllable.

30 **[0091]** For example, the model service gateway may trigger the resource allocation process in response to an invocation request for a target business model and upon detecting that the

target business model is in a dormant state. If the resource manager on the computing node finds that there are insufficient available resources to awaken or deploy the target business model, the resource manager sends a resource eviction request to the dynamic eviction module. The dynamic eviction module identifies, based on the traffic field, a candidate model on the current computing node that is in an idle state with no traffic, e.g., an LLM instance that has not been invoked for a long time. The dynamic eviction module then sends an instruction "/hibernate" or "/sleep" by invoking the RESTful API interface of the LLM instance. Upon receiving the instruction, the LLM instance executes internal resource cleanup logic, e.g. unloading its model weights and KV Cache from GPU memory to host memory, and releasing the GPU computing units and some CPU cores it occupies. After completing the resource release, the LLM instance sends a confirmation message to the dynamic eviction module. The dynamic eviction module then encapsulates the resource information such as GPU memory block IDs and CPU core IDs into a message and sends the message to the resource manager. Upon receiving this information, the resource manager adds this information to the available resource pool and re-evaluates the resource allocation request for the target business model. If the total amount of available resources now meets the requirement of the target business model, the resource manager selects appropriate target resources from the available resource pool and allocates them to the target business model, thereby allowing the target business model to be awakened or loaded.

[0092] In summary, through the above processing, this embodiment effectively solves the problem of low resource allocation efficiency in conventional solutions due to that resource release is not timely or explicit. The hibernation interface of the model instance and an explicit resource return mechanism are provided, it is ensured that resources can be reclaimed in an orderly and efficient manner and promptly fed back to the resource manager. This enables the resource manager to make allocation decisions quickly based on real-time resource availability, significantly reducing the waiting time for resource allocation and improving the response speed of the system to the deployment or awakening of new business models. In addition, this controlled resource reclamation method also avoids additional overhead and instability that may be caused by forcibly terminating model instances, thereby improving overall resource utilization and system stability.

[0093] Moreover, existing model management systems awake models and load model data when handling model invocation requests, which lacks an efficient strategy for copying

business model data during this process, leading to high data transfer latency if model caches are distributed across different computing nodes, impacting overall performance. To address this, according to this embodiment, if model caches corresponding to the model instance are distributed on the same computing node, copying business model data of the target business model is performed according to an inter-process communication strategy. If model caches corresponding to the model instance are not distributed on the same computing node, copying business model data of the target business model is performed according to a remote memory access strategy.

[0094] Specifically, the inter-process communication strategy refers to a mechanism for data exchange and sharing between different processes within the same computing node. This strategy typically utilizes mechanisms provided by the operating system, such as shared memory, message queues, semaphores, and pipes, to achieve direct data transfer at the memory level, avoiding the overhead of network transmission and the latency of protocol processing. For example, a shared memory mechanism may be employed to directly map the business model data of the target service model into the address spaces of multiple processes, thereby achieving zero-copy data sharing; alternatively, specific hardware acceleration interfaces, e.g. CUDA-IPC, may be utilized to allow different CUDA contexts (typically corresponding to different processes) on the same physical GPU to directly access each other's memory, enabling efficient GPU memory data copying. The remote memory access strategy refers to a mechanism by which one computing node among different computing nodes can directly access the memory of another computing node without the involvement of the target node's CPU. This strategy is typically performed by bypassing the operating system kernel and CPU involvement relying on network hardware that supports RDMA functionality (such as InfiniBand NICs or Ethernet NICs supporting RoCE), significantly reducing data transmission latency and CPU overhead, thus improving data transmission bandwidth. For example, GPU-Direct RDMA technology may be employed to allow a GPU to directly send data to or receive data from the memory of a remote GPU, without going through buffering in the host CPU and system memory, thereby achieving efficient cross-node data transmission in a distributed environment.

[0095] Based on this, the solution described in this embodiment is intended to optimize the efficiency of the model management system when awakening a model and loading business model data, particularly with respect to the distribution of model caches. If the model serving

gateway responds to an invocation request, and the model controller allocates resources and sends an awakening operation to the model instance, the model instance needs to load the business model data of the target business model. In this case, the system dynamically selects the most suitable data copy strategy based on the actual distribution of the model cache
5 corresponding to the model instance. Specifically, if the model cache corresponding to the model instance and the target service model data are located on the same computing node, the system adopts an inter-process communication strategy for data copying. This choice is based on the low-latency and high-bandwidth characteristics of data transmission within the same node, and can ensure that data is transferred quickly and efficiently from the cache to the
10 model instance by avoiding network transmission overhead. Conversely, if the model cache corresponding to the model instance and the target business model data are distributed on different computing nodes, the system switches to a remote memory access strategy. This strategy effectively addresses the challenges of cross-node data transmission by bypassing the CPU overhead of traditional network protocol stacks and utilizing dedicated hardware to
15 achieve high-speed, low-latency data transmission. Through this intelligent strategy selection, the solution of this embodiment ensures that, regardless of how the model cache is distributed, the business model data can be copied in an optimal manner, thereby significantly reducing the data loading time during the model awakening process. This mechanism, closely integrated with the resource allocation, model instance creation, and awakening processes of
20 the above-described model management system, collectively constitutes an efficient on-demand model loading and runtime environment, particularly in a sparse traffic scenario, effectively reduces the TTFT (Time To First Token) latency for model cold starts, enhancing user experience.

[0096] For example, if a model controller needs to awaken a target business model and
25 instruct a model instance of the target business model to load business model data, the model instance first determines whether the required model cache (e.g., model weights and KV Cache) is located in the GPU memory of the current computing node (i.e., the node where the model instance is running) or in the GPU memory of another computing node. For example, if the model cache corresponding to the model instance is distributed on the same computing
30 node, the system may use CUDA-IPC (CUDA Inter-Process Communication) for efficient data copying. In this case, the model data itself (e.g., weights and KV Cache) may be pre-stored in page-locked memory. If the model instance needs to load it, different CUDA

contexts (potentially corresponding to different processes) may directly access and copy GPU memory data on the same physical GPU via the CUDA-IPC mechanism, fully utilizing PCIe bandwidth to achieve extremely low-latency data transfer. On the other hand, if the model cache corresponding to the model instance is not distributed on the same computing node—
5 meaning the model cache is located on a remote computing node—the system may use GPU-Direct RDMA (GPU Direct Remote Direct Memory Access) for data copying. In this scenario, by leveraging network hardware that supports RDMA functionality, e.g. InfiniBand network cards, the GPU may directly transfer data from the GPU memory of the remote computing node to the GPU memory of the local computing node, or vice versa, without
10 requiring multiple copies through the host CPU and system memory. This approach fully utilizes the bandwidth of the InfiniBand network, ensuring efficient, low-latency data transfer even in distributed environments.

[0097] In summary, through the above processing, in this embodiment, the most suitable data copy strategy can be intelligently selected based on the actual distribution of the model
15 cache, thereby significantly optimizing the transmission efficiency of the target business model data during the awakening process. In a case that the model cache corresponding to the model instance is distributed on the same computing node, adopting an inter-process communication strategy leverages the local high-speed data transmission mechanism, avoiding unnecessary network overhead and substantially reducing data copy time. In a case
20 that the model cache corresponding to the model instance is not distributed on the same computing node, adopting a remote memory access strategy effectively addresses the challenges of cross-node data transmission, bypassing the limitations of traditional network protocol stacks to achieve high-speed, low-latency data transmission. This mechanism of dynamically matching the cache location with the copy strategy, combined with the rapid
25 awakening and resource allocation capabilities of the above-described model management system, ensures that the model instance can quickly and efficiently load the required data, thereby significantly reducing the TTFT latency for model cold starts, thus improving the response speed and user experience of model processing tasks. Especially in large model sparse traffic scenarios, this efficient data copy mechanism serves as a critical foundation for
30 the effective operation of functions such as runtime instance reuse and dynamic eviction, ensuring that underlying vLLM instances can be woken up quickly, thereby enhancing the resource utilization and overall performance of the entire model management system.

[0098] The model management system in this embodiment includes a model service gateway and a model controller. In a case that the model service gateway determines, in response to an invocation request for a target business model, that the target business model is in a dormant state, the model service gateway may modify an expected state of a model resource object of the target business model to an awakened state. Then, upon detecting that the expected state of the model resource object is the awakened state, the model controller allocates target resources to the target business model and sends an awakening operation to a model instance of the target business model, and modifies, according to a resource allocation result and an awakening operation result, the actual state of the model resource object is modified to the awakened state. Finally, in a case that the model service gateway detects that the actual state of the model resource object is the awakened state, the model service gateway executes a model processing task corresponding to the invocation request using the target business model. Therefore, the state of the model and resource allocation is dynamically managed through the model service gateway and the model controller, and the expected state is modified, resources are allocated and the model is awakened upon detecting a dormant state, thereby reducing model cold-start latency and improving GPU resource utilization.

[0099] Corresponding to the above system embodiments, a model management method is further provided according to the embodiments of the present disclosure. Figure 3 shows a flowchart of a model management method according to an embodiment of the present disclosure. As shown in Figure 3, the method is applied to a model management system. The model management system includes a model service gateway and a model controller, and the method includes the following steps S302 to S306.

[0100] In step S302, the model service gateway modifies an expected state of a model resource object of the target business model to an awakened state in response to an invocation request for a target business model upon determining that the target business model is in a dormant state.

[0101] In step S304, the model controller allocates target resources to the target business model upon detecting that the expected state of the model resource object is the awakened state, sends an awakening operation to a model instance of the target business model, and modifies an actual state of the model resource object to the awakened state based on a resource allocation result and an awakening operation result.

[0102] In step S306, a model processing task corresponding to the invocation request is executed using the target business model upon detecting that the actual state of the model resource object is the awakened state.

5 [0103] In an embodiment, the model controller includes a resource manager and an instance manager. The model service gateway is further configured to create a model resource object corresponding to the target business model in response to a model registration request submitted for the target business model. The instance manager is configured to send a resource allocation request to the resource manager in a case that the model resource object is detected. The resource manager is configured to allocate resources to be released for the target
10 business model in response to the resource allocation request. The instance manager is further configured to create a model instance for the target business model after resource allocation, and perform a resource release operation on the resources to be released; and switch the target business model to a dormant state in a case that resource release is completed.

15 [0104] In an embodiment, the model service gateway is further configured to detect an actual state in the model resource object of the target business model in response to an invocation request for the target business model; execute a step of modifying the expected state of the model resource object of the target business model to the awakened state in a case that the actual state is the dormant state; and execute a step of executing a model processing task corresponding to the invocation request using the target business model in a case that the
20 actual state is the awakened state.

[0105] In an embodiment, the resource manager is configured to detect whether a computing node corresponding to the target business model has available resources, determine target resources from the available resources if the computing node has available resources and allocate the target resources to the target business model. The instance manager
25 is configured to send a model awakening request to the model instance of the target business model, where the model awakening request carries resource information corresponding to the target resources. The model instance is configured to load model data corresponding to the target business model in the target resources according to the resource information in response to the model awakening request; and modify the actual state of the model resource object to
30 the awakened state according to a data loading result.

[0106] In an embodiment, the model controller further includes a dynamic eviction

module. The resource manager is further configured to send a resource eviction request to the dynamic eviction module if the computing node corresponding to the target business model has no available resources. The dynamic eviction module is configured to detect traffic fields in candidate model resource objects corresponding to at least one candidate model in response to the resource eviction request; determine, according to a detection result, a target candidate model among the at least one candidate model with a no-traffic identifier; perform a resource eviction task on the target candidate model, and execute a step of allocating target resources to the target business model upon determining that an amount of available resources in the computing node meets an allocation condition according to a task execution result.

10 **[0107]** In an embodiment, the dynamic eviction module is further configured to invoke a hibernation interface of a candidate model instance of the target candidate model, switch the target candidate model to a dormant state based on the hibernation interface, and return allocated resources of the target candidate model to the resource manager. The resource manager is further configured to execute the step of allocating target resources to the target business model upon determining that the amount of available resources in the computing node meets the allocation condition according to a resource return result.

[0108] In an embodiment, in a case that model caches corresponding to the model instance are distributed on the same computing node, copying business model data of the target business model is performed according to an inter-process communication strategy. In a case that model caches corresponding to the model instance are not distributed on the same computing node, copying business model data of the target business model is performed according to a remote memory access strategy.

[0109] The model management system in this embodiment includes a model service gateway and a model controller. In a case that the model service gateway determines, in response to an invocation request for a target business model, that the target business model is in a dormant state, the model service gateway may modify an expected state of a model resource object of the target business model to an awakened state. Then, upon detecting that the expected state of the model resource object is the awakened state, the model controller allocates target resources to the target business model and sends an awakening operation to a model instance of the target business model, and modifies, according to a resource allocation result and an awakening operation result, the actual state of the model resource object to the

awakened state. Finally, in a case that the model service gateway detects that the actual state of the model resource object is the awakened state, the model service gateway executes a model processing task corresponding to the invocation request using the target business model. Therefore, the state of the model and resource allocation is dynamically managed through the model service gateway and the model controller, and the expected state is modified, resources are allocated and the model is awakened upon detecting a dormant state, thereby reducing model cold-start latency and improving GPU resource utilization.

[0110] The above is a schematic solution of a model management method in this embodiment. It should be noted that the technical solution of the model management method and the technical solution of the above model management system belong to the same concept. For details not described in detail in the technical solution of the model management method, reference may be made to the description of the technical solution of the above model management system.

[0111] 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.

[0112] 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 640 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).

[0113] 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.

[0114] 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.

[0115] The processor 420 is used to execute computer executable instructions, which, when executed by the processor, perform the steps of the above-described model management method.

[0116] 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 model 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 technical solution of the model management method described above.

[0117] 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 model management method.

[0118] 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 model 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 model management method described above.

[0119] 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 model

management method.

[0120] 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 model 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 model management method described above.

[0121] 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.

[0122] 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.

[0123] 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.

[0124] 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.

5 **[0125]** 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
10 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 model management system, comprising:

a model service gateway, configured to modify, in response to an invocation request for a
5 target business model an expected state of a model resource object of the target business
model to an awakened state upon determining that the target business model is in a dormant
state; and

a model controller, configured to allocate target resources to the target business model
upon detecting that the expected state of the model resource object is the awakened state, send
10 an awakening operation to a model instance of the target business model, and modify an
actual state of the model resource object to the awakened state according to a resource
allocation result and an awakening operation result, wherein

the model service gateway is further configured to execute a model processing task
corresponding to the invocation request by using the target business model upon detecting that
15 the actual state of the model resource object is the awakened state.

2. The model management system according to claim 1, wherein the model controller
comprises a resource manager and an instance manager, wherein

the model service gateway is further configured to create a model resource object
corresponding to the target business model in response to a model registration request
20 submitted for the target business model;

the instance manager is configured to send a resource allocation request to the resource
manager upon detecting the model resource object;

the resource manager is configured to allocate resources to be released for the target
business model in response to the resource allocation request; and

25 the instance manager is further configured to create a model instance for the target
business model after resource allocation, and perform a resource release operation on the
resources to be released; and switch the target business model to a dormant state in a case that
resource release is completed.

3. The model management system according to claim 1, wherein the model service

gateway is further configured to detect an actual state in the model resource object of the target business model in response to an invocation request for the target business model; execute, if the actual state is the dormant state, the step of modifying the expected state of the model resource object of the target business model to the awakened state; and execute, if the
5 actual state is the awakened state, the step of executing a model processing task corresponding to the invocation request using the target business model.

4. The model management system according to claim 2, wherein the resource manager is configured to detect whether a computing node corresponding to the target business model has available resources, determine, if the computing node has available resources, target
10 resources from the available resources and allocate the target resources to the target business model;

the instance manager is configured to send a model awakening request to the model instance of the target business model, wherein the model awakening request carries resource information corresponding to the target resources; and

15 the model instance is configured to load, in response to the model awakening request, model data corresponding to the target business model in the target resources according to the resource information; and modify the actual state of the model resource object to the awakened state according to a data loading result.

5. The model management system according to claim 4, wherein the model controller
20 further comprises a dynamic eviction module;

the resource manager is further configured to send a resource eviction request to the dynamic eviction module if the computing node corresponding to the target business model has no available resources; and

the dynamic eviction module is configured to detect, in response to the resource eviction
25 request, traffic fields in candidate model resource objects corresponding to at least one candidate model; determine, according to a detection result, a target candidate model among the at least one candidate model with a no-traffic identifier; perform a resource eviction task on the target candidate model; and execute the step of allocating target resources to the target business model upon determining according to a task execution result that an amount of

available resources in the computing node meets an allocation condition.

6. The model management system according to claim 5, wherein the dynamic eviction module is further configured to invoke a hibernation interface of a candidate model instance of the target candidate model, switch the target candidate model to a dormant state based on the hibernation interface, and return allocated resources of the target candidate model to the resource manager; and

the resource manager is further configured to execute the step of allocating target resources to the target business model upon determining according to a resource return result that the amount of available resources in the computing node meets the allocation condition.

7. The model management system according to claim 1, wherein in a case that model caches corresponding to the model instance are distributed on the same computing node, copying business model data of the target business model is performed according to an inter-process communication strategy; and

in a case that model caches corresponding to the model instance are not distributed on the same computing node, copying business model data of the target business model is performed according to a remote memory access strategy.

8. A model management method, applied to a model management system, wherein the model management system comprises a model service gateway and a model controller, the method comprises:

modifying, by the model service gateway in response to an invocation request for a target business model, an expected state of a model resource object of the target business model to an awakened state upon determining that the target business model is in a dormant state;

allocating, by the model controller, target resources to the target business model upon detecting that the expected state of the model resource object is the awakened state, sending, by the model controller, an awakening operation to a model instance of the target business model, and modifying, by the model controller, an actual state of the model resource object to the awakened state according to a resource allocation result and an awakening operation result; and

executing, by the model service gateway, a model processing task corresponding to the

invocation request using the target business model upon detecting that the actual state of the model resource object is the awakened state.

9. A computing device, comprising:

a memory, configured to store computer executable instructions; and

5 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,

10 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.