TASK SCHEDULING METHOD AND SYSTEM AND STORAGE MEDIUM

TECHNICAL FIELD

[0001] This specification relates to the field of resource allocation, and in particular, to a task scheduling method and system and a storage medium.

BACKGROUND

[0002] With development of Internet technologies and continuous expansion of the data scale, requirements for task processing are also constantly increasing. To improve data processing efficiency, a manner of allocating a computing task to a plurality of physical machines for parallel processing emerges. When there is an upstream-downstream dependency relationship between a plurality of related tasks, how to properly allocate computational resources to improve overall task execution efficiency is particularly important. In a related technology, tasks are randomly allocated to corresponding physical machines. As a result, an association relationship between dependent tasks is ignored, and unnecessary time is often consumed.

[0003] Therefore, a new task scheduling method needs to be provided to allocate tasks based on a dependency relationship between upstream and downstream tasks, thereby shortening overall time consumed by the tasks.

[0004] Content of the background part is merely information learned of by the applicant, and neither means that the information has entered a public domain before an application date of this disclosure, nor means that the information can become a conventional technology of this disclosure.

SUMMARY

[0005] According to a first aspect, this specification provides a task scheduling method, including: obtaining a target directed graph, where the target directed graph is a directed acyclic graph formed based on a dependency relationship between N tasks, and N is an integer greater than 1; sequentially allocating the N tasks to M machines based on the dependency relationship in an order of executable moments from early to late, to generate a scheduling table, where M is an integer greater than 1, and in each time of task allocation, a to-be-allocated task is allocated to a machine that is expected to be capable of completing the task at the earliest time, where the

executable moment is determined based on a moment at which all upstream tasks on which each task depends are completed and expected idle moments of the M machines; and outputting the scheduling table.

[0006] According to a second aspect, this specification further provides a task scheduling system, including: at least one storage medium, storing at least one instruction set used to perform task scheduling; and at least one processor, communicatively connected to the at least one storage medium. When running, the at least one processor reads the at least one instruction set to perform the following operations based on an instruction of the at least one instruction set: obtaining a target directed graph, where the target directed graph is a directed acyclic graph formed based on a dependency relationship between N tasks, and N is an integer greater than 1; sequentially allocating the N tasks to M machines based on the dependency relationship in an order of executable moments from early to late, to generate a scheduling table, where M is an integer greater than 1, and in each time of task allocation, a to-be-allocated task is allocated to a machine that is expected to be capable of completing the task at the earliest time, where the executable moment is determined based on a moment at which all upstream tasks on which each task depends are completed and expected idle moments of the M machines; and outputting the scheduling table.

[0007] According to a third aspect, this specification further provides a computer-readable nonvolatile storage medium. The computer-readable nonvolatile storage medium stores at least one instruction set, and the at least one instruction set is executed by at least one processor to implement the task scheduling method according to the first aspect.

[0008] It can be learned from the above-mentioned technical solutions that, according to the task scheduling method and system and the storage medium provided in this specification, when a plurality of tasks in a target directed graph are scheduled, the plurality of tasks are sequentially allocated based on a dependency relationship between the tasks in an order of executable moments from early to late. In each time of task allocation, a to-be-allocated task is allocated to a machine that is expected to be capable of completing the task at the earliest time. Therefore, according to the method, system, and storage medium provided in this specification, allocation logic can be optimized by using an associated relationship between dependent tasks, thereby shortening overall time for completing a plurality of tasks.

[0009] Other functions of the task scheduling method and system and the storage medium provided in this specification are listed in the following descriptions. Based on the descriptions, content described by using the following numbers and examples is clear to a person of ordinary skill in the art. The creative aspects of the task scheduling method and system and the storage medium provided in this specification can be fully explained by practice or by using the method, apparatus, and combination described in the following detailed example.

BRIEF DESCRIPTION OF DRAWINGS

[0010] To describe the technical solutions in the embodiments of this specification more clearly, the following briefly describes the accompanying drawings needed for describing the embodiments. Clearly, the accompanying drawings in the following description show merely some embodiments of this specification, and a person of ordinary skill in the art can still derive other drawings from these accompanying drawings without creative efforts.

[0011] FIG. 1 is a schematic diagram of an application scenario of a task scheduling system according to some embodiments of this specification;

[0012] FIG. 2 is a schematic diagram of a hardware structure of a computing device according to some embodiments of this specification;

[0013] FIG. 3 is a schematic flowchart of a task scheduling method according to some embodiments of this specification;

[0014] FIG. 4 is a schematic flowchart of each time of task allocation according to some embodiments of this specification;

[0015] FIG. 5 is an example diagram of determining a target task according to some embodiments of this specification;

[0016] FIG. 6 is an example diagram of determining an expected completion moment according to some embodiments of this specification; and

[0017] FIG. 7 is an example diagram of task scheduling according to some embodiments of this specification.

DESCRIPTION OF EMBODIMENTS

[0018] Specific application scenarios and requirements of this specification are provided in the following descriptions, so that a person skilled in the art can manufacture and use content of this specification. For the person skilled in the art, various local modifications to the disclosed embodiments are apparent, and the general principles defined herein can be applied to other embodiments and applications without departing from the spirit and scope of this specification. Therefore, this specification is not limited to the shown embodiments, but has a widest scope consistent with that of the claims.

[0019] The terms used herein are only intended to describe a particular example embodiment, but do not impose a limitation. For example, unless otherwise specified in the context, the singular forms "one", "a", and "the" used herein can also include a plural form. When used in this specification, the terms "include", "comprise", and/or "have" indicate/indicates existence of an associated integer, step, operation, element, and/or component, but does not exclude existence of

one or more other features, integers, steps, operations, elements, components, and/or groups, or another feature, integer, step, operation, element, component, and/or group can be added to the system/method.

[0020] In consideration of the following descriptions, these and other features of this specification, operations and functions of related components of the structure, and economy of combination and manufacturing of components can be significantly improved. With reference to the accompanying drawings, all of these form a part of this specification. However, it should be clearly understood that the accompanying drawings are merely used for the purpose of illustration and description, and are not intended to limit the scope of this specification. It should be further understood that the accompanying drawings are not drawn to scale.

[0021] The flowchart used in this specification illustrates operations implemented by a system in some embodiments of this specification. It should be clearly understood that operations in the flowchart can be implemented out of sequence. On the contrary, operations can be implemented reversely or simultaneously. In addition, one or more other operations can be added to the flowchart. One or more operations can be removed from the flowchart.

[0022] In this specification, "X includes at least one of A, B, or C" means that X includes at least A, X includes at least B, or X includes at least C. In other words, X can include only any one of A, B, or C, or include both any combination of A, B, and C and other possible content/elements. The any combination of A, B, and C can be A, B, C, AB, AC, BC, or ABC.

[0023] In this specification, unless explicitly stated, an association relationship generated between structures can be a direct association relationship or an indirect association relationship. For example, when "A is connected to B" is described, unless it is explicitly stated that A is directly connected to B, it should be understood as that A can be directly connected to B or can be indirectly connected to B. For another example, when "A is above B" is described, unless it is explicitly stated that A is directly above B (A is adjacent to B and A is above B), it should be understood as that A can be directly above B, or A can be indirectly above B (A is separated from B by another element and A is above B), and so on.

[0024] For ease of description, terms appeared in this specification are first explained as follows.

[0025] A directed acyclic graph (DAG) is a directed graph consisting of vertices and edges. The vertices represent tasks or events, and the edges represent dependency relationships between the tasks or the events. The graph in the directed acyclic graph has a direction, indicating an execution sequence of the tasks or the events, while "acyclic" indicates that there is no cyclic dependency relationship, ensuring the ordering of the tasks or the events.

[0026] Task scheduling is a process of sorting, allocating, and executing a plurality of tasks

according to a certain rule and algorithm. The objective of task scheduling is to make proper use of system resources, improve system efficiency and responsiveness, and satisfy requirements and constraints of tasks.

[0027] Privacy-preserving computation is a technology set for implementing data analysis and computation while data is protected from being disclosed to the outside, so that the data is available and invisible. Data value is converted and released while data privacy security is fully protected. It can be learned that privacy-preserving computation is a computing method for protecting data (privacy data/sensitive data). Encryption processing is performed on the data, so that the data cannot be directly accessed or obtained in a computing process, to protect user privacy.

[0028] This specification provides a task scheduling method and a task scheduling system. The task scheduling system can perform the task scheduling method to implement efficient and proper task scheduling and arrangement. For example, in a multi-task application scenario, the task scheduling system can perform the task scheduling method, so that a plurality of machines can be used to process tasks in parallel.

[0029] The task scheduling method provided in this specification can be applied to a plurality of architecture systems. For example, the plurality of machines can be computing node machines in a system such as a distributed system, a micro service system, or a cloud architecture system. For example, the plurality of machines can be machines deployed in a same computer cluster, or can be machines deployed in computer clusters of different institutions. In some embodiments, the plurality of machines can communicate with each other to implement data transmission, for example, to implement transmission of intermediate computing data or computing result data, so that tasks with a dependency relationship can be executed. Communication between the plurality of machines can be implemented in different manners. For example, the plurality of machines can be in a direct communicative connection or an indirect communicative connection. The direct communicative connection can be that data is directly transmitted between the plurality of machines without being forwarded. The indirect communicative connection can be a communicative connection between the plurality of machines and a central platform, and communication between the plurality of machines is implemented through the central platform. In the indirect communicative connection, the plurality of machines can upload an intermediate file or a result file to storage space of the central platform. The storage space can be a cloud storage server, and can use a file storage system or an object storage service system.

[0030] The task scheduling method provided in this specification can be applicable to a plurality of task scheduling application scenarios. For example, in some big data analysis tasks, a task of computing data in different databases can be allocated to different machines for parallel computation, to improve computing efficiency. For example, in an Internet of Things system, there

are a large quantity of robots, intelligent devices, etc. The task scheduling system can perform a proper arrangement to perform centralized and unified scheduling on originally distributed tasks, to improve production and serving efficiency. For another example, in privacy-preserving computation, each institution can upload its encrypted file to a cloud storage platform, and then a computing platform performs unified computing resource scheduling, which can ensure data security and fully use computational resources at an edge end.

[0031] An application scenario of a privacy-preserving computation platform is used as an example for specific description. In the privacy-preserving computation platform, a plurality of institutions usually need to jointly participate in a privacy-preserving computation task to perform collaborative computation, to obtain a final computing result. However, raw data between different institutions is privacy data. The privacy data is usually not allowed to be transmitted between different institutions. For example, in a process of training a large model, to make the model more adaptable and versatile, institutions usually need to simultaneously participate in training. In this case, each institution holds some training data, and the training data is not allowed to be transmitted between different institutions. In some embodiments, each institution can clean, align, desensitize, and encrypt its own training data, and then upload the data to a cloud server for use in subsequent model training. In some embodiments, each institution can perform model training on a machine of the institution by using training data of the institution, and upload model parameters obtained through training by the institution to a cloud server for use in subsequent model training. For example, user portrait data needs to be used in training of an information recommendation model. The user portrait data relates to user data of a plurality of institutions, and the user data is not allowed to be disclosed. Each institution can desensitize and encrypt its own user data and then upload the user data to a cloud storage server of the institution for use in subsequent model training. Each institution can train an information recommendation model based on user data of the institution, and upload model parameters obtained through training by the institution to a cloud storage server of the institution for use in subsequent model training. For another example, compliant data of a plurality of parties needs to be used in training of a credit risk assessment model of a financial institution. The compliant data is not allowed to be disclosed. Each institution can desensitize and encrypt its own compliant data and then upload the data to a cloud storage server of the institution, so that the financial institution can communicate with a high-quality data institution for subsequent model training. Each institution can train a credit risk assessment model based on compliant data of the institution, and upload model parameters obtained through training by the institution to a cloud storage server of the institution for use in subsequent model training.

[0032] Training of a large model can be offline training or online training. A directed graph of a model training task refines a whole large training task into a plurality of small tasks that have a

dependency relationship. Based on this, a computing platform allocates the plurality of small tasks to machines of a plurality of institutions by using the task scheduling method provided in this specification, to fully use computational resources of the institutions.

[0033] It should be noted that the above-mentioned application scenario of the privacy-preserving computation platform is only one of a plurality of application scenarios provided in this specification. A person skilled in the art should understand that the task scheduling method and system described in this specification applying to other application scenarios also fall within the protection scope of this specification.

[0034] FIG. 1 is a schematic diagram of an application scenario of a task scheduling system 003 according to some embodiments of this specification. For ease of understanding, a privacy-preserving computation scenario is used as an example for description in the following description. As shown in FIG. 1, the application scenario can include a computing platform 001, a computing node 002, and a task scheduling system 003. The task scheduling system 003 can be configured to schedule a plurality of tasks in the computing platform 001 for computation, to schedule the plurality of tasks to a plurality of machines in the computing node 002.

[0035] The computing node 002 can be computing resources and equipped facilities deployed in an institution. Different computing nodes 002 can be deployed in different institutions. In the schematic diagram shown in FIG. 1, one computer point 002 can correspond to one institution. Privacy-preserving computation is used as an example. Different types of institutions may exist in different application scenarios. For example, in training of an information recommendation model, the institutions can include a merchant with an information recommendation requirement, a media platform, and a third-party data management platform. The media platform can provide rich user portrait data. A plurality of institutions can select, through user data matching, data of a common user as training data for use in subsequent model training. For another example, in training of a credit risk assessment model of a financial institution, the institutions can include a financial institution and a third-party data source institution. There may be one or more third-party data source institutions, which are used to provide high-quality compliant user data. More comprehensive training data can be provided for the risk assessment model by combining the data of the plurality of data source institutions.

[0036] The institution can connect the computing node 002 to the computing platform 001 in a plurality of manners, such as privatized deployment, clustered deployment, or software as a service (SaaS) deployment-free. The computing platform 001 can schedule a computing device in the computing node 002. For ease of description, a quantity of a plurality of computing nodes 002 is defined as M. M is an integer greater than 1. Each computing node 002 can include at least one computing device. Each computing device can be a server or a client. A plurality of computing

devices in each computing node 002 can form a computer cluster. For ease of description, the computer cluster corresponding to each computing node 002 is defined as a computer. M computing nodes 002 correspond to M machines. The computing node 002 can use any form of system architecture. For example, the computing node 002 can use a system architecture such as a cloud native architecture, a distributed architecture, a centralized architecture, or a micro service architecture.

[0037] The computing node 002 can be deployed with cloud storage space. For example, the computing node 002 can store data in a cloud storage server provided by a cloud service vendor by using a service such as cloud storage space lease and cloud storage data management that are provided by the cloud storage service vendor. For example, the computing node 002 can store data in a cloud storage server of the node through self-established storage. The storage space can use a file storage system or an object storage service (OSS) system. A system architecture of the storage space is not limited in this specification. In privacy-preserving computation, different computing nodes 002 can obtain an intermediate result by using the computing platform 001. The computing node 002 does not allow privacy data to be stored in a cloud storage server of the computing platform 001. Therefore, the computing node 002 cannot directly obtain privacy data of another institution.

[0038] The computing platform 001 can be communicatively connected to the computing node 002 and provide various services for the computing node 002. The computing platform 001 can resolve a collaboration problem between a plurality of computing nodes 002. Using privacy-preserving computation as an example, the computing platform 001 can provide a standardized privacy-preserving computation product, and different institutions can select proper access manners based on usage requirements and system architectures of the institutions. The computing platform 001 can make data available and invisible based on a privacy-preserving computation technology, to resolve a problem of security and privacy protection of collaborative data computation between enterprises, thereby implementing cross-institution task cooperation. The computing platform 001 can include at least one computing device. The at least one computing device can be a server, or a server cluster including a plurality of servers. The computing platform 001 can be a system of any architecture. For example, the computing platform 001 can use a system architecture such as a cloud native architecture, a distributed architecture, a centralized architecture, or a micro service architecture.

[0039] In privacy-preserving computation (for example, model training) that requires joint participation of a plurality of institutions, the computing platform 001 can relate to the M machines corresponding to the M computing nodes 002 to jointly execute a computing task, to fully use computational resources of the institutions, and improve execution efficiency of the computing

task. The task scheduling system 003 can perform task scheduling on the computing task in the computing platform 001, to allocate subtasks to the M machines in the M computing nodes 002 for execution. It should be noted that there can be a dependency relationship between different subtasks, that is, execution of a downstream subtask depends on an output result of an upstream subtask. Data transmission between machines in different computing nodes 002 needs to be implemented by using a cloud storage server. In other words, when a downstream subtask and an upstream subtask on which the downstream subtask depends are not executed on a same machine, the upstream subtask needs to upload an execution result to the cloud storage server, and the downstream subtask needs to download the execution result of the upstream subtask from the cloud storage server before being executed. When a downstream subtask and an upstream subtask on which the downstream subtask depends are executed on a same machine, the downstream subtask can directly obtain an execution result of the upstream subtask from a local memory of an execution machine. For data of a same size, duration required for directly obtaining the data from the local memory is less than duration required for downloading the data from the cloud storage server.

[0040] The task scheduling system 003 can perform task scheduling based on a directed graph. The computing platform 001 can divide a computing task into N tasks, and generate a target directed graph 100 based on a dependency relationship between the N tasks. The task scheduling system 003 can sequentially allocate the N tasks to M machines based on the task dependency relationship in the target directed graph 100 in an order of executable moments from early to late, and in each time of task allocation, allocate a task to a machine that can complete the task at the earliest time, to generate a scheduling table. In a task scheduling process, the task scheduling system 003 provided in this specification fully uses a dependency relationship between tasks, to avoid that associated tasks are randomly allocated, resulting in consumption of unnecessary communication time, thereby shortening overall time consumed by the tasks.

[0041] The task scheduling system 003 can include at least one computing device. The at least one computing device can perform the task scheduling method provided in this specification when running. In some embodiments, the task scheduling system 003 and the computing platform 001 can be devices independent of each other and communicatively connected to each other. In some embodiments, the task scheduling system 003 can alternatively be the computing platform 001.

[0042] FIG. 2 is a schematic diagram of a hardware structure of a computing device 200 according to some embodiments of this specification. In some embodiments, the computing device 200 can be a computing device in the task scheduling system 003. As shown in FIG. 2, the computing device 200 includes at least one storage medium 230 and at least one processor 220. In some embodiments, the computing device 200 can further include an internal communication bus

210. In some embodiments, the computing device 200 can further include a communication port 250. In some embodiments, the computing device 200 can further include an I/O component 260.

[0043] The internal communication bus 210 can be connected to different system components, including the storage medium 230 and the processor 220. The I/O component 260 supports input/output between the computing device 200 and other components.

[0044] The storage medium 230 can include a data storage apparatus. The data storage apparatus can be a non-transitory storage medium, or can be a transitory storage medium. For example, the data storage apparatus can include one or more of a disk 232, a read-only storage medium (ROM) 234, or a random access storage medium (RAM) 236. The storage medium 230 further includes at least one instruction set stored in the data storage apparatus. The instruction set is computer program code. The computer program code can include a program, a routine, an object, a component, a data structure, a process, a module, etc. for performing the task scheduling method provided in this specification.

[0045] The communication port 250 is used for data communication between the computing device 200 and the outside. For example, the computing device 200 can be connected to a network 130 through the communication port 250. The at least one processor 220 is communicatively connected to the at least one storage medium 230 through the internal communication bus 210. The at least one processor 220 is configured to execute the at least one instruction set. When the task scheduling system 001 runs, the at least one processor 220 reads the at least one instruction set and performs, based on an instruction of the at least one instruction set, the task scheduling method provided in this specification.

[0046] The processor 220 can perform all steps included in the task scheduling method. The processor 220 can be in a form of one or more processors. The processor 220 can send an execution instruction. The processor 220 can include one or more hardware processors, for example, a microcontroller, a microprocessor, a reduced instruction set computer (RISC), an application-specific integrated circuit (ASIC), an application-specific instruction set processor (ASIP), a central processing unit (CPU), a graphics processing unit (GPU), a physical processing unit (PPU), a microcontroller unit, a digital signal processor (DSP), a field programmable gate array (FPGA), an advanced RISC machine (ARM), a programmable logic device (PLD), any circuit or processor that can perform one or more functions, or any combination thereof.

[0047] For only the purpose of description, only one processor 220 is described in the computing device 200 in this specification. However, it should be noted that the computing device 200 in this specification can further include a plurality of processors. Therefore, the operations and/or method steps disclosed in this specification can be performed by one processor or can be performed jointly by a plurality of processors, as described in this specification. For example, if

the processor 220 in the computing device 200 in this specification performs step A and step B, it should be understood as that step A and step B can be jointly or separately performed by two different processors 220 (for example, a first processor performs step A, and a second processor performs step B, or a first processor and a second processor jointly perform step A and step B).

[0048] FIG. 3 is a schematic flowchart of a task scheduling method P100 according to some embodiments of this specification. As described above, the at least one computing device 200 in the task scheduling system 003 can perform the task scheduling method P100. Specifically, the storage medium 230 can store at least one group of instruction sets used for task scheduling. The processor 220 can be communicatively connected to the storage medium 230. The processor 220 can read an instruction set stored in a local storage medium and/or a database of the processor, and then perform the task scheduling method P100 based on a stipulation of the instruction set.

[0049] As shown in FIG. 3, the task scheduling method P100 can include the following step.

[0050] S110: Obtain a target directed graph 100.

[0051] The target directed graph 100 is a directed acyclic graph formed based on a dependency relationship between N tasks, where N is an integer greater than 1. As described above, a directed acyclic graph can represent dependency relationships between a plurality of tasks by using vertices and edges. Each vertex represents a task, and at least one vertex is a start vertex used to represent a start task. Each edge represents a dependency relationship between two tasks. The edge is an edge with a direction, that is, pointing from an upstream task to a downstream task, to indicate that execution of the downstream task depends on an execution result of the upstream task. It should be noted that the start task represented by the start vertex has no upstream task. There is no ring in the directed acyclic graph, that is, there is no cyclic dependency, which ensures orderly execution of tasks.

[0052] The target directed graph 100 can be a directed acyclic graph formed based on a dependency relationship between N tasks. In some embodiments, the N tasks can be a plurality of subtasks obtained by dividing one task, where N is an integer greater than 1. For example, the N tasks can be a plurality of subtasks obtained by decomposing an offline large model training task. The plurality of subtasks can include a preprocessing subtask such as data augmentation. The plurality of subtasks can further include dividing a dataset into a training set and a verification set. A relatively large training set can be divided into a plurality of parts, so that the training subtasks are executed by using a plurality of machines. The plurality of subtasks can further include training a model by using a specified training set. The plurality of subtasks can further include integrating results of the plurality of training subtasks. The plurality of subtasks can further include verifying a large model obtained through training on the verification set.

[0053] In some embodiments, the dependency relationship can include that execution of a

downstream task depends on an output result of an upstream task. The above-mentioned offline large model training task continues to be used as an example. For example, a model training subtask depends on execution results of data preprocessing and/or dataset division subtasks. An execution result of the data preprocessing subtask can be outputting a dataset obtained after processing such as data augmentation. An execution result of the dataset division subtask can be outputting one or more training sets obtained after division. For example, a training result integration subtask depends on training results of the plurality of training subtasks. An execution result of the training subtask can be outputting a training model obtained through training by using a corresponding training set. For another example, a large model verification subtask depends on the training result integration subtask. The training result integration subtask is integrating training models obtained by the plurality of training subtasks and outputting a large model obtained in this offline training task. An output result of each subtask can be stored in a local machine that executes the subtask, or can be uploaded to a cloud storage server. When a downstream task and an upstream task of the downstream task are not executed on a same machine, the downstream task can download required output data of the dependent task from the cloud storage server. When a downstream task and an upstream task of the downstream task are executed on a same machine, the downstream task can directly download required output data of the dependent task from a memory of the local machine that executes the subtask.

[0054] In some embodiments, the N tasks can include at least one start task. Each start task depends on no upstream task. Execution of the start task does not depend on an output result of an upstream task. Data required when the start task is executed can be obtained from specified storage space. For example, when the start task is a training subtask, training data and a pre-training model need to be used. In this case, the required data and model can be obtained from a cloud storage server.

[0055] The target directed graph 100 can be represented in any form of data structure. For example, the target directed graph 100 can be an adjacency matrix. For example, the target directed graph 100 can alternatively be an adjacency list. A data structure of the target directed graph 100 is not limited in this specification. In some embodiments, the target directed graph 100 can be generated by the computing platform 001 and pre-stored in the computing platform 001. The task scheduling system 003 directly uses the target directed graph 100 when performing the task scheduling method P100. In some embodiments, the target directed graph 100 can alternatively be generated by another system, and the task scheduling system 003 invokes the target directed graph from the outside when performing the task scheduling method P100.

[0056] As shown in FIG. 3, the task scheduling method P100 can further include the following step.

[0057] S130: Sequentially allocate the N tasks to M machines based on the dependency relationship in an order of executable moments from early to late, to generate a scheduling table.

[0058] The executable moment is determined based on a moment at which all upstream tasks on which each task depends are completed and expected idle moments of the M machines. In each time of task allocation, a to-be-allocated task is allocated to a machine that is expected to be capable of completing the task at the earliest time, where M is an integer greater than 1.

[0059] Before step S130 is specifically described, two concepts are first clarified. In this specification, "moment" is used to represent an instantaneous time point, and "duration" is used to represent a time length between two moments.

[0060] When an executable moment of each task is determined in the task scheduling method P100, a moment at which all upstream tasks of the current task are completed and expected idle moments of the M machines corresponding to the M computing nodes 002 are comprehensively considered. In this specification, that a quantity of machines that can be allocated is M is used as an example for description, where M is an integer greater than 1. As described above, the dependency relationship can mean that execution of a downstream task depends on an output result of an upstream task. Therefore, for each task, the current task cannot start to be executed until all upstream tasks on which the current task depends are completed. It should be specially noted that, because the start task has no upstream task, it can be considered that a quantity of upstream tasks of the start task is zero. One or more start tasks may exist in the target directed graph 100. When the executable moment of each task is determined, in addition to the moment at which all the upstream tasks of the current task are completed, the expected idle moments of the M machines that can be allocated further need to be considered. In other words, even if all the upstream tasks of the current task are completed, the current task still cannot be executed if no machine is idle at the moment at which all the upstream dependent tasks are completed. Similarly, even if a machine is idle, the current task still cannot be executed if all the upstream tasks of the current task are not completed.

[0061] In the task scheduling method P100, tasks are allocated in an order of executable moments from early to late, and in each time of task allocation, an earliest executable task is selected for allocation. A task executable moment does not remain unchanged after being initially determined, but may change based on each round of task allocation.

[0062] FIG. 4 is a schematic flowchart of each time of task allocation according to some embodiments of this specification. As shown in FIG. 4, step S130 can include the following step.

[0063] S131: Determine an earliest executable target task from to-be-allocated tasks.

[0064] The to-be-allocated tasks are tasks that have not been allocated in the target directed graph 100 after a previous round of task allocation. For a first round of task allocation, there is no

previous round of task allocation, and the to-be-allocated tasks can include all the N tasks. The target task is an earliest executable task in the to-be-allocated tasks. The earliest executable task means that an executable moment of the target task is the earliest in all the to-be-allocated tasks. In each time of task allocation, the target task in the current round of task allocation is determined, and the target task is allocated to a corresponding machine.

[0065] Step S131 can include: determining at least one executable candidate task from the to-be-allocated tasks; and determining the target task from the at least one candidate task.

[0066] All upstream tasks on which each candidate task depends have been allocated. In each time of allocation, all tasks in the to-be-allocated tasks can be classified into a candidate task or a non-candidate task based on whether all dependent tasks are allocated. All upstream tasks on which the candidate task depends have been allocated. For the start task, because the start task has no upstream dependent task, it is considered by default that all upstream tasks of the start task have been allocated. In each time of allocation, at least one candidate task is determined from the to-be-allocated tasks.

[0067] FIG. 5 is an example diagram of determining a target task according to some embodiments of this specification. In FIG. 5, the target directed graph 100 includes eight tasks, that is, the quantity N of tasks is equal to 8. An arrow points from an upstream task to a downstream task to indicate a dependency relationship between the tasks. A dashed-line arrow indicates that an upstream task at the beginning of the arrow has been allocated. FIG. 5 shows a case that a task E has been allocated after a first round of allocation. In a second round of allocation, all remaining tasks other than the task E that has been allocated are to-be-allocated tasks. All upstream tasks E of a task C have been allocated, and a task A and a task B have no upstream task. Therefore, the task C, the task A, and the task B are determined as candidate tasks in the second round of allocation. Next, a target task is determined from at least one candidate task.

[0068] As described above, the target task is the earliest executable task in the to-be-allocated tasks. Similarly, the target task is also an earliest executable task in the at least one candidate task. In some embodiments, when a quantity of candidate tasks is 1, the candidate task is the target task. When a quantity of candidate tasks is greater than 1, to determine the target task, an executable moment of each candidate task needs to be determined. In some embodiments, the determining the target task from the at least one candidate task can include: determining an earliest idle moment of the M machines after a previous round of task allocation, where the earliest idle moment is an earliest moment in M expected idle moments at which the M machines respectively complete allocated tasks; determining a moment at which all upstream tasks on which each candidate task depends are completed; for each candidate task, determining, as an executable moment of the current candidate task, a larger one in the earliest idle moment and the moment at which all the

upstream tasks on which the current candidate task depends are completed; and determining, from the at least one candidate task, a candidate task corresponding to an earliest moment in the executable moments as the target task. In other words, for each candidate task, the executable moment is an earliest moment in moments at which all upstream tasks on which the current candidate task depends are completed and a machine is idle. In some embodiments, when there is more than one candidate task corresponding to the earliest moment in the executable moments, any one of the candidate tasks can be determined as the target task.

[0069] As described above, the executable moment of each task is determined jointly by the moment at which the upstream dependent tasks are completed and the expected idle moments of the M machines. Continuing to use FIG. 5 as an example, t0 represents a start moment of task scheduling, and it is assumed that all machines that can be allocated are in an idle state at the moment t0. In FIG. 5, that a quantity of machines that can be allocated is 3, that is, M is equal to 3, is used as an example for description. When the second round of task allocation is performed, it can be determined that, after the previous round of task allocation, an expected idle moment is t1 after a machine 1 completes the task E, no task is allocated to a machine 2 and a machine 3, and expected idle moments of both the machine 2 and the machine 3 are t0. It is determined that the expected idle moment t0 in the three expected idle moments of the three machines is the earliest idle moment.

[0070] For the candidate tasks in the current round, that is, the task C, the task A, and the task B, all upstream tasks of the task C are completed at the moment t1, and the task A and the task B have no upstream task, so that it is considered by default that all upstream tasks of the task A and the task B are completed at the moment t0. For each candidate task, the larger one in the earliest idle moment and the moment at which all the upstream tasks on which the current candidate task depends are completed is determined as the executable moment of the current candidate task. Therefore, it can be determined that an executable moment of the task C is a larger one t1 in t0 and t1, and executable moments of the task A and the task B are a larger one t0 in t0 and t0. An earliest executable moment in the executable moments t1, t0, and t0 of the task C, the task A, and the task B is t0. Candidate tasks corresponding to the earliest executable moment t0 are the task A and the task B, and any one of the task A and the task B can be selected as the target task. In FIG. 5, the task A is selected as the target task in the current round of task allocation.

[0071] In the task scheduling method P100, tasks are allocated in an order of executable moments of the tasks from early to late, and in each round of allocation, a task whose executable moment is the earliest is selected as a target task, and the target task is allocated.

[0072] As shown in FIG. 4, step S130 can further include the following step.

[0073] S133: Determine M expected completion moments at which the target task is executed

on the M machines.

[0074] An execution result of an upstream task needs to be used when each task in the target directed graph 100 is executed. The execution result of the upstream task can be stored in a machine in the computing node 002 that executes the upstream task and cloud storage space. The start task has no upstream task, and data required by the start task is stored in the cloud storage space. Required data needs to be obtained before each task is executed. When a downstream task and all upstream tasks are executed on a same machine, execution results of the upstream tasks are stored on the same machine and do not need to be downloaded from the cloud storage space. A downstream task that has a plurality of upstream tasks needs to determine whether each upstream task of the downstream task has been executed on a current machine. If the upstream task is executed on another machine, an execution result of the current upstream task needs to be obtained from the cloud storage space.

[0075] In some embodiments, the determining M expected completion moments at which the target task is executed on the M machines can include: determining an expected start moment at which the target task is executed on each of the M machines; determining expected duration required when the target task is executed on each machine, where the expected duration includes expected execution duration and expected communication duration, and the expected communication duration includes duration for the target task to obtain required data; and adding the expected start moment at which the target task is executed on each machine and the expected duration to obtain an expected completion moment at which the target task is executed on each machine, to obtain the M expected completion moments.

[0076] The expected start moment of the target task on each machine is an earliest moment at which the target task can start to be executed when the target task is allocated to the current machine. In one aspect, execution of the target task depends on an execution result of an upstream task, and therefore, the target task can be executed only after all upstream tasks are completed. In another aspect, when the target task is executed on the current machine, computational resources can be released only after the current machine complete all allocated tasks. Therefore, when the expected start moment is determined, an execution completion moment of all the upstream tasks and the expected idle moment of each machine need to be considered comprehensively.

[0077] In some embodiments, the determining an expected start moment at which the target task is executed on each of the M machines can include: determining an expected idle moment at which each machine completes an allocated task; determining a moment at which all upstream tasks on which the target task depends are completed; and for each machine, using a larger one in the expected idle moment of the current machine and the moment at which all the upstream tasks on which the target task depends are completed as an expected start moment at which the target

task is executed on the current machine.

[0078] FIG. 6 is an example diagram of determining an expected completion moment according to some embodiments of this specification. As shown in FIG. 6, in the previous round of allocation, the task E, the task A, and the task B have been respectively allocated to the machine 1, the machine 2, and the machine 3. The task C is the target task in the current round of allocation. Expected idle moments of the machine 1, the machine 2, and the machine 3 are respectively t1, t2, and t3. A completion moment of the upstream task A on which the target task C depends is t1. If the target task C is executed on the machine 1, an expected start moment is a larger one in the expected idle moment t1 of the machine 1 and the moment t1 at which all the upstream tasks are completed, and is still t1. If the target task C is executed on the machine 2, an expected start moment is a larger one t2 in the expected idle moment t2 of the machine 2 and the moment t1 at which all the upstream tasks are completed. If the target task C is executed on the machine 3, an expected start moment is a large one t3 in the expected idle moment t3 of the machine 3 and the moment t1 at which all the upstream tasks are completed. Therefore, the expected start moments at which the target task C is executed on the machine 1, the machine 2, and the machine 3 are respectively t1, t2, and t3.

[0079] The expected duration required when the target task is executed on each machine can be expected total time consumed by the target task on the current machine from a preparation work before the task is executed to an end of task execution. In some embodiments, the expected duration includes expected communication duration and expected execution duration. The expected communication duration can be the preparation work before the task is executed, and can include a data preparation work. Even for a same task, the expected duration required when the task is executed on different machines may be different.

[0080] In some embodiments, the expected communication duration can include duration for the target task to obtain required data. The data required by the target task can include an execution result of an upstream task. When the target task and an upstream task on which the target task depends are not executed on a same machine, an execution result of the upstream task is stored in a cloud memory of the computing platform 001, and the target task needs to obtain the data from the cloud memory of the computing platform 001. When the target task and an upstream task on which the target task depends are executed on a same machine, an execution result of the upstream task is stored in a machine that executes the target task, and the target task can directly obtain execution result data from the machine that executes the target task without obtaining the data from the cloud memory of the computing platform 001. Communication time for directly obtaining the data from the local machine is less than time consumed for downloading the data from the cloud memory. When the target task and the upstream task are executed on a same machine, it can

be considered that required expected communication duration is zero. In some embodiments, expected communication duration for different machines to download same data from a cloud is the same.

[0081] Using offline training of a large model in privacy-preserving computation as an example, large model training often relates to large-scale model parameters and training data. Downloading a training model and the training data from a cloud storage platform becomes an important time-consuming part in the training process. If a downstream task and an upstream task are arranged on a same computer for execution, time consumed for downloading data can be greatly shortened.

[0082] Continuing to refer to FIG. 6, when the target task C and the upstream task are executed on a same machine, expected communication duration is zero; and when the target task and the upstream task are executed on different machines, expected communication duration for downloading an execution result of the upstream task from the cloud storage space is T_D. The upstream task of the target task C is the task E. When the target task C is arranged on the machine 1 for execution, because the upstream task E has been executed on the machine 1, the expected communication duration is zero. When the target task C is arranged on the machine 2 for execution, because the upstream task E has not been executed on the machine 2, the expected communication duration is T_D. Similarly, when the target task C is arranged on the machine 3 for execution, because the upstream task E has not been executed on the machine 3, the expected communication duration is also T_D. In FIG. 6, the target task C has only one upstream task. When the target task has a plurality of upstream tasks, execution of each of the plurality of upstream tasks needs to be considered.

[0083] In some embodiments, when the target task depends on at least one upstream task, the expected communication duration can include maximum duration in at least one communication duration corresponding to the at least one upstream task. When the target task obtains an execution result of the at least one upstream task, for each upstream task, if the current upstream task is not on a machine the same as that executes the target task, execution result data corresponding to the current upstream task needs to be downloaded from the cloud storage server; or if the current upstream task is on a machine the same as that executes the target task, the execution result data is directly obtained from the same local machine. Expected communication duration required for obtaining the data locally is less than expected communication duration for downloading the data from the cloud storage server. In other words, for each upstream task in the at least one upstream task, communication duration required when the target task and the current upstream task are executed on a same machine is less than communication duration required when the target task and the current upstream task are not executed on a same machine. In some embodiments, for each

upstream task of the target task, communication duration required when the target task and the current upstream task are executed on a same machine is zero or near zero. In some embodiments, when the target task needs to obtain a plurality of execution results, operations of obtaining data can be performed in parallel, and the expected communication duration is the maximum duration in the at least one expected communication duration corresponding to the at least one upstream task.

[0084] The expected execution duration can be a time length from a start to an end of execution of the target task. In some embodiments, the expected execution duration is positively correlated with task complexity, and a more complex task indicates longer expected execution duration. In some embodiments, the expected execution duration is positively correlated with a data scale, and a larger data scale indicates longer expected execution duration.

[0085] In some embodiments, expected execution duration required when the target task is executed on different machines is the same. Continuing to refer to FIG. 6, expected execution time required when the target task C is executed on the three machines is the same and is T_E. The expected start moment at which the target task is executed on each machine and the expected duration are added to obtain an expected completion moment at which the target task is executed on each machine. Therefore, it can be learned that, an expected completion moment at which the target task C is executed on the machine 1 is obtained by adding the expected start moment t1 and the expected duration T_E, that is, t4; an expected completion moment at which the target task C is executed on the machine 2 is obtained by adding the expected start moment t2 and the expected duration (the expected communication duration T_D+the expected execution duration T_E), that is, t5; and an expected completion moment at which the target task C is executed on the machine 3 is obtained by adding the expected start moment t3 and the expected duration (the expected communication duration T_D+the expected execution duration T_E), that is, t6. So far, it can be determined that three expected completion moments at which the target task C is executed on the machine 1, the machine 2, and the machine 3 in the current round of task allocation are respectively t4, t5, and t6.

[0086] In some embodiments, expected execution duration required when the target task is executed on different machines is negatively correlated with a machine processing speed. When different machines execute the target task, expected execution duration required when the target task is executed on the different machines may be different due to a difference between performance of the machines. When a same task is executed, a machine with a larger processing speed consumes shorter expected execution duration, and on the contrary, a machine with a smaller processing speed consumes longer expected execution duration.

[0087] As shown in FIG. 4, step S130 can further include the following step.

[0088] S135: Allocate the target task to a target machine, where the target machine is a machine corresponding to an earliest moment in the M expected completion moments.

[0089] As described above, the M expected completion moments corresponding to the target task on the M machines can be determined jointly by the expected start moments of the target task on the M machines and the expected duration in which the target task is executed. After the M expected completion moments are obtained, the target task is allocated to the machine corresponding to the earliest moment in the M expected completion moments. Continuing to refer to FIG. 6, in the current round of task allocation, the three expected completion moments at which the target task C is executed on the machine 1, the machine 2, and the machine 3 are respectively t4, t5, and t6. An earliest moment in the three expected completion moments is t4, and the machine 1 corresponding to t4 is the target machine. The target task C is allocated to the target machine, that is, the machine 1.

[0090] In conclusion, in step S130, the N tasks in the target directed graph 100 are sequentially allocated to the M machines in the order of the executable moments from early to late based on the dependency relationship between the N tasks. In each time of task allocation, a target task is selected from at least one candidate task, and the target task is allocated to a machine that is expected to be capable of executing the task at the earliest time.

[0091] As shown in FIG. 3, the task scheduling method P100 can further include the following step.

[0092] S150: Output the scheduling table.

[0093] The scheduling table can represent each of the M machines and an allocated task corresponding to each machine. The scheduling table can be data in any format. The scheduling table can be stored by using a database. The scheduling table can use any data structure. For example, the scheduling table can be a dictionary. For example, the scheduling table can be a matrix. A format and a data structure of the scheduling table are not limited in this specification.

[0094] In conclusion, in the task scheduling method P100 provided in this specification, based on the dependency relationship between the tasks in the target directed graph 100, when task scheduling is performed, communication time consumed for obtaining an execution result of an upstream task by a downstream task is fully considered, and in each round of allocation, an earliest executable task is allocated to a machine that can execute the task at the earliest time. In some related technologies, a task is randomly allocated to any of the M machines without considering a case that additional data transmission time is consumed if a downstream task and its upstream task are not executed on a same machine. Compared with the random allocation manner, the task scheduling method P100 can reduce a quantity of times of downloading data from a cloud memory, thereby shortening overall task completion time.

[0095] FIG. 7 is an example diagram of task scheduling according to some embodiments of this specification. As shown in FIG. 7, the target directed graph 100 includes six tasks. A dependency relationship between the tasks is indicated by an arrow in the graph, and the arrow points from an upstream task to a downstream task. To facilitate description of how each round of allocation is performed, expected communication duration and expected execution duration are marked on a right side of each task. "Expected communication duration T_D: x1, x2" indicates that the downstream task has two upstream dependent tasks, and expected communication duration required when a machine executing the downstream task obtains execution result data of the two upstream tasks from a cloud is respectively x1 and x2. If a downstream task and an upstream task are executed on a same machine, expected communication duration required for obtaining an execution result of the upstream task by the downstream task is 0. A start task has no upstream dependent task, and it is considered by default that required data needs to be obtained from the cloud. "Expected execution duration T_E: y" indicates that expected execution duration required when the task is executed on each machine is y. In this example, an example in which expected execution time required when a same task is executed on different machines is the same is used for description.

[0096] The following specifically describes how each round of allocation is performed.

[0097] In a first round of allocation, all tasks are to-be-allocated tasks. A task A and a task F have no upstream task, and are candidate tasks in the current round of allocation. Expected idle moments of both a machine 1 and a machine 2 are 0, and the earliest idle moment is a smaller one in the two expected idle moments, that is, 0. An executable moment of the task A is a larger one in a moment 0 at which all upstream tasks are completed and the earliest idle moment 0 of the machine, that is, 0. Similarly, an executable moment of the task B is 0. Because the two candidate tasks have a same executable moment, any one of the two tasks can be selected as the target task. In the example, the task A is selected as the target task. An expected start moment of the task A on the machine 1 is a larger one in the moment 0 at which the upstream tasks are completed and the expected idle moment 0 of the machine 1, that is, 0. Expected duration required when the task A is executed on the machine 1 is expected communication duration 4 plus expected execution duration 2, that is, 6. An expected completion moment at which the task A is executed on the machine 1 is the expected start moment 0 plus the expected duration 6, that is, 6. Similarly, an expected start moment of the task A on the machine 2 is 0, expected duration is 6, and an expected completion moment is 6. The expected completion moments of the task A on the two machines are the same. Therefore, any one of the two machines can be selected as the target machine to execute the task A. In the example, the machine 1 is selected as the target machine. In the first round of allocation, the task A is allocated to the machine 1.

[0098] In a second round of allocation, tasks other than the task A all are to-be-allocated tasks. A task F has no upstream task, and all upstream tasks of the task B are completed. The task F and the task B are candidate tasks in the current round of allocation. Expected idle moments of both the machine 1 and the machine 2 are respectively 6 and 0, and the earliest idle moment is a smaller one in the two expected idle moments, that is, 0. An executable moment of the task F is a larger one in a moment 0 at which all upstream tasks are completed and the earliest idle moment 0 of the machine, that is, 0. An executable moment of the task B is a larger one in a moment 6 at which all upstream tasks are completed and the earliest idle moment 0 of the machine, that is, 6. Therefore, the task F whose executable moment is earlier is selected as the target task. An expected start moment of the task F on the machine 1 is a larger one in the moment 0 at which the upstream tasks are completed and the expected idle moment 6 of the machine 1, that is, 6. Expected duration required when the task F is executed on the machine 1 is expected communication duration 4 plus expected execution duration 3, that is, 7. An expected completion moment at which the task F is executed on the machine 1 is the expected start moment 6 plus the expected duration 7, that is, 13. Similarly, an expected start moment of the task F on the machine 2 is 0, expected duration is 7, and an expected completion moment is 7. Therefore, the machine 2 whose expected completion moment is earlier is selected as the target machine. In the second round of allocation, the task F is allocated to the machine 2.

[0099] In the following description, repeated procedures are not described again.

[0100] In a third round of allocation, the task B and a task C are candidate tasks in the current round of allocation. The earliest idle moment is a smaller one 6 in two expected idle moments of the machine 1 and the machine 2. Executable moments of the task B and the task C are respectively 6 and 7. Therefore, the task B is selected as the target task. An expected start moment of the task B on the machine 1 is 6. Expected duration required when the task B is executed on the machine 1 is expected communication duration 0 (the upstream task A has been executed on the machine 1) plus expected execution duration 5, that is, 5. An expected completion moment at which the task B is executed on the machine 1 is the expected start moment 6 plus the expected duration 5, that is, 11. Similarly, an expected start moment of the task B on the machine 2 is 7, expected duration is 9, and an expected completion moment is 16. Therefore, the machine 1 is selected as the target machine. In the third round of allocation, the task B is allocated to the machine 1.

[0101] In a fourth round of allocation, the task C and a task D are candidate tasks in the current round of allocation. The earliest idle moment of the machine 1 and the machine 2 is 7. Executable moments of the task C and the task D are respectively 7 and 11. Therefore, the task C is selected as the target task. An expected start moment of the task C on the machine 1 is 11. Expected duration required when the task C is executed on the machine 1 is expected communication duration 3

(obtaining an execution result of the upstream task F) plus expected execution duration 2, that is, 5. An expected completion moment at which the task C is executed on the machine 1 is the expected start moment 11 plus the expected duration 5, that is, 16. Similarly, an expected start moment of the task C on the machine 2 is 7, expected duration is 6, and an expected completion moment is 13. Therefore, the machine 2 is selected as the target machine. In the fourth round of allocation, the task C is allocated to the machine 2.

[0102] In a fifth round of allocation, the task D and a task E are candidate tasks in the current round of allocation. The earliest idle moment of the machine 1 and the machine 2 is 11. Executable moments of the task D and the task E are respectively 11 and 13. Therefore, the task D is selected as the target task. An expected start moment of the task D on the machine 1 is 11. Expected duration required when the task D is executed on the machine 1 is expected communication duration 0 plus expected execution duration 1, that is, 1. An expected completion moment at which the task D is executed on the machine 1 is the expected start moment 11 plus the expected duration 1, that is, 12. Similarly, an expected start moment of the task D on the machine 2 is 13, expected duration is 3, and an expected completion moment is 16. Therefore, the machine 1 is selected as the target machine. In the fifth round of allocation, the task D is allocated to the machine 1.

[0103] In a sixth round of allocation, the task E is the only candidate task in the current round of allocation, and is also the target task. An expected start moment of the task E on the machine 1 is 13 (the moment 13 at which all upstream tasks are completed is later than an expected idle moment 12 of the machine 1). Expected duration required when the task E is executed on the machine 1 is expected communication duration 3 plus expected execution duration 2, that is, 5. An expected completion moment at which the task E is executed on the machine 1 is the expected start moment 13 plus the expected duration 5, that is, 18. Similarly, an expected start moment of the task E on the machine 2 is 13, expected duration is 4, and an expected completion moment is 17. Therefore, the machine 2 is selected as the target machine. In the fifth round of allocation, the task E is allocated to the machine 2.

[0104] In conclusion, after six rounds of task allocation are performed, a task scheduling table is obtained: {machine 1: task A, task B, and task D; machine 2: task F, task C, and task E}. It can be learned that, after allocation is performed by using the task scheduling method provided in this specification, tasks in an upstream-downstream dependency relationship are allocated to a same machine as far as possible, thereby shortening consumed communication time and improving overall task execution efficiency.

[0105] Another aspect of this specification provides a computer-readable non-temporary storage medium. The computer-readable non-temporary storage medium stores at least one group of executable instructions use to perform task scheduling. When the executable instructions are

executed by a processor, the executable instructions instruct the processor to implement steps of the task scheduling method P100 in this specification. In some possible implementations, aspects of this specification can alternatively be implemented in a form of a program product, including program code. When the program product runs on the task scheduling system 003, the program code is used to enable the task scheduling system 003 to perform steps of the task scheduling method P100 described in this specification. The program product used to implement the above-mentioned method can be a portable compact disk read-only memory (CD-ROM) including program code, and can run on the task scheduling system 003. However, the program product in this specification is not limited thereto. In this specification, a readable storage medium can be any tangible medium that includes or stores a program, and the program can be used by or in combination with an instruction execution system. The program product can be any combination of one or more readable media. The readable medium can be a readable signal medium or a readable storage medium. For example, the readable storage medium can be but is not limited to an electrical system, apparatus, or device, a magnetic system, apparatus, or device, an optical system, apparatus, or device, an electromagnetic system, apparatus, or device, an infrared system, apparatus, or device, or a semiconductor system, apparatus, or device, or any combination thereof. More specific examples of the readable storage medium include: an electrical connection including one or more conducting wires, a portable disk, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or flash memory), an optical fiber, a portable compact disc read-only memory (CD-ROM), an optical storage component, a magnetic storage component, or any proper combination thereof. The computer-readable storage medium can include a data signal propagated in a baseband or as part of a carrier, and carries readable program code. The data signal propagated in such a way can be in a plurality of forms, and includes but is not limited to an electromagnetic signal, an optical signal, or any proper combination thereof. The readable storage medium can further be any readable medium other than the readable storage medium, and the readable medium can be used to send, propagate, or transmit a program used by or in combination with an instruction execution system, apparatus, or component. The program code included in the readable storage medium can be transmitted in any proper medium, including but not limited to a wireless medium, a wired medium, an optical cable, an RF medium, etc., or any proper combination thereof. The program code for performing operations in this specification can be written in any combination of one or more program design languages. The program design language includes an object-oriented program design language, such as Java or C++, and further includes a conventional procedural program design language, such as a "C" language or a similar program design language.

[0106] Specific embodiments of this application are described above. Other embodiments fall

within the scope of the appended claims. In some cases, the actions or steps described in the claims can be performed in an order different from that in the embodiments, and the desired results can still be achieved. In addition, the process depicted in the accompanying drawings does not necessarily need a particular order or consecutive order to achieve the desired results. In some implementations, multitasking and parallel processing are possible or may be advantageous.

[0107] In conclusion, after reading the detailed disclosure, a person skilled in the art can understand that the above-mentioned detailed disclosure can be presented only in an example manner, and can impose no limitation. Although it is not explicitly stated herein, a person skilled in the art can understand that the requirements of this application include various proper changes, improvements, and modifications to the embodiments. These changes, improvements, and modifications are intended to be proposed in this application, and fall within the spirit and scope of the example embodiments of this application.

[0108] In addition, some terms in this application are used to describe the embodiments of this application. For example, "one embodiment", "an embodiment", and/or "some embodiments" mean that particular features, structures, or characteristics described with reference to the embodiments can be included in at least one embodiment of this application. Therefore, it can be emphasized and understood that two or more of "an embodiment", "one embodiment", or "alternative embodiments" referred in parts of this application do not necessarily indicate the same embodiment. In addition, particular features, structures, or characteristics can be properly combined in one or more embodiments of this application.

[0109] It should be understood that in the above-mentioned descriptions of the embodiments of this application, to help understand a feature, for the purpose of simplifying this application, various features are combined in a single embodiment, accompanying drawing, or description thereof in this application. However, this does not mean that the combination of these features is necessary. It is entirely possible for a person skilled in the art to mark some devices and understand the devices as separate embodiments when reading this application. In other words, the embodiments in this application can also be understood as an integration of a plurality of secondary embodiments. Content of each secondary embodiment is also true when features are less than all features in one of the above-mentioned disclosed embodiments.

[0110] Each patent, patent application, publication of a patent application, and materials, for example, articles, books, instructions, publications, documents, and products, that are cited in this specification and that are other than any related historical litigation documents, any same historical litigation documents that may be inconsistent or conflict with this document, or any same historical litigation documents that may impose limiting impact on the widest scope of the claims can be incorporated herein by reference and used for all purposes now or later associated with this

document. In addition, the terms in this document shall prevail if there is any inconsistency or conflict between the descriptions, definitions, and/or use of the terms associated with any included material and the descriptions, definitions, and/or use of the terms associated with this document.

[0111] Finally, it should be understood that the principles of the implementation solutions of this application are described in the implementation solutions of the application disclosed in this specification. Other modified embodiments also fall within the scope of this application. Therefore, the embodiments disclosed in this application are merely examples and not limitations. A person skilled in the art can use an alternative configuration according to the embodiments of this application to implement the application in this application. Therefore, the embodiments of this application are not limited to the embodiments accurately described in the application.

CLAIMS

What is claimed is:

1. A task scheduling method, comprising:

obtaining a target directed graph, wherein the target directed graph is a directed acyclic graph formed based on a dependency relationship between N tasks, and N is an integer greater than 1;

sequentially allocating the N tasks to M machines based on the dependency relationship in an order of executable moments from early to late, to generate a scheduling table, wherein M is an integer greater than 1, and in each time of task allocation, a to-be-allocated task is allocated to a machine that is expected to be capable of completing the task at the earliest time, wherein the executable moment is determined based on a moment at which all upstream tasks on which each task depends are completed and expected idle moments of the M machines; and

outputting the scheduling table.

2. The method according to claim 1, wherein the dependency relationship comprises that execution of a downstream task depends on an output result of an upstream task, the N tasks comprise at least one start task, and each start task depends on no upstream task.

3. The method according to claim 1, wherein each time of task allocation comprises:

determining an earliest executable target task from to-be-allocated tasks;

determining M expected completion moments at which the target task is executed on the M machines; and

allocating the target task to a target machine, wherein the target machine is a machine corresponding to an earliest moment in the M expected completion moments.

4. The method according to claim 3, wherein the determining an earliest executable target task from to-be-allocated tasks comprises:

determining at least one executable candidate task from the to-be-allocated tasks, wherein all upstream tasks on which each candidate task depends have been allocated; and

determining the target task from the at least one candidate task.

5. The method according to claim 4, wherein the determining the target task from the at least one candidate task comprises:

determining an earliest idle moment of the M machines after a previous round of task allocation, wherein the earliest idle moment is an earliest moment in M expected idle moments at which the M machines respectively complete allocated tasks;

determining a moment at which all upstream tasks on which each candidate task depends are completed;

for each candidate task, determining, as an executable moment of the current candidate task,

a larger one in the earliest idle moment and the moment at which all the upstream tasks on which the current candidate task depends are completed; and

determining, from the at least one candidate task, a candidate task corresponding to an earliest moment in the executable moments as the target task.

6. The method according to claim 3, wherein the determining M expected completion moments at which the target task is executed on the M machines comprises:

determining an expected start moment at which the target task is executed on each of the M machines;

determining expected duration required when the target task is executed on each machine, wherein the expected duration comprises expected execution duration and expected communication duration, and the expected communication duration comprises duration for the target task to obtain required data; and

adding the expected start moment at which the target task is executed on each machine and the expected duration to obtain an expected completion moment at which the target task is executed on each machine, to obtain the M expected completion moments.

7. The method according to claim 6, wherein the determining an expected start moment at which the target task is executed on each of the M machines comprises:

determining an expected idle moment at which each machine completes an allocated task;

determining a moment at which all upstream tasks on which the target task depends are completed; and

for each machine, using a larger one in the expected idle moment of the current machine and the moment at which all the upstream tasks on which the target task depends are completed as an expected start moment at which the target task is executed on the current machine.

8. The method according to claim 6, wherein when the target task depends on at least one upstream task, the expected communication duration comprises maximum duration in at least one communication duration corresponding to the at least one upstream task; and

for each upstream task in the at least one upstream task, communication duration required when the target task and the current upstream task are executed on a same machine is less than communication duration required when the target task and the current upstream task are not executed on a same machine.

9. The method according to claim 8, wherein for each upstream task, communication duration required when the target task and the current upstream task are executed on a same machine is zero.

10. The method according to claim 6, wherein expected execution duration required when the target task is executed on different machines is the same.

11. The method according to claim 6, wherein expected execution duration required when the target task is executed on different machines is negatively correlated with a machine processing speed.

12. A task scheduling system, comprising:

at least one storage medium, storing at least one instruction set used to perform task scheduling; and

at least one processor, communicatively connected to the at least one storage medium, wherein when running, the at least one processor reads the at least one instruction set to perform the following operations based on an instruction of the at least one instruction set:

obtaining a target directed graph, wherein the target directed graph is a directed acyclic graph formed based on a dependency relationship between N tasks, and N is an integer greater than 1;

sequentially allocating the N tasks to M machines based on the dependency relationship in an order of executable moments from early to late, to generate a scheduling table, wherein M is an integer greater than 1, and in each time of task allocation, a to-be-allocated task is allocated to a machine that is expected to be capable of completing the task at the earliest time, wherein the executable moment is determined based on a moment at which all upstream tasks on which each task depends are completed and expected idle moments of the M machines; and

outputting the scheduling table.

13. The system according to claim 12, wherein the dependency relationship comprises that execution of a downstream task depends on an output result of an upstream task, the N tasks comprise at least one start task, and each start task depends on no upstream task.

14. The system according to claim 12, wherein in each time of task allocation, the at least one processor performs the following operations:

determining an earliest executable target task from to-be-allocated tasks;

determining M expected completion moments at which the target task is executed on the M machines; and

allocating the target task to a target machine, wherein the target machine is a machine corresponding to an earliest moment in the M expected completion moments.

15. The system according to claim 14, wherein to determine the earliest executable target task from the to-be-allocated tasks, the at least one processor performs the following operations:

determining at least one executable candidate task from the to-be-allocated tasks, wherein all upstream tasks on which each candidate task depends have been allocated; and

determining the target task from the at least one candidate task.

16. The system according to claim 15, wherein to determine the target task from the at least one candidate task, the at least one processor performs the following operations:

determining an earliest idle moment of the M machines after a previous round of task allocation, wherein the earliest idle moment is an earliest moment in M expected idle moments at which the M machines respectively complete allocated tasks;

determining a moment at which all upstream tasks on which each candidate task depends are completed;

for each candidate task, determining, as an executable moment of the current candidate task, a larger one in the earliest idle moment and the moment at which all the upstream tasks on which the current candidate task depends are completed; and

determining, from the at least one candidate task, a candidate task corresponding to an earliest moment in the executable moments as the target task.

17. The system according to claim 14, wherein to determine the M expected completion moments at which the target task is executed on the M machines, the at least one processor performs the following operations:

determining an expected start moment at which the target task is executed on each of the M machines;

determining expected duration required when the target task is executed on each machine, wherein the expected duration comprises expected execution duration and expected communication duration, and the expected communication duration comprises duration for the target task to obtain required data; when the target task depends on at least one upstream task, the expected communication duration comprises maximum duration in at least one communication duration corresponding to the at least one upstream task; and for each upstream task in the at least one upstream task, communication duration required when the target task and the current upstream task are executed on a same machine is less than communication duration required when the target task and the current upstream task are not executed on a same machine; and

adding the expected start moment at which the target task is executed on each machine and the expected duration to obtain an expected completion moment at which the target task is executed on each machine, to obtain the M expected completion moments.

18. The system according to claim 17, wherein to determine the expected start moment at which the target task is executed on each of the M machines, the at least one processor performs the following operations:

determining an expected idle moment at which each machine completes an allocated task;

determining a moment at which all upstream tasks on which the target task depends are completed; and

for each machine, using a larger one in the expected idle moment of the current machine and the moment at which all the upstream tasks on which the target task depends are completed as an

expected start moment at which the target task is executed on the current machine.

19. The system according to claim 17, wherein expected execution duration required when the target task is executed on different machines is the same or the expected execution duration is negatively correlated with a machine processing speed.

20. A computer-readable nonvolatile storage medium, wherein the computer-readable nonvolatile storage medium stores at least one instruction set, and the at least one instruction set is executed by at least one processor to implement the task scheduling method according to any one of claims 1 to 11.