METHODS AND SYSTEMS FOR DATA STORAGE AND MODIFICATION

CROSS-REFERENCE TO RELATED APPLICATION

[0001] This application claims priority to Chinese application No. 202410336828.8 filed on March 22, 2024, the entire content of which is incorporated herein by reference.

TECHNICAL FIELD

[0002] The present disclosure relates to the field of distributed storage technology, and in particular, to a method and a system for data storage and modification.

BACKGROUND

[0003] A random write to data in the local file system, such as repeated modification to a local file (e.g., a word file, an excel file, etc.) is a relatively common and simple operation. However, for the random write to data in a distributed storage system, there are more complex issues related to the consistency of the distributed data and data synchronization of the distributed storage nodes.

[0004] Existing distributed storage systems usually only support append writing to data, while the random write to data (such as a file) is usually realized by replacing or overwriting the old file with a new file, or by logically deleting the old file with an expiration or invalid mark. However, in application scenarios involving a large number of large files (e.g., video) and small files (e.g., pictures) in areas such as video security, the need for a data storage and the random write is more frequent, and the above modifications to the data are stored in a less efficient manner, the storage overhead or pressure (e.g., invalid files, fragmentation of the file's residuals) increases and is also more likely to result in the risk of the inconsistency of the data in the distributed storage. Also, the large number of small files adds pressure to manage and maintain the system.

[0005] Accordingly, a method and a system for storing and modifying data are provided to ensure the consistency and reliability of modification of files while improving the storage and management efficiency of files in a distributed storage system.

SUMMARY

[0006] One of the embodiments of the present disclosure provides a method for data modification. The method may include storing large files in one or more data nodes and, in response to receiving

a modification instruction, performing a modification process on a target file at a target modification position of the target file, the target file including the large files.

[0007] One of the embodiments of the present disclosure provides a system for data modification. The system may include a service module implemented on a client, one or more data nodes, and one or more management nodes. The one or more data nodes are configured to store large files, the one or more management nodes are configured to store metadata information of the large files; the client module is configured to: in response to receiving a modification instruction for the target file, obtain the metadata information corresponding to the target file from the one or more management nodes; read the target file from the one or more data nodes based on the metadata information; perform a pre-modification process on the target file based on the modification instruction, determine pre-modification data, and send the pre-modification data to the one or more data nodes. The one or more data nodes are configured to: cache the pre-modification data and return a pre-modification result to the service module; and obtain the target file after modification by performing a modification process on the target file based on the pre-modification data.

[0008] One of the embodiments of the present specification provides a data modification device comprising: a memory and a processor connected to the memory. The memory is used to store program instructions, and the processor is used to execute the program instructions for implementing the method for data modification.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] The present disclosure is further described in terms of exemplary embodiments. These exemplary embodiments are described in detail with reference to the drawings. The drawings are not to scale. These embodiments are non-limiting exemplary embodiments, in which like reference numerals indicate similar structures throughout the several views of the drawings, and wherein:

[0010] FIG. 1 a schematic diagram illustrating an application scenario of a system for data storage and modification according to some embodiments of the present disclosure;

[0011] FIG. 2A is a schematic diagram illustrating a system for data storage and modification according to some embodiments of the present disclosure;

[0012] FIG. 2B is a schematic diagram illustrating a directory view according to some embodiments of the present disclosure;

[0013] FIG. 3A is a flowchart illustrating an exemplary process for data modification according to some embodiments of the present disclosure;

[0014] FIG. 3B is a schematic diagram illustrating a process for data modification according to some embodiments of the present disclosure;

[0015] FIG. 4 is an flowchart illustrating an exemplary manner for pre-modification processing according to some embodiments of the present disclosure;

[0016] FIG. 5A is a flowchart illustrating an exemplary manner for writing modification data to data pieces according to some embodiments of the present disclosure;

[0017] FIG. 5B is a schematic diagram illustrating an exemplary erasure code storage solution according to some embodiments of the present disclosure;

[0018] FIG. 5C is a schematic diagram illustrating a process for writing modification data to data pieces according to some embodiments of the present disclosure;

[0019] FIG. 6 is a flowchart illustrating an exemplary manner for data aggregation according to some embodiments of the present disclosure;

[0020] FIGs. 7A and 7B are flowcharts illustrating an exemplary manner for data reading according to some embodiments of the present disclosure;

[0021] FIG. 7C is a schematic diagram illustrating a data reading inconsistency according to some embodiments of the present disclosure;

[0022] FIG. 8 is a schematic diagram illustrating a workflow of a user-state file system according to some embodiments of the present disclosure;

[0023] FIG. 9 is a schematic diagram illustrating a device for data random write according to some embodiments of the present disclosure; and

[0024] FIG. 10 is a schematic diagram illustrating a computer storage medium according to some embodiments of the present disclosure.

DETAILED DESCRIPTION

[0025] In more clearly illustrate the technical solutions of the embodiments of the present disclosure, the accompanying drawings required to be used in the description of the embodiments are briefly described below. Obviously, the accompanying drawings in the following description are only some examples or embodiments of the present disclosure, and it is possible for those skilled in the art to apply the present disclosure to other similar scenarios according to these drawings without creative labor. Unless obviously obtained from the context or the context illustrates otherwise, the same numeral in the drawings refers to the same structure or operation.

[0026] It should be understood that the terms "system," "device," "unit," and/or "module" as used herein is a way to distinguish between different components, elements, parts, sections, or assemblies at different levels. However, the words may be replaced by other expressions if other words accomplish the same purpose.

[0027] Unless the context clearly suggests an exception, the words "one," "a," "an," and/or "the" do not refer specifically to the singular, but may also include the plural. Generally, the terms "including" and "comprising" suggest only the inclusion of clearly identified operations and elements, which do not constitute an exclusive list, and the method or device may also include other operations or elements.

[0028] Flowcharts are used in the present disclosure to illustrate operations performed by the system according to embodiments of the present disclosure. It should be appreciated that the preceding or following operations are not necessarily performed in an exact sequence. Instead, the operations may be processed in reverse order or simultaneously. Also, it is possible to add other operations to these processes or remove the operation or operations from them.

[0029] FIG. 1 is a schematic diagram illustrating an application scenario of a system for data storage and modification according to some embodiments of the present disclosure.

[0030] As shown in FIG. 1, an application scenario 100 includes a processing device 110, a network 120, a terminal 130, a storage device 140, and a data collection device 150.

[0031] The processing device 110 processes data and/or information obtained from the terminal 130, the storage device 140, and the data collection device 150. For example, the processing

device 110 obtains data such as a video, an image, a sound, a text, etc. collected by the data collection device 150. As another example, the processing device 110 performs deletion, modification, etc., on the data such as the video, the image, the sound, the text, etc.

[0032] In some embodiments, the processing device 110 is a single server or a server group. In some embodiments, the processing device 110 is local or remote. The processing device 110 is directly connected to the terminal 130, he storage device 140, and the data collection device 150 to access stored or obtained information and/or data. In some embodiments, the processing device 110 is implemented on a cloud platform. Merely by way of example, the cloud platform includes a private cloud, a public cloud, a hybrid cloud, a community cloud, a distributed cloud, an internal cloud, a multi-layer cloud, etc., or any combination thereof.

[0033] In some embodiments, the processing device 110 s a distributed server group, which includes a plurality of server nodes. In some embodiments, the plurality of server nodes includes one or more management nodes and one or more data nodes. More descriptions of the one or more management nodes and the one or more data nodes may be found in FIG. 2A and the descriptions thereof.

[0034] The network 120 includes any suitable network that facilitates an exchange of information and/or data for the application scenario 100. In some embodiments, one or more components of the application scenario 100 (e.g., the terminal 130, the processing device 110, the storage device 140, or the data collection device 150) transmit the information and/or data with one or more other components of the application scenario 100 via the network 120. For example, the processing device 110 obtains video data, image data, etc. from the data collection device 150 via the network 120. In some embodiments, the network 120 is any one or more of a wired network or a wireless network. In some embodiments, the network is any of a variety of topologies, such as peer-to-peer, shared, centralized, or a combination of the topologies.

[0035] The terminal 130 includes a mobile device 130-1, a tablet 130-2, a laptop 130-3, etc., or any combination thereof. In some embodiments, the terminal 130 interacts with other components in the application scenario 100 via the network 120. In some embodiments, the terminal 130 receives the information and/or instructions input by the user and send the received information

and/or instructions to the processing device 110 via the network 120. For example, the user is able to input the instructions (e.g., checking, deleting, modification, etc.) on the data (e.g., file data such as videos, images, texts, etc.) through the terminal 130. In some embodiments, the terminal 130 installs a preset application program.

[0036] The storage device 140 stores the data and/or instructions. In some embodiments, the storage device 140 stores the data obtained from the processing device 110, the terminal 130, and/or the data collection device 150. For example, the storage device 140 stores the data (e.g., the videos, the images) obtained by the data collection device 150, etc. In some embodiments, the storage device 140 stores the data and/or instructions used by the processing device 110 to perform the exemplary methods described in the present disclosure. For example, the storage device 140 stores the instructions for the processing device 110 to perform the methods shown in the respective flowcharts. In some embodiments, the storage device 140 includes a mass storage device, a removable storage device, a volatile read-write memory, a read-only memory (ROM), etc., or any combination thereof. In some embodiments, the storage device 140 is implemented on a cloud platform. In some embodiments, the storage device 140 is a portion of the processing device 110.

[0037] The data collection device 150 is used to obtain various types of collection data. Exemplarily, the data collection device 150 includes a video collection device 150-1 (e.g., a surveillance camera device) for obtaining video data, an image collection device 150-2 (e.g., a camera, a camcorder) for obtaining image data, and a sound collection device 150-3 (e.g., a microphone) for obtaining sound data, etc. In some embodiments, the data collection device 150 sends the collection data to the processing device 110 and/or the storage device 140 via the network 120 to store and/or process the collection data.

[0038] The above description is for illustrative purposes only, and various variations may be made according to actual application scenarios. It should be noted that the application scenario 100 is provided for illustrative purposes only and is not intended to limit the scope of the present disclosure. For those skilled in the art, a variety of modifications or variations are made according to the description herein. However, these changes and modifications does not depart from the scope of the present disclosure.

[0039] FIG. 2A is a schematic diagram illustrating a system for data storage and modification according to some embodiments of the present disclosure.

[0040] As shown in FIG. 2A, the system for data storage and modification 200 includes a user end 210 and a server end 220.

[0041] The user end 210 is used to interact with a user (e.g., a system manager, a system user) and/or the server end 220, which is in a form of a terminal 130. In some embodiments, the user end 210 is used to receive an operation instruction on the data from the user. The operation instruction includes, but is not limited to, performing checking, deletion, modification, etc., on the data.

[0042] In some embodiments, the user end 210 includes a client application 211, which includes various forms of software or applications such as a browser, a client, a command-line interface (CLI) program, etc., for example, an EXE executable application under a running environment of Microsoft operating system (Microsoft Windows), a Mac App under Apple operating system (e.g., Apple Mac OS); or an app under the running environment of a tablet computer or a smartphone (e.g., iOS, Android), etc.

[0043] The client application 211 has a visual user interface (UI) for enabling an interaction with the user. For example, the client application 211 presents the user with a directory structure, a list of files, etc., as well as receives an operation instruction (e.g., addition, deletion, modification, etc.) from the user on the directory and/or files. For different operation instructions (e.g., file modifications), service requests are correspondingly sent to the server end 220, or client service interfaces (e.g., file modification service interfaces) of the server end 220 (e.g., the service module 221 of the client) are called, so that corresponding services or functions are implemented by the server end 220.

[0044] The server end 220 is used to perform storage and/or processing on the data, and the server end 220 is configured as a processing device (e.g., the processing device 140). For example, the server end 220 is implemented on a cloud platform. In some embodiments, the server end 220 includes a client service module 221, one or more management nodes 222, and one or more data nodes 223.

[0045] The client service module 221 is used to interact with the user end 210, the one or more management nodes 222, and/or the one or more data nodes 223. In some embodiments, the client service module 221 is deployed in an independent server or deployed in the one or more management nodes 222.

[0046] The client service module 221 includes a plurality of preset service interfaces. The service interface is in a form of an application programming interface (API). Exemplarily, the service interface includes, but is not limited to, a file read service interface, a file deletion service interface, a file modification service (e.g., a file append write service, a file random write service) interface, etc. In the present disclosure, the service or function corresponding to the service interface may be called as a client service (e.g., a file reading service, a file deletion service, a file modification service, etc.).

[0047] In some embodiments, the client service module 221 serves as an entrance for accessing various services (e.g., the file modification service, etc.) in the server end 220 for a user or the user end 210. For example, the user or the user end 210 accesses various services provided by the one or more management nodes 222 and/or the one or more data nodes 223 via the client service module 221.

[0048] The one or more management nodes 222 are used to manage information associated with files. The one or more management nodes 222 may be in the form of one or more servers/server nodes. As shown in FIG. 2A, the one or more management nodes 222 include a plurality of nodes such as a management node 222-1, a management node 222-2, etc.

[0049] In some embodiments, the one or more management nodes 222 include a primary management node and a backup management node. For example, the management node 222-1 serves as the primary management node and the management node 222-2 serves as the backup management node.

[0050] In some embodiments, the primary management node (e.g., the management node 222-1) is used to serve as an entry node for the one or more management nodes 222 for interacting with one or more other components (e.g., the client service module 221, the one or more data nodes 223) in the system for data storage and modification 200. For example, the client service module

221 accesses the primary management node to obtain the information associated with files (e.g., metadata information).

[0051] In some embodiments, the backup management node is used to back up the metadata of the primary management node to enable metadata fault-tolerant . For example, when the data of the primary management node is lost or abnormal, the metadata of the backup management node is used for recovering the data of the primary management node.

[0052] In some embodiments, the backup management node is used to enable a fault-tolerant processing of the primary management node. For example, the backup management node monitors the operation status of the primary management node using, for example, a heartbeat mechanism, and when the primary management node (e.g., the management node 222-1) fails, the backup management node (e.g., the management node 222-2) replaces the management node 222-1 to become a new primary management node.

[0053] In some embodiments of the present disclosure, by disposing the primary management node with one or more backup management nodes, the backup of the metadata of the management node is implemented. At the same time, a quick response to data exceptions and/or fault tolerance of the main management node is enabled, thereby ensuring the normal operation of the primary management node.

[0054] In some embodiments, the management node 222 is configured as a plurality of metadata server (MDS) for storing the metadata information. For example, the primary management node 222-1 is configured as a primary metadata server and the backup management node 222-2 is configured as a backup metadata server. In some embodiments, the metadata server and the one or more data nodes are nodes independent from each other.

[0055] The metadata refers to information related to various types of objects. The objects may be a file object, a target object, etc. The metadata includes directory metadata, file metadata, etc.

[0056] In some embodiments, the one or more management nodes 222 store the metadata based on a Key-Value (KV) data table.

[0057] In some embodiments, the one or more management node store the directory information of the files, which is used to reflect a logical organizational structure and/or a hierarchy of the files.

The directory information includes a root directory and a user directory, and in some embodiments, the directory information is generated by the user via the user end 210 when creating the root directory and the user directory. The root directory and/or the user directory store a logical file object, for example, a logical expression of the file when uploaded by the user. For more descriptions about the logical file object, please refer to elsewhere in the present disclosure (e.g., FIG. 3A).

[0058] In some embodiments, the directory information is also referred to as the directory metadata, which is stored based on a bucket table in the form of the KV data table. The directory metadata is used to manage the directory, e.g., to present a structural view of the root directory and the user directory, as well as the operations on the directory (e.g., operations such as creation, deletion, renaming, etc.) to the user.

[0059] FIG. 2B is a schematic diagram illustrating an exemplary directory view according to some embodiments of the present disclosure.

[0060] The directory view includes a root directory and a plurality of layers of sub-directories under the root directory. As shown in FIG. 2B, a directory view 230 includes a root directory 231, a sub-directory 232 is included under the root directory 231, and a sub-directory 233 is included under the sub-directory 232. In some embodiments, the user mounts the root directory 231 by default via the user end 210, and each sub-directory is a bucket created by a user. The bucket is used to represent a logical storage space used to provide the user with the ability to manage file data (e.g., videos, images, texts). The file data in the bucket is also referred to as a logical file object, and the logical file object is a logical expression of real file data in distributed storage so that the user is able to easily perform a visualized management.

[0061] In some embodiments, the management node 222 further includes a file management service interface. The file management service interface is used to provide a service for managing the directories and/or the files. For example, the service such as creating a directory, deleting a directory, uploading a file, and deleting a file, etc.

[0062] In some embodiments, the file management service interface is implemented based on a portable operating system interface (POSIX) standard or specification, so as to enable a standard

file operation (e.g., opening, reading, writing, etc., the files) on the files or the directories. In some embodiments, the file management service interface includes a file truncate operation for changing a size of the file based on the POSIX.

[0063] In some embodiments, the management node 222 also provides a network file system (NFS). The NFS service is used to provide the users with the ability to access services related to a file storage based on a standard NFS protocol. In some embodiments, the file management service interface is implemented based on a portable operating system interface (POSIX) standard or specification, so as to enable a standard file operation (e.g., opening, reading, writing, etc. the files) on the files or the directories. With the NFS service, the user is able to operate on a remote file in the same way as it does on a local file.

[0064] In some embodiments of the present disclosure, by providing the file management service through the one or more management nodes, the user is able to intuitively manage and maintain the files of the distributed storage, which improves the efficiency of the management of the files and user experience. Additionally, the file service interface implemented based on the POSIX standard improves a compatibility of the file service interface while providing the users with more services for performing standard operations on the files.

[0065] In some embodiments, the metadata includes file metadata. The file metadata is stored based on a key-value pair data table manner. In some embodiments, the file metadata includes large file metadata for storing information related to large files.

[0066] The file metadata includes static attribute information of the file, which includes, but is not limited to, a file type (e.g., a video type, an image type, a text type, etc.), a file name, a file size (e.g., a storage space occupied), a time of creation (e.g., a timestamp), etc. The metadata information also includes dynamic attribute information, which includes, but is not limited to, a historical access time (e.g., time opened or read), a historical number of accesses, a recent modification time, etc.

[0067] It should be understood that the attribute information (e.g., a static attribute, a dynamic attribute) of a file is determined based on actual requirements. Merely by way of example, for a video file, the static attribute information also includes the duration of the video, etc., and the

dynamic attribute also includes a most recently played point in time, etc. The static attribute is fixed and unchanged, and the dynamic attribute is updated in real-time.

[0068] In some embodiments, the file metadata includes label information for a file. The label information is used to label the file, which is determined based on the actual situation of the file (e.g., the content of the file, etc.). For example, for an image file, the corresponding label information of the image file includes a type of object in the image (e.g., a person, an animal, and a plant, or any combination thereof), a name, or other customized label information. More descriptions of the metadata may be found in FIG. 4 and the descriptions thereof.

[0069] In some embodiments, the metadata also includes aggregation file metadata used for storing information related to an aggregated file. More descriptions for the aggregation files please see, FIG. 6 and the descriptions thereof.

[0070] The one or more data nodes 223 are used to store real data/actual content of the file, for example, video data corresponding to the video file, image data corresponding to the image file, text data corresponding to the text file, etc.

[0071] In some embodiments, the one or more data nodes 223 are in a form of a plurality of servers/server nodes. As shown in FIG. 2A, the one or more data nodes 223 include a plurality of data nodes such as a data node 223-1, a data node 223-2, a data node 223-3, etc. In some embodiments, different data nodes are used to store data corresponding to different portions of the same file.

[0072] In some embodiments, the one or more data nodes 223 are communicatively connected to one or more other components in the system for data storage and modification 200 to implement a data interaction. For example, each data node in the one or more data nodes 223 (e.g., the data node 223-1, the data node 223-2, etc.) is communicatively connected to the client service module 221, and the one or more management nodes 222 (e.g., the primary management node 222-1). A plurality of data nodes in the one or more data nodes 223 are also communicatively connected to each other.

[0073] The data interaction includes, but is not limited to, an interaction of program instructions, data, and/or information. In some embodiments, one of the one or more management nodes 222

and the one or more data nodes 223 also includes a plurality of APIs to enable one or more other components (e.g., the client service module 221) to interact with the node.

[0074] For example, the one or more management nodes 222 (e.g., the management node 222-1, the management node 222-2, etc.) include interfaces such as a metadata obtaining service, a metadata update service, etc. For example, in response to the client service module 221 initiating a service request for obtaining metadata, the management node 222 performs an obtaining process on the metadata information and feeds back the metadata information; in response to the one or more data nodes 223 (such as the data node 223-1, the data node 223-2, etc.) initiating a service request for updating the metadata, the one or more management nodes 222 perform an updating process on the metadata, so as to realize metadata data deletion, synchronization, modification, etc.

[0075] Similarly, the one or more data nodes 223 includes interfaces such as the data reading service, the data writing service, etc., of the data, to enable the one or more data nodes 223 to respond to the service request initiated by one or more other components (e.g., the client service module 221) to perform corresponding processes (e.g., obtaining file contents, writing file contents).

[0076] In some embodiments, the server end 220 stores various types of data (e.g., the videos, the images, the texts, etc.) in a plurality of different data nodes (e.g., the data node 223-1, the data node 223-2, etc.) in a decentralized manner through the client service module 221, the one or more management nodes 222, and the one or more data nodes 223, so as to achieve a distributed storage of the data. In the present disclosure, the server end 220 is also referred to as a distributed storage system.

[0077] The distributed storage is determined based on actual requirements (e.g., business scenarios, a data volume, a count and/or an availability of the data nodes, etc.). Merely by way of example, the distributed storage includes, but is not limited to, a distributed object storage (DOS), a distributed block storage (DBS), a distributed file system (DFS), a distributed database system (DDBS), a distributed key-value storage, etc., or any one or combination thereof.

[0078] In some embodiments, the distributed storage system stores the large files in a plurality of data nodes, and in response to receiving the modification instruction, a modification processing is performed on a target file at a target modification position of the target file, the target file including

the large files. More descriptions of the large file and modification processing may be found in FIG. 3A and the descriptions thereof.

[0079] In some embodiments, the target file also includes an aggregation file, and the distributed storage system performs a file aggregation processing on the plurality of small files to generate the aggregation file. The aggregation file is processed as the large file (e.g., the modification processing on the large file). More descriptions of the file aggregation processing may be found in FIG. 6 and the descriptions thereof.

[0080] In some embodiments of the present disclosure, a reliability and a security of a massive data storage is achieved by the distributed storage of the data. In addition, by separating the management nodes from the data nodes, and separating the metadata and the real data for storage, a file storage scale of the storage system is enhanced while the management efficiency of the files is facilitated.

[0081] FIG. 3A is a flowchart illustrating an exemplary method for data modification according to some embodiments of the present disclosure.

[0082] In some embodiments, a process 300 is performed by a distributed storage system. As shown in FIG. 3A, the process 300 includes the following operations.

[0083] In 310, in response to receiving a modification instruction, pre-modification data may be determined by performing a pro-modification process on a target file through a service module implemented on a user end, and the pre-modification data may be sent to one or more data nodes.

[0084] The target file refers to a file that needs to be updated/modified and stored, which is specified by the modification instruction. It is noted that the target file before updating is data uploaded by a user or the data collected by a collection device (e.g., the data collection device 150), which includes, but is not limited to, various types of files such as videos, images, audios, and texts.

[0085] In some embodiments, the target file includes a large file. The large file refers to a file whose file size (i.e., an occupied storage space or a byte length) is greater than a storage space threshold. The storage space threshold is preset based on requirements, e.g. 1000M.

[0086] In some embodiments, the distributed storage system divides the target file into one or more data objects based on the size of the target file such that the size of each data object is within

a preset object size threshold (e.g., 200M, 500M, etc.). The data object refers to a logical expression of real data/physical data (e.g., binary data/binary files of the target file or a portion thereof) in the distributed storage system, and a plurality of data objects corresponding to the target file represents a plurality of different portions of the target file. In some embodiments, the plurality of data objects of the target file are stored at one or more management nodes in a form of logical file objects, at this time, a logical file object is expressed based on a form of metadata.

[0087] The distributed storage system operates on the target file or a portion thereof by performing operations (e.g., querying, modifying, deleting, etc.) on one or more data objects of the target file. In the following, one or more data objects of the target file is also referred to as the file objects or sub-file objects.

[0088] In some embodiments, each sub-file object of the target file is stored in a plurality of data nodes in a form of a plurality of data pieces. The distributed storage system merges the plurality of data pieces of the sub-file object to obtain the data corresponding to the sub-file object, and further merges the plurality of sub-file objects corresponding to the target file to obtain the target file. For more about the data pieces, please refer to elsewhere in the present disclosure (e.g., FIG. 4).

[0089] In some embodiments of the present disclosure, considering differences in the sizes of different target files, first, the target file is divided into a count of equal-sized data object, so as to improve the efficiency of the distributed storage and the processing uniformity of the target file by subsequent data nodes, and avoid situations where the sizes of the data pieces are too large, too small, or of uneven sizes due to differences in the sizes of different target files. At the same time, a targeted processing (e.g., modifications) of specific portions of the target file is facilitated.

[0090] The modification instruction refers to an instruction that performs an update processing on the target file. The update processing includes processing such as modifying, deleting, etc., the target file or a portion of the target file.

[0091] In some embodiments, the modification instruction includes an append write instruction. The append write instruction refers to an instruction that modifies the target file at the last index of the target file. For example, the append write instruction is adding new data at the end of the file.

[0092] In some embodiments, the modification instruction includes a random write instruction. The random write instruction refers to an instruction that modifies the target file at a random position in the target file. For example, the random write instruction is inserting the data at the random position in the target file. A positioning of the random position of the target file is achieved based on a file positioning operation (e.g., a seek method of a file I/O operation).

[0093] In some embodiments, the modification instruction (e.g., the append write instruction, the random write instruction) includes modification information including modification parameters and their corresponding modification parameter values. In some embodiments, the modification instruction (e.g., the append write instruction, the random write instruction) includes modification information, which includes modification parameters and their corresponding modification parameter values.

[0094] The modification position is expressed as a modification beginning index, or a modification offset, etc. of the modification in the target file. The modification beginning index is expressed as a character or a byte index in the target file, e.g., 1k indicates a 1k bytes shift from the begin position of the target file.

[0095] The offset is expressed as an offset length relative to a reference position (e.g., the begin position). The offset length refers to the length of the offset character or byte (e.g., 10, 30). Taking a default begin position being the reference position as an example, an offset length of 1 indicates that beginning from index 0, panning 1 character or byte along a direction of a character or byte sequence of the target file (i.e., a sequential direction) results in an offset position of 1.

[0096] The modification starting position or the offset position is used to indicate a target modification position of the target file.

[0097] The modification information corresponding to the modification instruction (e.g., the append write instruction, the random write instruction) is obtained based on manners including, but not limited to, program instructions, API requests (e.g., parameter lists of APIs), etc.

[0098] The pre-modification process refers to modifications to the target file by the service module implemented on the client based on the modification instructions, which includes I/O operations on the target file, for example, reading and/or opening the target file, writing the modification data, etc.

The pre-modification data refers to data that is obtained after the pre-modification process is completed.

[0099] In some embodiments, the pre-modification process includes performing a strip cache process on the target file at the service module implemented on the client to obtain the pre-modification data.

[0100] Combining FIG. 3B, FIG. 3B is a schematic diagram illustrating a process for data modification according to some embodiments of the present disclosure.

[0101] As shown in FIG. 3B, the client service module 221 performs an I/O operation on a target file 301 by operation 311. The target file 301 is determined based on the parameter value of a file name or a file unique id for a modification instruction (e.g., a random write instruction). The client service module 221 loads the target file 301 and writes the modification data to the target file at a target modification position of the target file. The target modification position is determined based on the parameter value corresponding to an offset position parameter of the modification instruction, and the modification data is determined based on the parameter value of the modification data parameter of the modification instruction.

[0102] The client service module 221 performs a stripe caching process on the modified target file 301 by operation 313 to obtain pre-modification data 303 corresponding to different portions of the target file 301. As shown in FIG. 3B, the pre-modification data 303 includes pre-modification data 303-1,..., pre-modification data 303-n corresponding to different portions (e.g., data pieces) of the target file 301. More descriptions of the manner of pre-modification processing may be found in FIG. 4 and the descriptions thereof.

[0103] In some embodiments, the service module implemented on the client initiates pre-modification requests to a plurality of data nodes via a network to send the pre-modification data to the corresponding data nodes respectively.

[0104] As shown in FIG. 3B, the client service module 221 initiates the pre-modification request to the data node 223-1,..., the data node 223-n based on operations 315-1,..., and 315-n, respectively, to send the pre-modification data 303-1,..., pre-modification data 303-n, respectively, to the data node 223-1,..., the data node 223-n.

[0105] In 320, the one or more data nodes may cache the pre-modification data and return a pre-modification result to the service module.

[0106] In some embodiments, each data node includes a cache device, which is in a form of a Random Access Memory (RAM), a flash memory, etc. Each data node, after receiving the pre-modification data sent by the service module implemented on the client, writes or stores the pre-modification data into the caching device, thereby obtaining cached pre-modification data. In the present disclosure, the process of the one or more data nodes writing or storing the pre-modification data to the cache device is referred to as a one-phase modification process. More descriptions of the data nodes may be found in FIG. 2A and the descriptions thereof.

[0107] Continuing to combine FIG. 3B, and taking the data node 223-1 as an example, as shown in FIG. 3B, after the data node 223-1 receives the pre-modification data 303-1 sent by the client service module 221, a buffer is created for storing the pre-modification data 303-1 in the caching device of the data node 223-1, for example, the pre-modification data buffer 322-1 shown in FIG. 3B, and write the pre-modification data 303-1 to the pre-modification data buffer 322-1 by operation 321-1 (e.g., the I/O operation). Similarly, other data nodes (e.g., the data node 223-n) write the pre-modification data 303-n to the pre-modification data buffer 322-n by operation 321-n.

[0108] The pre-modification result is used to indicate a processing status of a one-phase modification process for each of the relevant data nodes, e.g., the pre-modification result includes the processing status of the one-phase modification process for the data node 223-1,..., the processing status of the one-phase modification process for the data node 223-n.

[0109] The processing status of the first stage of modification process includes a variety of preset pre-modification status. Exemplarily, the pre-modification status includes a pre-modification success status, a pre-modification failure status, or other preset statuses.

[0110] In some embodiments, when the pre-modification status is failed (i.e., the pre-modification failure status), the pre-modification result also includes exception information for the one or more data nodes.

[0111] The exception information for the one or more data nodes also includes a hardware exception and a software exception. Merely by way of example, the hardware exception includes a

device exception (e.g., disk bad sectors), etc. The software exception includes different types of program exceptions, for example, a data input/output exception (IO exception), a memory out-of-memory exception, etc. It is understood that the exception information of the one or more data nodes is determined according to the actual situations of different data nodes, for example, the exception information also includes a network exception, etc.

[0112] The data node that fails to be pre-medicated is referred to as an exception/faulty data node or an abnormal/faulty node; the data node that succeeds to be pre-medicated is referred to as a normal data node or a normal node.

[0113] Continuing to referring to FIG. 3B, taking the data node 223-1 as an example, as shown in FIG. 3B, the data node 223-1, after completing the current one-phase modification process, sends the pre-modification result to the client service module 221 according to operation 323-1. Similarly, other data nodes (e.g., data nodes 223-n) send the pre-modification results to the client service module 221 according to operation 323-n.

[0114] In some embodiments, the service module implemented on the client respectively stores the pre-modification result corresponding to each data node (not shown in the figure). For example, the pre-modification results are stored as caches, logs, etc.

[0115] In 330, in response to that the pre-modification result satisfies a preset condition, the service module may submit a request for a two-phase modification to the one or more data nodes.

[0116] The preset condition is used to balance whether the pre-modification result feedback by each data node satisfies a condition for initiating the two-phase modification request. For example, the preset condition is whether a count of pre-modification success or failure statuses in the pre-modification results satisfy a pre-set threshold. Refer to the descriptions below for more details on the two-phase modification.

[0117] In some embodiments, the service module implemented on the client determines whether or not the preset condition is satisfied based on the pre-modification results returned by the respective data nodes.

[0118] In some embodiments, the service module implemented on the client determines, based on the pre-modification result, a count of abnormal data nodes in the one or more data nodes, and in

response to that the count of abnormal data nodes is less than or equal to a fault tolerance threshold, determines that the pre-modification result satisfies the preset condition.

[0119] For example, the service module implemented on the client determines, based on the pre-modification result of the each data node, the count of abnormal data nodes, and determines whether the count of abnormal data nodes is less than or equal to a fault tolerance threshold, and in response to that a determination result indicates that the count of abnormal data nodes is less than or equal to the fault tolerance threshold, the pre-modification result satisfies the preset condition, in response to that the determination result indicates that the count of abnormal data nodes is not less than or equal to the fault tolerance threshold, the pre-modification result does not satisfy the preset condition.

[0120] In some embodiments, in response to that the count of the abnormal data nodes is greater than the fault tolerance threshold, the client service stops modifying the target file.

[0121] In some embodiments, for the abnormal data node, the service module implemented on the client reports to the management node (e.g., a metadata server) based on the exception information fed back from the abnormal data node to make the management node to update the relevant information of the abnormal data node. For example, the management node marks the data node as an abnormal status based on the exception information corresponding to the abnormal node reported by the service module implemented on the client.

[0122] The fault tolerance threshold refers to a count of abnormal data nodes that are allowed to appear in the one or more data nodes, which reflects a fault tolerance or affordability of the distributed storage system, and the higher the value, the higher the count of abnormal data nodes that is able to be afforded, thus the greater the reliability of the distributed storage system. For example, if the fault tolerance quantity threshold is 2, when the count of abnormal data nodes is less than or equal to 2, it indicates that the pre-modification result satisfies the pre-set condition, so that the relevant processing continues to be performed, for example, the service module implemented on the client submits the two-phase modification request to the one or more data nodes.

[0123] In some embodiments, the fault tolerance threshold is determined based on the count of the parity blocks. More descriptions of the parity blocks may be found in FIG. 5A and the descriptions thereof.

[0124] In some embodiments, in response to that the pre-modification result satisfies the preset condition, the service module implemented on the client sends the request for the two-phase modification (also referred to as second stage modification request) to each normal data node separately to perform operation 340.

[0125] In some embodiments, in response to determining the pre-modification result not satisfying a preset condition, the service module implemented on the client sends a rollback request to the data node. The rollback request is used to instruct the data node to perform a pre-modification rollback process. The pre-modification rollback process includes, but is not limited to, the data node deleting the pre-modification data stored in a cache region to release cache resources.

[0126] In some embodiments, the rollback request further includes instructing the data node to perform a rollback transaction of the two-phase modification request. It should be noted that the service module implemented on the client initiates the pre-modification request to the data node for several times based on the actual situation of the pre-modification data (e.g., a count of strips), that is, before the current pre-modification request, a portion of the pre-modification data has already been written to a physical disk, and at this time, the rollback request needs to instruct the data node to perform the rollback transaction.

[0127] In 340, the one or more data nodes may write the pre-modification data to the target file based on the two-phase modification request.

[0128] The two-phase modification request is used to instruct the one or more data nodes to perform a two-phase modification process. In some embodiments, the two-phase modification process is configured to support a process of the transaction process. The transaction includes a commit transaction and a rollback transaction.

[0129] In some embodiments, the two-phase modification request includes a transaction committing request and a rollback transaction request. The transaction committing request is configured to instruct the data node to write the pre-modification data to the target file; and the

rollback transaction request is configured to instruct the data node to restore modified target file to the target file before modification.

[0130] The commit transaction refers to a process that instructs a data node in the two-phase modification to perform a pre-modification data commitment process to write the pre-modification data to a physical storage space (e.g., a disk) to achieve the modification of the real data of the target file.

[0131] As shown in FIG. 3B, in response to the two-phase modification request, the data node 223-1,..., the data node 223-n respectively perform the commitment process of the pre-modification data according to operation 324-1,..., operation 324-n, respectively, so as to write the pre-modification data to the data block $Block_0$,..., data block $Block_n$, and return relevant information (e.g., the success or failure status, the exception information, etc.) of the commitment process to the client service module 221 according to operation 325-1,..., operation 325-n. Taking the data node 223-1 as an example, the data node 223-1 writes the data corresponding to pre-modification data buffer 303-1 (i.e., the pre-modification data 303-1) to a data block $Block_0$, and returns a status of the writing and a length of the data in the data block $Block_0$ to the client service module 221. When the write successes, the status of a successful commit is returned, and when the write fails, the status of a failed commit and/or a failed exception message, etc., is returned. The data block $Block_0$ is a physical data block in the data node, whose correspondence with the pre-modification data 303-1 is determined based on the metadata server. For more descriptions of the correspondence between the physical data block and the pre-modification data, please refer to elsewhere in the present disclosure (e.g., FIG. 5A).

[0132] In some embodiments, for a situation where some data nodes submit successfully and some data nodes fail to submit, the service module implemented on the client counts the submission results returned by the data nodes, e.g., determines whether the count of failed submissions is less than a fault tolerance threshold, so as to determine whether the count of failure is within a fault tolerance capacity. And combined with of the data node for a cyclic redundancy check (CRC) checking mechanism, etc., of the committed data blocks, the correctness of the committed data blocks are ensured. The service module implemented on the client can also reports the block

corresponding to the data node that failed to commit (e.g., the aforementioned data block $Block_0$) to the management node to mark the file status, and at the same time, if the target file is in a recoverable state (within the fault-tolerance capacity), the file is intelligently recovered. For example, based on the data corresponding to the data block that is committed successfully, the data block that failed to be committed is reconstructed according to an erasure code (EC) algorithm.

[0133] The rollback transaction is a two-phase modification that instructs each data node to perform a data roll back to undo the modifications to the target file so that the target file is restored to its pre-modification state.

[0134] It should be understood that different data nodes perform the commit transactions independently and in parallel. When one or more data nodes commit transaction exceptions (e.g., failure to write in the sliced data), while other data nodes have already committed successfully, a data inconsistency occurs, and at this time, the service module implemented on the client makes a decision on whether to initiate the rollback transaction request based on the process of the commit transactions performed by each data node. For example, when the data node has a commit transaction exception greater than the fault tolerance threshold, the service module implemented on the client sends the rollback transaction request to the data node that committed the transaction successfully.

[0135] In some embodiments of the present disclosure, considering the independence and concurrency of the commit process of the pre-modification data by different data nodes, a distributed transaction processing mechanism is introduced through a design of the two-phase modification process, so that the target file or a portion thereof is modified in a count of different data nodes in the process of modification, the commit or the rollback transactions of the plurality of data nodes are executed uniformly, thus ensuring a data consistency of the target file. Especially when the exceptions occur in the plurality of data nodes and need to be rolled back, the plurality of data nodes perform the rollback operation in a unified way, thus ensuring that the target file is restored to its pre-modification state as a whole.

[0136] In some embodiments, the service module implemented on the client sending the pre-modification data to the data node based on operation 310, and the service module implemented on

the client committing the two-phase modification request to the data node based on operation 330, are carried out in batches. For example, for a plurality of strips obtained after the target file is striped, the service module implemented on the client constructs the plurality of strips into a stripe group, and uses the pre-modification data corresponding to the stripe group as the pre-modification data processed according to the operation 310. Compared to the processing performed strip by strip, the processing of the plurality of strips in batches reduces the IO operations of the subsequent data nodes. More descriptions of the stripe group may be found in FIG. 5A and the descriptions thereof.

[0137] In some embodiments, in response to that after the one or more data nodes write the pre-modification data to the target file based on the two-phase modification request, the one or more data nodes perform the CRC on the data block corresponding to the target file to obtain a data block status. When the data block status is abnormal, the one or more data nodes send the corresponding data block information to the one or more management nodes to enable the one or more management nodes to perform a file status update and/or a file data recovery on the target file.

[0138] The file status update of the target file by the one or more management nodes includes updating the metadata corresponding to the target file, for example, marking the data piece or the data block information corresponding to the abnormal data block as abnormal. The file data recovery of the target file by the one or more management nodes is a reconstruction of the data corresponding to the abnormal data block according to the EC algorithm to recover the data of the data block.

[0139] In some embodiments of the present disclosure, by first performing the pre-modification process on the target file, and then carrying out the two-phase modification process, a potential abnormal condition is able to be discovered in a timely manner in the pre-modification phase, so as to improve a reliability of the two-phase modification process. In addition, when an exception of the data node that prevents the modification of the target file as a whole is detected, a strategy is adjusted in a timely manner (e.g., enabling a backup management node, etc.) or to give up the modification. In this way, data consistency and security are ensured, which reduces or avoids the

consumption of manpower and material resources brought about by the data recovery after the target file is written.

[0140] FIG. 4 is a flowchart illustrating an exemplary process for pre-modification according to some embodiments of the present disclosure.

[0141] In some embodiments, a process 400 is performed by a distributed storage system. As shown in FIG. 4, the process 400 includes following operations.

[0142] In 410, metadata information may be obtained from one or more management nodes.

[0143] In some embodiments, a service module implemented on a client obtains the metadata information of a target file from one or more management nodes (e.g., a primary management node or a metadata server). More descriptions of the management node, the primary management node, and the metadata serve may be found in FIG. 2A and the descriptions thereof.

[0144] In some embodiments, the metadata is stored in the one or more management nodes based on a key-value storage model.

[0145] The key-value storage model includes, but is not limited to, a key-value data table, a configuration file with a key-value structure (e.g., a JSON configuration file), etc. The key is expressed as the index of a value for retrieving or obtaining the value, which is a unique identifier set based on a preset rule. The value is expressed as information corresponding to the key, which is set according to actual needs, for example, the value is one or a combination of various information including, but not limited to, the information of a file object (such as a distribution of data pieces), an Internet protocol (IP) address of a data node, and attribute information of a file (such as a size, a creation time, etc.).

[0146] In some embodiments, the metadata includes large file metadata. The large file metadata is stored in a form of the key-value data table (also known as a large file table), which is used to store information (e.g., the attribute information) related to large files.

[0147] In some embodiments, the large file metadata includes information of one or more file objects corresponding to the large files, e.g., the attribute information for each file object, information about a distribution of a plurality of data pieces (e.g., a data block block) for each file object (e.g., the distribution of one or more data nodes, storage positions of the data blocks, etc.), etc.

[0148] In some embodiments, the metadata includes aggregation file metadata. The distributed storage system (e.g., one or more management nodes) stores mapping relationships between aggregation files and a plurality of smaller files based on the key-value storage model. The mapping relationship is expressed as the aggregation file metadata. In some embodiments, the aggregation file metadata is stored in the form of the key-value data table (also known as an aggregation file table).

[0149] The small file refers to a file (e.g., a small image, a small text file, etc.) that occupies a storage space whose size is less than a preset threshold (e.g., 128K) , and the physical data or real data (e.g., original image data of the small image, text data of the small text file) corresponding to the small file is stored in a certain data node. In some embodiments, the distributed storage system aggregates a plurality of small files based on the size scales of the small files, generates the aggregation file, and distributes the aggregation file in the form of large files. More descriptions of the aggregation file may be found in FIG. 6 and the descriptions thereof.

[0150] The aggregation file metadata is used to reflect a logical correspondence of the aggregation file with the plurality of smaller files. For example, the aggregation file is generated by performing an aggregation processing on the plurality of small files, i.e., the aggregation file corresponds to the plurality of small files. In some embodiments, a key of the aggregation file metadata is set to the identifier of the aggregation file, and a value is set to small file collection information corresponding to the plurality of small files, so as to logically generate the mapping relationship between the aggregated file and the plurality of small files.

[0151] In some embodiments, the small file collection information in the aggregation file metadata includes small file metadata corresponding to each small file. The small file metadata corresponding to each small file is also set based on the structure of the key-value, which includes attribute information of the small file (e.g., a name, a creation date, etc.).

[0152] The small file metadata includes a logical offset of the small file in the aggregation file. The logical offset is used to indicate a beginning index of the small file in the aggregate file. Different small files correspond to different logical offsets. The small file metadata includes the size of the small file (e.g., a byte length). Understandably, for each small file written to the aggregation

file, through the logical offset and the size of the small file in the aggregation file, the small file object corresponding to the small file and real data corresponding to the small file object are obtained.

[0153] In some embodiments of the present disclosure, by storing the aggregation file metadata at the one or more management nodes, management of the aggregation file and the corresponding great count of small files is facilitated, which improves the retrieval and management efficiency of the distributed storage system for the small files.

[0154] In some embodiments, each small file in the small file collection corresponding to the aggregated file is stored based on a small file table. The small file table is in the form of the key-value data table, which is used to store the attribute information (e.g., the creation date, the size, etc.) corresponding to each small file. For example, for the each small file, the Key is a small file identifier, and the value includes the attribute information for the small file.

[0155] In some embodiments, the small file collection information of the aggregation file metadata includes a small file identifier (i.e., the key of the small file in the small file table) corresponding to each small file and the logical offset of the small file. Based on this, the mapping relationship between the aggregation file and the corresponding plurality of smaller files is implemented.

[0156] In some embodiments of the present disclosure, by storing the metadata of the small files through the small file table, a decoupling with the aggregation file is implemented, which is conducive to improving the efficiency of the management and maintenance of massive small files. Meanwhile, through the mapping relationship between the aggregation file and the small files, a quick retrieval of the corresponding small files in the aggregated file is implemented.

[0157] In some embodiments, the distributed storage system performs an update processing and/or a synchronization processing of the aggregated file metadata. More descriptions may be found in FIG. 2A and the descriptions thereof.

[0158] In some embodiments, in response to receiving a modification instruction of a user on the target file, the service module implemented on the client obtains corresponding metadata information from the one or more management nodes. For example, for a large file, the service module implemented on the client obtains the large file metadata corresponding to the large file from the one or more management nodes (e.g., the primary management node); for an aggregation file, the

service module implemented on the client obtains the aggregation file metadata corresponding to the aggregated file from the one or more management nodes.

[0159] In 420, the target file may be read from the one or more data nodes based on the metadata information.

[0160] In some embodiments, the service module implemented on the client obtains physical data (i.e., data corresponding to a physical data block or the data piece) corresponding to the target file from the one or more data nodes based on the metadata information of the target file to obtain the target file and perform a file IO operation (e.g., open) to open the target file.

[0161] Exemplarily, the service module implemented on the client obtains the file object corresponding to the target file based on the key corresponding to the target file (e.g., a target file identifier), obtains the data pieces stored in the one or more data nodes based on the metadata (e.g., the distribution of a plurality of physical data blocks or data pieces corresponding thereto) of the file object, and opens the target file by reading the data corresponding to the plurality of data pieces. It should be noted that the service module implemented on the client opening the target file refers to loading the data of the target file into a cache (e.g., a memory of the service module implemented on the client) to obtain the corresponding cache data of the target file.

[0162] In some embodiments, the service module implemented on the client determines, based on the offset corresponding to the modification instruction, a target file object in the plurality of file objects corresponding to the target file, and open, based on the metadata information of the target file object, the file corresponding to the target file object via the one or more data nodes. It is understood that the target file object is a portion of the target file, and modifying the file corresponding to the target file object is equivalent to modifying the target file.

[0163] In 430, a target modification position of the target file may be positioned based on the modification instruction including the offset volume of the target file and a length of modification data, and the target modification position may be determined based on the offset volume.

[0164] In some embodiments, the modification instruction corresponding to the target file includes one or more modification operation information. Each modification operation information includes the offset volume of the target file and a length of the modification data. Exemplarily, the target file

has one or more modifications (e.g., the modifications made by the user in several different positions), and the one or more modifications correspond to a different beginning position and different modification data (e.g., the content and the corresponding byte length).

[0165] The offset volume is used to indicate a count of characters or bytes to be offset relative to a reference position, such as the beginning position of the target file. A positioning processing refers to a process of translating a file pointer from the reference position (e.g., translating a count of characters or bytes corresponding to the offset volume) based on the offset volume, so that the file pointer moves to the target modification position corresponding to the current modification processing, and the modification processing (e.g., an insertion of new data, etc.) is then be performed at the target modification position. The service module implemented on the client realizes the positioning processing of the target file by calling a positioning function (e.g. a Seek function).

[0166] In some embodiments, the service module implemented on the client determines, based on the offset volume of the target file corresponding to the modification instruction, a target sub-file object in the plurality of sub-file objects corresponding to the target file, opens, based on the metadata information of the target sub-file object, the file data corresponding to the target sub-file object through the one or more data nodes, and positions the target modification position corresponding to the target sub-file object based on the modification instruction.

[0167] It is understood that the target sub-file object is a portion of the target file, and the modification of the file data corresponding to the target sub-file object is equivalent to the modification of the target file. In some embodiments of the present disclosure, the target sub-file object to be modified for the target file is determined based on the offset volume of the modification instruction, and a replacing or overwriting processing is performed on the target sub-file object. In this way, random write processing on the target file is realized, and at the same time, the random write processing on the target sub-file object helps to reduce resource consumption brought about by loading the entire target file.

[0168] In 440, the modification data may be written to a plurality of data pieces corresponding to the target file the pre-modification data may be determined.

[0169] The pre-modification data refers to cache data corresponding to the modification data after the service module implemented on the client writes the modification data to the target file. Merely by way of example, the pre-modification data is the cache data obtained by the service module implemented on the client after inserting the modification data at the target modification position of the target file.

[0170] In some embodiments, the service module implemented on the client loads the target file into the cache, and writes the modification data into the cache data based on the cache data corresponding to the target file and the modification instruction corresponding to the target file (e.g., the offset volume and the length of the modification data) to obtain the pre-modification data.

[0171] In some embodiments, the service module implemented on the client performs a striping processing on the cache data corresponding to the target file to obtain the plurality of data pieces corresponding to the target file, then, writes the modification data into one or more data pieces based on the modification instruction, and determines the pre-modification data.

[0172] It should be noted that the pre-modification data here is the cache data stored in the service module implemented on the client, and the plurality of data pieces are used for subsequent distribution to the plurality of data nodes. More descriptions of the striping, data slicing, and determining the pre-modification data may be found in FIG. 5A and the descriptions thereof.

[0173] In some embodiments of the present disclosure, the metadata is provided by the metadata server, the real data is provided by the one or more data nodes, and the pre-modification data is determined in the service module implemented on the client. In this way, a separation of a signaling flow and a data flow is realized in the modification processing of the target file, thereby making the distributed storage system have a more balanced load.

[0174] FIG. 5A is a flowchart illustrating an exemplary process for writing modification data to data pieces according to some embodiments of the present disclosure.

[0175] In some embodiments, a process 500 is performed by a distributed storage system. As shown in FIG. 5A, process 500 includes the following operations.

[0176] In 510, the modification data may be modified in a strip cache corresponding to a target file to determine strip cache data.

[0177] The stripe cache is a buffer (e.g., a memory) in a service module implemented on the client that is used to store striped data. The striped data refers to data before modification and after the data is striped, and the data before modification is a target file or a portion of the target file, e.g., data combined by a plurality of data pieces corresponding to the target file or a portion of the target file read from a plurality of data nodes.

[0178] A strip processing refers to generating a plurality of strips (stripe) of data corresponding to the target file before modification. Each strip includes a plurality of cells or blocks, and the sizes of the plurality of cells or blocks are the same (e.g., 1M, 512K, etc.) or different. A portion of the plurality of strips form a strip group.

[0179] In some embodiments, the strip process is implemented based on a preset erasure code (EC) algorithm. The EC algorithm includes, but is not limited to, an array code, a reed-solomon code (RSC) algorithm, a low density parity check code (LDPC) algorithm, and other algorithms.

[0180] In some embodiments, the service module implemented on the client determines a EC storage scheme through a preset EC algorithm, such as the RSC algorithm. The EC storage scheme refers to a way in which the striped data is stored, which is related to EC parameters.

[0181] The EC parameters include a count k of content data blocks and a count m of parity data blocks, which are determined based on a count n of the data nodes participating in a distributed storage. For example, the EC parameters satisfy n=k+m.

[0182] The EC parameters also include a count of strips, which is determined based on a size of the strip data. The EC parameters also include a manner of a block layout, e.g., a continuous layout, a striped layout, etc.

[0183] In the following, an EC storage scheme is illustrated in terms of the striped layout, and the EC storage scheme is denoted as RS(k, m).

[0184] In some embodiments, the EC storage scheme RS(k, m) includes, for a plurality of blocks in one strip, a plurality of (e.g., k counts of) content data blocks and a plurality of (e.g., m counts of) parity data blocks.

[0185] The content data blocks are used to store content data corresponding to the target file (e.g., original data before modification or data after modification), and the parity data block is used to store

coded data generated from the data of the plurality of content data blocks. The data corresponding to the k (e.g., 3) counts of content data blocks and m (e.g., 2) counts of parity data blocks are stored in physical blocks of the n (e.g., 5) counts of data nodes, respectively.

[0186] Merely by way of example, the count of data nodes n is the count of the plurality of data nodes or a portion of the plurality of data nodes in the distributed storage system, e.g., the count of the data nodes in the actual distributed storage system is greater than n, and the data nodes other than the n count of data nodes are used as backup data nodes.

[0187] FIG. 5B is a schematic diagram illustrating an exemplary EC storage scheme according to some embodiments of the present disclosure.

[0188] As shown in FIG. 5B, original data 501 of a target file is data before modification, which includes ABCDEFG..., etc. A~G each denotes a portion of the data or content of the original data 501, for example, 1M bytes in length. In the following, A~G are referred to as data section A, data section B... of the target file, respectively.

[0189] In some embodiments, a service module implemented on the client generates strip data 512 corresponding to the EC storage scheme (e.g., RS(3, 2)) based on the original data 501 by performing a striping initialization processing through a preset EC algorithm (e.g., the RS algorithm) according to operation 511. The strip data 512 is stored in a cache including strip T_1, strip T_2, strip T_3, etc., and other strips, and each stripe includes a plurality of units or blocks. Before the modification, the target file is distributedly stored in a plurality of data nodes, and therefore, each block and the data thereof in the strip data 512 is in an one-to-one correspondence with a physical data block of each data node, and the stored data therein at the time of the stripe initialization. At the same time, the service module implemented on the client determines, based on the metadata of the target file, relevant information about the physical data block (e.g., an IP address of the data node to which the physical data block belongs, a disk address, a byte length of the data, etc.), to enable the service module implemented on the client to subsequently send the data corresponding to the block in the strip to the corresponding data node.

[0190] For example, for the EC code scheme RS (3, 2) corresponding to parameters k = 3, m = 2, n = 5, each strip (e.g., strip T_1) consists of three content data blocks and two parity data blocks in a

horizontal direction. The size of each of these content data blocks is preset (e.g. 1M bytes). In some embodiments, considering the uncertainty of the size of the modification data when a user modifies the target file, the size of the each content data block is able to be adjusted according to an actual situation, and it is understandable that the each content data block corresponds to a buffer in the cache, and the service module implemented on the client may be configured to adjust the size of the buffer to avoid an out of memory exception when writing the data in the buffer.

[0191] As shown in FIG. 5B, the strip T_1 includes (not shown in the figure) a total of 5 units or blocks, namely, T_{11}, T_{12}, T_{13}, T_{14}, and T_{15}; the strip T_2 includes (not shown in the figure) a total of 5 units or blocks, namely, T_{21}, T_{22}, T_{23}, T_{24}, and T_{25}, and so on.

[0192] The strip data 512 forms a block matrix of r rows (e.g., 3 rows) and n columns (e.g., 5 columns) as shown in FIG. 5B. The count of rows is determined based on the size of the original data 501, the count of the columns is determined based on a sum of the EC parameters k and m (e.g., 5 columns).

[0193] It should be noted that the strip data 512 shown in FIG. 5B is the cache data obtained when the service module implemented on the client performs a strip cache processing on the original data 501 before modification, the original data 501 is the target file or a portion thereof (e.g., one of a plurality of file objects corresponding to the target file), and the original data 501 is obtained by combining a data piece D_1, a data piece D_2, a data piece D_3, a data piece D_4, a data piece D_5 corresponding to the data node 223-1, the data node 223-2, the data node 223-3, the data node 223-4, the data node 223-5. That is, the data (e.g., Data A,..., Data G) and parity block data (e.g., Code1,..., Code 6, etc.) corresponding to the content data blocks are the data stored in each data node before the modification.

[0194] In some embodiments, the service module implemented on the client writes the modification data into the strip data based on a modification instruction to obtain strip cache data.

[0195] Combining FIG. 5C, FIG. 5C is a schematic diagram illustrating a process for writing modification data to data pieces according to some embodiments of the present disclosure.

[0196] The process in FIG. 5C is performed on a basis of FIG. 5B, and data 502 is data after the original data 501 is modified. Merely by way of example, a user performs a modification operation

at a target modification position of a target file, so that data section A and/or data section F in the original data 501 is updated. The target modification position is a random index in the target file. For example, the random index is a position in the target file with an index of 0, or a byte index position of 1K, 1M, etc. As shown in FIG. 5C, data section A' and data section F' in the data 502 are updated data section A and/or data section F. The modification operation by the user generates a corresponding modification instruction including an offset volume of the modification data and a length of the modification data, and the offset volume is determined based on the target modification position.

[0197] As shown in FIG. 5C, the service module implemented on the client determines a target content data block (e.g., a block corresponding to data A, a block corresponding to data F) in the strip data 512 based on the offset volume corresponding to the modification instruction, and writes the modification data into the target content data block to obtain modified strip cache data 513.

[0198] In 520, the parity block data may be determined based on the strip cache data.

[0199] The parity block data refers to encoded data generated based on the data corresponding to the plurality of content data blocks according to a preset EC algorithm. The parity data block is also called a redundant data block.

[0200] In some embodiments, a count of the parity data blocks is used to reflect a fault tolerance of the distributed storage system. The fault tolerance refers to an ability of the distributed storage system to rebuild or recover the original data based on remaining data pieces when some data pieces in the plurality of data pieces corresponding to the target file are abnormal (e.g., a data loss, a data corruption, etc.).

[0201] For example, when the count of the exception data nodes (which are denoted as “e”) is less than or equal to the count of the parity data blocks (which may be denoted as “m”), the data block data (e.g., the content data block and/or the parity data block data) corresponding to the remaining data nodes (whose count may be denoted as “n-e”) are reconstructed according to the EC algorithm to obtain the original data.

[0202] In some embodiments, the service module implemented on the client generates, based on the data of the content data block in each strip, the parity block data corresponding to the strip according to the preset EC algorithm.

[0203] Combining FIG. 5C, for each strip of the strip cache data 513 (e.g., the strip T_1, etc.), the service module implemented on the client generates the parity block data (e.g., code1' and code2') based on the data of the content data block (e.g., data A', data B, and data C) according to the preset EC algorithm. When the data of any data block in a certain strip is modified (e.g., the data A is modified to data A'), the data of the parity block in that strip also needs to be updated, e.g., code1' is different from code1, and code2' is different from code2.

[0204] In 530, the modification data and the parity block data may be written to the plurality of data pieces.

[0205] A data piece is used to express a portion of the data associated with the target file.

[0206] In some embodiments, one data piece corresponding to the target file is generated based on one or more strip pieces. Each unit or block in a stripe is expressed as one stripe piece. For example, the stripe pieces include stripe pieces corresponding to the content data blocks, and stripe slices corresponding to the parity data block. The plurality of strip pieces of the same strip are referred to as a group of strip pieces. As an example, as shown in FIG. 5C, in strip T_1, each of the content data block A', the content data block B, or the content data block C is one strip piece; each of the parity data block code 1' and the parity data block code 2' is one strip piece, and a plurality of strips of strip T_1 are a group of strip pieces (referred to as a strip piece group T_1). Similarly, a plurality of strip pieces of strip T_2 are another group of strip pieces (referred to as a strip split group T_2).

[0207] A plurality of consecutive blocks with the same order in the plurality of strips (i.e., a plurality of blocks in the same column in the block matrix) are also expressed as one strip piece. As an example, as shown in FIG. 5C, in column 1, the content data block A' of the strip T_1, the content data block D of the strip T_2, and the content data block G of the strip T_3 are constructed as one strip piece. Continuously, in column 5, the parity data block code 2' of the strip T_1, the parity data block code 4' of the strip T_2, and the parity data block code 6' of the strip T_3 are constructed as one strip

piece. The strip piece with the plurality of columns (e.g., columns 1-5) corresponding to the plurality of strips are referred to as a batch of strip pieces. Understandably, a batch of stripe pieces includes a plurality of groups of stripe pieces.

[0208] It should be understood that the strip piece is used to generate a data piece or a portion of the data piece of the target file (e.g., the data piece D_1, the data piece D2, etc., of FIG. 5C).

[0209] The process of the service module implemented on the client writing the modification data and the parity block data to the plurality of data pieces refers to a process of writing the modification data and the parity block data to each strip piece.

[0210] In some embodiments, the service module implemented on the client obtains the pre-modification data corresponding to each data piece based on the plurality of data pieces. As shown in FIG. 5 C, the service module implemented on the client obtains, based on data piece D_1, data piece D_2, data piece D_3, data piece D_4, data piece D_5, corresponding pre-modification data-1, pre-modification data-2, pre-modification data-3, pre-modification data-4, pre-modification data-5.

[0211] In some embodiments, the service module implemented on the client further sends the plurality of pre-modification data to the corresponding data nodes (e.g., the data node 223-1, the data node 223-2,..., the data node 223-5 as shown in FIG. 5C) to perform operation 320. More descriptions of operation 320, please refer to FIG. 3A and descriptions thereof.

[0212] In some embodiments, the service module implemented on the client sends the pre-modification data generated by a group of strip pieces to the data nodes separately.

[0213] In some embodiments, the service module implemented on the client also sends the pre-modification data generated by the plurality of groups of strip pieces (i.e., a batch of the strip pieces) to the data nodes. By performing a unified processing through the plurality of groups of stripe pieces, I/O operations of the subsequent data nodes are reduced, which is conducive to reducing a load on the data nodes.

[0214] In some embodiments of the present disclosure, using the EC technology, the fault tolerance of the data modification is improved, and the reliability of the distributed storage system for the data storage is improved.

[0215] FIG. 6 is a flowchart illustrating an exemplary process for data aggregation according to some embodiments of the present disclosure.

[0216] In some embodiments, a process 600 is performed by a distributed storage system. As shown in FIG. 6, the process 600 includes following operations.

[0217] In 610, attribute information of the plurality of small files may be obtained.

[0218] The plurality of small files need to be stored in the distributed storage system, which include, but are not limited to, various types of files or data such as videos, images, text, etc. In some embodiments, the small files (e.g., images, etc.) are collected by a collection device (e.g., the collection device 150). More descriptions of the small files may be found in FIG. 4 and the descriptions thereof.

[0219] The attribute information of a small file includes, but is not limited to, information such as a file type (e.g., the image), a file name, a file size, and a creation time, etc.

[0220] In 620, target small files may be determined from the plurality of small files based on the attribute information.

[0221] The target small files are small files that require aggregation. For example, when the service module implemented on the client receives the plurality of small files uploaded, the service module implemented on the client first determines a count of the small files, and when the count of the small files reaches a preset data threshold, the small files are taken as the target small files.

[0222] In some embodiments, the distributed storage system determines, based on the attribute information of the small files, a correlation degree between the small files, and, determines, based on the correlation, the target small files from the small files.

[0223] The correlation degree between the small files are used to reflect a degree of a correlation relationship between the files. The correlation degree has a variety of expressions such as a value in an [0,1], and the greater the value, the higher the correlation degree between the plurality of small files. In some embodiments, the distributed storage system determines the correlation degree between one of the small files and another small file, and determines the plurality of small files with high correlations as the target small files. Merely by way of example, the plurality of small files originating from the same collection device are considered to have a high correlation degree.

[0224] In some embodiments, the distributed storage system determines a similarity degree between the plurality of small files based on the attribute information and labeling information of each of the plurality of small files; and determines the correlation degree between the plurality of small files based on the similarity degree. Labeling information is determined according to a real situation. For example, the labeling information includes one or more data nodes, a directory, or a storage space (bucket) in which a label is stored.

[0225] The similarity degree is used to indicate a degree of similarity between any two files. The greater the similarity degree, the higher the correlation degree. The similarity degree is a value in an interval [0,1], and the greater the value, the higher the similarity degree. For example, when a small file is of an image type, if another small file is of a text type, the similarity degree between the two small files is set to 0, which indicates that the two small files are not correlated with each other; if another small file is of the image type as well, the similarity degree between the two small files is set to 0.5, indicating that there is a certain correlation degree of between the two small files.

[0226] In some embodiments, the labeling information also includes an access frequency of the small files. The access frequency is used to reflect how often the small files are read and/or updated. For example, the higher the count of times the small files are read in a preset period of time, the greater the access frequency.

[0227] In some embodiments, the labeling information is related to the contents of the small files, which is used to further determine the similarity degree between the plurality of small files to adjust the correlation degree between the plurality of small files.

[0228] The labeling information of a small file is set according to the content of the small file and the labeling information is used to indicate categorized information related to the contents of the small file. For example, for an image file whose content is about a dog, the labeling information is set to the dog, a pet dog, an Alaskan dog, etc.; for a text file whose content is about automobile repair, the labeling information is set to an automobile, a repair, etc. It is appreciated that the labeling information is a combination of one or more of the categorical information related to the contents of the small files. When the plurality of small files of the same file type have the same or similar labeling information, the higher their similarity degree, the higher their correlation degree.

[0229] In some embodiments, the distributed storage system screens, based on the file type and content categorization information of each small file, from the plurality of small files, the small files with the similarity degree or the correlation degree higher than a threshold (e.g., 0.7) as the target small files.

[0230] In some embodiments of the present disclosure, by considering the attribute information and the labeling information of different files, the target small files with high correlation degree are more accurately screened, thereby making the subsequently generated aggregation files more targeted.

[0231] In 630, the aggregation file may be generated by performing file aggregation on the target small files.

[0232] The file aggregation process refers to a process of combining the plurality of small target files to generate a large file, and the generated large file also is referred to as the aggregation file.

[0233] In some embodiments, the distributed storage system writes the data (e.g., a byte stream, a character stream, etc.) corresponding to the plurality of target small files to a single file separately by means of append write through a file I/O operation to generate the aggregation file.

[0234] Merely by way of example, the service module implemented on the client creates a file object (e.g., an image object, a text file object) and writes the data corresponding to p (e.g., 10) target small files (e.g., small images, small text files) to the file object based on a plurality of rounds (p rounds) of write processing, respectively. The start position of the write in the first round of write processing is 0, and based on the size of the target small files of the round, the starting position of the next round of write processing is determined. After the p rounds of write processing are completed, the aggregation file corresponding to the p target small files is obtained.

[0235] In some embodiments, in each round of write processing, the service module implemented on the client takes the start position corresponding to the target small file as a logical offset of the target small file in the aggregation file, and further updates the logical offset and the size of the target small file into the aggregation file metadata corresponding to the aggregation file in the one or more management nodes. After the plurality of rounds of writes are completed, the aggregation file that combines the plurality of target small files is obtained, and the aggregate file metadata

corresponding to the aggregation file is obtained. More descriptions of the aggregation file metadata may be found in FIG. 4 and the descriptions thereof.

[0236] In some embodiments, the aggregation files are distributedly stored as the large files. In some embodiments, the service module implemented on the client divides the aggregation file into a plurality of portions based on the size of the aggregation file, and each portion corresponds to a single file object. The service module implemented on the client performs the strip process on the data corresponding to the aggregation file or a portion thereof according to a preset EC algorithm to obtain strip cache data corresponding to the aggregation file and, based on operations 520 and 530, a plurality of data pieces corresponding to the aggregation file or a portion thereof is determined. The plurality of data pieces are distributed to a plurality of different data nodes to realize the distributed storage of the aggregation file into the plurality of data nodes. It is noted that when the aggregation file is distributedly stored as a large file, the service module implemented on the client also creates large file metadata corresponding to the aggregation file and stores the large file metadata in a large file table. More descriptions of the large file, the large file metadata, and the large file table may be found in FIG. 4 and the descriptions thereof.

[0237] In some embodiments, the one or more management nodes determine, based on the aggregation file metadata corresponding to a plurality of historical aggregation files, historical small files corresponding to the plurality of historical aggregation files; clustering the historical small files based on the attribute information of the historical small files to determine correlated small files; and update the plurality of historical aggregation files based on the correlated small files.

[0238] The historical aggregation file refers to an aggregated file stored in the distributed storage system for a duration greater than a preset time threshold (e.g., 30 days, 3 months, etc.). The plurality of small files corresponding to the history aggregation file also are referred to as the history small files.

[0239] The update processing of the aggregation file includes re-aggregating the correlated small files to generate a new aggregation file. The correlated small files are historical small files with high correlation degrees between different historical aggregation files.

[0240] In some embodiments, the one or more management nodes obtain the plurality of historical small files and their attribute information corresponding to each history aggregation file based on the aggregation file metadata of different historical aggregation files. And based on the attribute information of the plurality of historical small files, a plurality of correlated small files are determined, and further, the plurality of history aggregation files are updated based on the plurality of correlated small files.

[0241] In some embodiments, the one or more management nodes determine clustering features based on attribute information of the historical small files, and cluster the historical small files based on the clustering features to determine the plurality of correlated small files. A plurality of cluster centers are determined after clustering the historical small files, and the historical small file corresponding to each cluster center is determined as a cluster. The history small file corresponding to each cluster class is taken as a correlated small file. The clustering of the historical small files is performed based on an algorithm such as the K-Means clustering algorithm, the density-based clustering algorithm (DBSCAN algorithm), etc.

[0242] In some embodiments, the clustering feature of a historical small includes an access frequency to the historical small file. The access frequency of the historical small is determined based on dynamic attribute information and/or the labeling information of the historical small file. For example, in response to receiving a user's operation (e.g., viewing) on the historical small file in the historical aggregated file, the one or more management nodes accumulate the access frequency of the historical small file by 1, and update the small file metadata corresponding to the historical small file. In some embodiments of the present disclosure, as a great count of different history small files are distributed in different history aggregation files, considering the access frequency to each of the history small files, by clustering the plurality of historical small files with access frequencies, I/O operations on the aggregation files are reduced, and the management efficiency of the small files is improved.

[0243] In some embodiments of the present disclosure, through generating the aggregation file by performing the file aggregation processing on the plurality of small files, an occurrence of a great amount of storage fragments in the one or more data nodes is reduced, and an overhead for the one

or more management nodes to store the metadata of a great count of small files is reduced, as compared to storing a great count of small files individually. In general, the storage efficiency and management performance of the distributed storage system are improved, thereby improving the reliability of the entire distributed storage system.

[0244] FIGs. 7A and 7B are flowcharts illustrating exemplary processes for data reading according to some embodiments of the present disclosure.

[0245] In some embodiments, a process 700 is performed by a distributed storage system. As shown in FIG. 7A, the process 700 includes following operations.

[0246] In 710, in response to receiving a data read request, a file read handle may be created by a service module implemented on a client.

[0247] In some embodiments, as shown in FIG. 7B, through operation 711, the service module implemented on the client creates the file read handle to initialize a data buffer based on a data read request.

[0248] In 720, by the service module, whether the file write handle exists may be determined.

[0249] In 730, in response to determining that the file write handle exists, by the client service module, the file read handle may be pointed to the data buffer of the file write handle, and latest data from the data buffer may be merged with read disk data, and the latest read data may be returned.

[0250] Combining FIG. 7B, the service module implemented on the client, when determining that the file write handle exists, points the file read handle to a file write handle buffer through operation 731 and performs operation 732.

[0251] In some embodiments, in a read-while-write scenario, latest written data exists in a write buffer corresponding to the write handle, which is not yet written to the storage device (e.g., a disk) of one or more data nodes. At this time, the service module implemented on the client is unable to read the latest data on the disk. Therefore, during a read operation, if the write handle exists, a read scenario needs to sense the existence of a written scenario to avoid that the read data is not the latest data (e.g., historical cache data), and when returning the data, a determination is made according to a cache hit to combine the finally returned data, to ensure correctness of the finally returned data. It is noted that the cache hit refers to whether target data (e.g., read data, a data

piece corresponding to a target file, or a portion of the target file, etc.) exists in a cache (e.g., a cache device of the one or more data nodes), and in response to that the target data exists in the cache, it indicates the cache hit.

[0252] In response to that the file write handle does not exist, operation 732 may be performed directly, and the service module implemented on the client also obtains file metadata and accesses the corresponding one or more data nodes to read the data piece.

[0253] In 733, by the service module implemented on the client, whether the read data pieces of the one or more data nodes involve the cache hit may be determined.

[0254] In some embodiments of the present disclosure, to reduce a risk of data consistency in a process of data modification, a two-phase modification is performed for a modification operation. Specifically, the service module implemented on the client sends pre-modification data to the one or more data nodes, and based on a pre-modification result of the one or more data nodes, the one or more data nodes are informed of the two-phase modification processing. As I/O operations of a plurality of data nodes are asynchronous operations, there is a situation where after the two-phase modification processing (e.g., flushing the pre-modification data to the disk) returns successfully, the data is still in the data node cache. Therefore, the read data must contain the portion of the latest data in the cache, and based on a result of whether or not the cache hit involves, the latest data combined within the one or more data nodes is returned to the service module implemented on the client. More descriptions of the pre-modification may be found in FIG. 4 and the descriptions thereof, and more descriptions of the two-phase modification may be found in FIG. 3A and the descriptions thereof.

[0255] Specifically, in response to that the read data piece involves the cache hit, operation 734 may be performed to combine the data piece cached by the one or more data nodes with the read data piece to obtain the latest data piece, and further, operation 735 may be performed where the one or more data nodes return the data piece to the service module implemented on the client.

[0256] In response to that the read data piece does not involve the cache hit, operation 735 may be performed, and the one or more data nodes return the data piece directly to the service module

implemented on the client. It is understood that when the read data piece does not involve the cache hit, the read data piece of the one or more data nodes at this time is the latest data.

[0257] In some embodiments, the service module implemented on the client, upon receiving the data piece returned by the one or more data nodes, operation 736 may be performed to perform a consistency verification on the data piece to determine whether a fault tolerance condition is satisfied.

[0258] The consistency verification is performed in the manner shown in FIG. 7C, and the fault tolerance condition is determined based on a count of inconsistent data pieces in the plurality of data pieces and a fault tolerance threshold. For example, when the count of inconsistencies is less than or equal to the fault tolerant data threshold, the fault tolerance condition is satisfied, and when the count of inconsistencies is greater than the fault tolerant data threshold, the fault tolerance condition is not satisfied. More descriptions of the fault tolerance threshold may be found in FIG. 3A and the descriptions thereof. In response to that the fault tolerance condition is not satisfied, the read fails.

[0259] In response to satisfying the fault tolerance condition, operation 737 may be performed to determine whether the write handle buffer involves hit, and if the write handle buffer involves hit, operation 738 may be performed to combine the latest data of the write handle buffer and a data return (e.g., return the data to a user) may be performed. If the write handle buffer does not involves hit, the data is returned directly.

[0260] In some embodiments of the present disclosure, considering that real data stored in storage media of the one or more data nodes (e.g., disks), and the data in the cache (e.g., the cache of the one or more data nodes and/or the service module implemented on the client, etc.) are historical data, the hit determination is performed on the data in the cache, and the disk data and the cache data are combined after it is determined that the data is hit, to ensure that the user is reading the latest and most complete content of the data.

[0261] FIG. 7C is a schematic diagram illustrating a data reading inconsistency according to some embodiments of the present disclosure.

[0262] In a data read scenario of a distributed storage system, there are some data inconsistencies. For example, a plurality of data nodes returns data pieces corresponding to a

target read file or a portion of the target read file, where the data pieces are historical data pieces, or there is a data corruption and other situations that lead to the inconsistencies in a target read file.

[0263] At this time, there is a problem of dirty reading, and data restoration is required. As shown in FIG. 7C, in data piece 1, data piece 2,..., data piece 5, respectively returned by the current data nodes, data piece 3 (shown in gray in the figure) is the historical data and the rest of the data pieces are the latest data.

[0264] In some embodiments, a transaction identifier is set with an initial value 0, and the value of the transaction identifier is accumulated by 1 for each written or modification.

[0265] For example, in the method for data modification illustrated in some embodiments of the present disclosure, for the plurality of data pieces within the same group of data pieces, the transaction identifiers of the data pieces in the same processing event need to be consistent. The same group of data pieces refers to data pieces corresponding to the one or more data nodes in the same request (e.g., a pre-modification request, a two-phase modification request).

[0266] For example, in a pre-modification phase, when the service module implemented on the client sends each data piece to the one or more data nodes in a one-to-one correspondence, the transaction identifier is accumulated by 1. For example, the transaction identifier is set from 0 to 1; when the one or more data nodes caches the received data piece successfully, the transaction identifier of each data piece is also accumulated by 1; similarly, the transaction identifier of each data piece is accumulated by 1 when committed to the disk.

[0267] Merely by way of example, as shown in FIG. 7C, when the transaction identifier of data piece 3 is inconsistent with the transaction identifiers of the rest of the data pieces, it indicates that the data piece 3 is the historical data, i.e., there is an inconsistency in the group of data pieces. In some embodiments of the present disclosure, by introducing the transaction identifier, when the data is read, the transaction identifiers of the different data pieces are compared, and whether there is the problem of consistency among the plurality of data pieces is determined, so as to ensure the correctness of data reading.

[0268] FIG. 8 is a schematic diagram illustrating a workflow of a user-state file system according to some embodiments of the present disclosure.

[0269] A file system is used to manage data on a storage device (e.g., a disk, a memory, a cache, etc.), for example, organize the data into directories and files, and provide interfaces (e.g., the interfaces based on a POSIX standard) for directories and/or file operations such as creating, reading, writing, deleting, etc., so that the file system provides users or applications with services of storing and/or managing data through these interfaces after the file system is mounted. The file system includes a fourth extended file system (EXT4), a network file system (NFS), and various other types of file systems. The file system usually runs in the kernel space of an operation system.

[0270] A file system in userspace (FUSE) framework consists of a keral module (fuse.ko), a userspace libfuse, and a fusemount, etc. With the FUSE framework, the user (e.g., a developer) implements or customizes the file system in the user space of the operation system without modifying a kernel. The file system implemented based on the FUSE framework is also referred to as a Filesystem in Userspace, such as the Fuse shown in FIG. 8.

[0271] FIG. 8 is a schematic diagram illustrating a workflow of a Filesystem in Userspace. As shown in FIG. 8, IO requests of a user (e.g., IO operation requests such as view, open, read, etc. on a file) enter a kernel space through a System Call, and the IO requests are sent to the FUSE through a Virtual File System (VFS) in the kernel space. The FUSE forwards the IO requests to a user space program (./fuse_user) corresponding to a mounting tool for processing, and a result of the processing is returned in the same way.

[0272] The FUSE provided by some embodiments of the present disclosure is applied to one or more management nodes and/or one or more data nodes, which makes a distributed storage system provided by some embodiments of the present disclosure more scalable and flexible. In this way, a quick customization is performed according to actual requirements to adapt to the requirements of a large-scale data management.

[0273] FIG. 9 is a schematic diagram illustrating a device for data random write according to some embodiments of the present disclosure.

[0274] As shown in FIG. 9, a device for data random write 900 includes a processor 910, a memory 920, an I/O device 930, and a bus 940. The processor 910, the memory 920, and the I/O device 930 are each connected to the bus 940. Program instructions are stored in the memory

920, and the processor 910 is used to perform the program instructions to implement a method for data random write described in the above embodiment.

[0275] In some embodiments, the processor 910 is also referred to as a central processing unit (CPU). The processor 910 is an integrated circuit chip with a signal processing capability. The processor 910 is also a general-purpose processor, a digital signal processor (DSP), an application specific integrated circuit (ASIC), a field programmable gate array (FPGA), or other programmable logic devices, a discrete gate or a transistor logic device, or a discrete hardware component. The general-purpose processor is a microprocessor, or the processor 910 is any conventional processor, etc.

[0276] FIG. 10 is a schematic diagram illustrating a computer storage medium according to some embodiments of the present disclosure.

[0277] As shown in FIG. 10, a computer storage medium 1000 stores a computer program 1010 which, when performed, implements the method provided in the above embodiment of the present disclosure. The computer program 1010 forms a program file to be stored in the form of a software product in the above computer storage medium 1000 to make a computer device (which is a personal computer, a server, or a network device, etc.) or a processor to perform all or some of the operations of the method of the various embodiments of the present disclosure. And the aforementioned computer storage media 1000 includes USB flash drives, mobile hard disks, read-only memories (ROM), random access memories (RAM), disks or CD-ROMs, and other kinds of media that stores program codes, or terminal devices such as computers, servers, cell phones, tablets, etc.

[0278] It is noted that the foregoing descriptions of the processes are for the purpose of exemplification and illustration only and do not limit the scope of application of the present disclosure. For those skilled in the art, various corrections and changes can be made to the process under the guidance of the present disclosure. However, these corrections and changes remain within the scope of the present disclosure.

[0279] The basic concepts have been described above, and it is apparent to those skilled in the art that the foregoing detailed disclosure serves only as an example and does not constitute a limitation

of the present disclosure. While not expressly stated herein, those skilled in the art may make various modifications, improvements, and amendments to the present disclosure. Those types of modifications, improvements, and amendments are suggested in the present disclosure, so those types of modifications, improvements, and amendments remain within the spirit and scope of the exemplary embodiments of the present disclosure.

[0280] Meanwhile, the present disclosure uses specific words to describe embodiments of the present disclosure. For example, “an embodiment”, “one embodiment”, and/or “some embodiments” means a feature, a structure, or a characteristic associated with at least one embodiment of the present disclosure. Accordingly, it should be emphasized and noted that “an embodiment,” “one embodiment,” or “a count of embodiments” referred to two or more times in different position in the present disclosure do not necessarily refer to the same embodiment. In addition, certain features, structures, or characteristics of one or more embodiments of the present disclosure are suitably combined.

[0281] Furthermore, unless expressly stated in the claims, the order of processing elements and sequences, the use of numerical letters, or the use of other names described herein are not intended to limit the order of the processes and methods of the present disclosure. While some embodiments of the invention that are currently considered useful are discussed in the foregoing disclosure by way of various examples, it should be appreciated that such details serve only illustrative purposes, and that additional claims are not limited to the disclosed embodiments. Rather, the claims are intended to cover all amendments and equivalent combinations that are consistent with the substance and scope of the embodiments of the present disclosure. For example, although the implementation of various components described above is embodied in a hardware device, it is also implemented as a software only solution, e.g., an installation on an existing server or mobile device.

[0282] Similarly, it should be noted that to simplify the presentation of the disclosure of the present disclosure, and thereby aiding in the understanding of one or more embodiments of the present disclosure, the preceding description of embodiments of the present disclosure sometimes combines a variety of features into a single embodiment, accompanying drawings, or description thereof.

However, this method of present disclosure does not imply that the objects of the present disclosure require more features than those mentioned in the claims. Rather, claimed subject matter lies in less than all features of a single foregoing disclosed embodiment.

[0283] In some embodiments, counts describing the components, attributes, and properties are used, and it is to be understood that such counts used in the description of embodiments are modified in some examples by the modifiers “about,” “approximately,” “substantially,” “approximately,” or “generally”. Unless otherwise noted, the terms “about,” “approximate,” or “approximately” indicates that a ±20% variation in the stated count is allowed. Correspondingly, in some embodiments, the numerical parameters used in the present disclosure and claims are approximations, which are subject to change based on the desired characteristics of individual embodiments. In some embodiments, the numerical parameters should consider the specified count of valid digits and employ general place-keeping. While the numerical domains and parameters used to confirm the breadth of their ranges in some embodiments of the present disclosure are approximations, in specific embodiments such values are set to be as precise as possible within a feasible range.

[0284] For each patent, patent application, patent application disclosure, and other material cited in the present disclosure, such as articles, books, manuals, publications, documents, etc., the entire contents of which are hereby incorporated by reference herein. Application history documents that are inconsistent with or conflict with the contents of the present disclosure are excluded, as are documents (currently or hereafter appended to the present disclosure) that limit the broadest scope of the claims of the present disclosure. It should be noted that to the extent that there is an inconsistency or conflict between the descriptions, definitions, and/or use of terms in the materials appurtenant to the present disclosure and those set forth herein, the descriptions, definitions and/or use of terms in the present disclosure shall prevail.

[0285] Finally, it should be understood that the embodiments described in the present disclosure are only used to illustrate the principles of the embodiments of the present disclosure. Other deformations also fall within the scope of the present disclosure. As such, alternative configurations of embodiments of the present disclosure are viewed as consistent with the teachings

of the present disclosure as an example, not as a limitation. Correspondingly, the embodiments of the present disclosure are not limited to the embodiments expressly presented and described herein.

WHAT IS CLAIMED IS:

1. A method for data modification, comprising:

storing large files in one or more data nodes; and

in response to receiving a modification instruction, performing a modification process on a target file at a target modification position of the target file, the target file including the large files.

2. The method of claim 1, wherein the in response to receiving a modification instruction, performing a modification process on a target file at a target modification position of the target file, the target file including the large files includes:

in response to receiving the modification instruction, by a service module implemented on a client, determining pre-modification data by performing a pre-modification process on the target file, and sending the pre-modification data to the one or more data nodes;

caching, by the one or more data nodes, the pre-modification data and returning, by the one or more data nodes, a pre-modification result to the service module;

in response to that the pre-modification result satisfies a preset condition, committing, by the service module, a two-phase modification request to the one or more data nodes; and

writing, by the one or more data nodes, the pre-modification data to the target file based on the two-phase modification request.

3. The method of claim 2, wherein the method further includes:

determining, based on the pre-modification result, a count of abnormal data nodes in the one or more data nodes; and

in response to that the count of abnormal data nodes is less than or equal to a fault tolerance threshold, determining that the pre-modification result satisfies the preset condition.

4. The method of claim 3, wherein the fault tolerance threshold is related to a count of parity blocks.

5. The method of claim 2, wherein the method further includes:

in response to determining that the pre-modification result does not satisfy the preset condition, sending a rollback request to the one or more data nodes by the service module, the rollback request being configured to instruct the one or more data nodes to perform a rollback transaction of the two-phase modification request.

6. The method of claim 2, wherein the in response to receiving the modification instruction, determining pre-modification data by pre-modifying the target file through a service module includes:

obtaining metadata information from one or more management nodes;

reading the target file from the one or more data nodes based on the metadata information;

positioning the target modification position of the target file based on the modification instruction, the modification instruction including an offset volume of the target file and a length of modification data, and the target modification position being determined based on the offset volume; and

writing the modification data to a plurality of data pieces corresponding to the target file and determining the pre-modification data.

7. The method of claim 6, wherein the writing the modification data into a plurality of data pieces corresponding to the target file includes:

modifying the modification data in a strip cache corresponding to the target file to determine strip cache data;

determining parity block data based on the strip cache data; and

writing the modification data and the parity block data to the plurality of data pieces.

8. The method of claim 1, wherein the target file includes an aggregation file, the method further includes:

generating the aggregation file by performing a file aggregation process on a plurality of small files.

9. The method of claim 8, wherein the aggregation file includes aggregation file metadata, the method further includes:

storing the aggregation file metadata based on a key-value pair storage mode, the aggregation file metadata indicating a mapping relationship between the plurality of small files and the aggregation file.

10. The method of claim 9, wherein the key-value pair storage mode includes a key-value data table stored in one or more management nodes.

11. The method of claim 8, wherein the generating the aggregation file by performing a file aggregation process on a plurality of small files includes:

obtaining attribute information of the plurality of small files;

determining target small files from the plurality of small files based on the attribute information; and

generating the aggregation file by performing the file aggregation process on the target small files.

12. The method of claim 11, wherein the determining target small files from the plurality of small files based on the attribute information includes:

determining a correlation degree between the plurality of small files based on attribute information of each of the plurality of small files; and

determining the target small files from the plurality of small files to be processed based on the correlation degree.

13. The method of claim 12, wherein the plurality of small files includes labeling information, and the determining a correlation degree between the plurality of the small files includes:

determining a similarity degree between the plurality of small files based on the attribute

information and labeling information of each of the plurality of small files; and

determining the correlation degree between the plurality of small files based on the similarity degree.

14. The method of claim 9, wherein the method further includes:

determining, based on aggregation file metadata corresponding to a plurality of historical aggregation files, historical small files corresponding to the plurality of historical aggregation files;

determining a clustering feature based on attribute information of the historical small files;

clustering the historical small files based on the clustering feature to determine a correlated small file; and

updating the plurality of historical aggregation files based on the correlated small file.

15. The method of claim 14, wherein the clustering feature includes an access frequency of each of the historical small files.

16. The method of claim 1, wherein the method further includes:

storing directory information of the large files through one or more management nodes, the directory information including a root directory and a user directory.

17. The method of claim 2, wherein the method further includes:

after the one or more data nodes write the pre-modification data to the target file according to the two-phase modification request,

performing, by the one or more data nodes, a cyclic redundancy check on a data block corresponding to the target file to obtain a data block status;

when the data block status is abnormal, sending, by the one or more data nodes, corresponding data block information to one or more management nodes to enable the one or more management nodes to perform a file status update and/or a file data recovery on the target file.

18. The method of claim 1 further comprising:

in response to receiving a data read request, establishing a file read handle through a service module implemented on a client;

determining, by the service module, whether the file write handle exists;

in response to determining that the file write handle exists, by the client service module, pointing the file read handle to a data buffer of the file write handle, and merging latest data from the data buffer with read disk data, and returning the latest read data.

19. A system for data modification including a service module implemented on a client, one or more data nodes, and one or more management nodes, wherein the one or more data nodes are configured to store large files, and the one or more management nodes are configured to store metadata information of the large files; and

the service module is configured to:

in response to receiving a modification instruction of a target file,

obtain the metadata information corresponding to the target file from the one or more management nodes;

read the target file from the one or more data nodes based on the metadata information;

perform a pre-modification process on the target file based on the modification instruction, determine pre-modification data, and send the pre-modification data to the one or more data nodes;

the one or more data nodes are configured to:

cache the pre-modification data and return a pre-modification result to the service module; and

obtain the target file after modification by performing a modification process on the target file based on the pre-modification data.

20. The system of claim 19, wherein the modification instruction includes an offset volume of the target file and a length of modification data;

the service module determines the pre-modification data in the following manner:

modifying the modification data in a strip cache corresponding to the target file to determine

strip cache data;

determining parity block data based on the strip cache data; and

determining the pre-modification data by writing the modification data and the parity block data to the plurality of data pieces.

21. The system of claim 19, wherein

the client service module is further configured to:

in response to that the pre-modification result satisfies a preset condition, commit a two-phase modification request to the one or more data nodes;

the data node is further configured to:

write the pre-modification data to the target file based on the two-phase modification request.

22. A device for data modification comprising:

a memory and a processor connected to the memory; wherein the memory is used to store program instructions, and the processor is used to execute the program instructions for implementing the method of any one of claims 1 to 18.