(21) International Application Number: PCT/CN2014/071592

(22) International Filing Date: 27 January 2014 (27.01.2014)

(25) Filing Language: English

(26) Publication Language: English


(71) Applicant: HUAWEI TECHNOLOGIES CO., LTD. [CN/CN]; Huawei Administration Building, Bantian, Longgang, Shenzhen, Guangdong 518129 (CN).

(72) Inventors: VERMA, Ajit; 780 Los Altos Ave., Los Altos, California 94022 (US). PLOCHER, John; 407 Belmont Ave., San Jose, California 95125 (US). OU, Xiongbing; Building 1, Rm 401, Longboayuan Area 3, Chappning, Beijing (CN).


Declarations under Rule 4.17:
— as to the applicant’s entitlement to claim the priority of the earlier application (Rule 4.17(1)(ii))

Published:
— with international search report (Art. 21(3))

(54) Title: DISTRIBUTED STORAGE OBJECT DELETE

(57) Abstract: System and methods for storage object distribution using dynamic policy controls are provided. An embodiment method deleting an object in a distributed storage system includes receiving, from a client, a delete request for an object, retrieving, by a primary datacenter, an object metadata list corresponding to the object to be deleted, and inserting, by the primary datacenter, a delete marker on a top of a stack data structure in an object metadata list.
Distributed Storage Object Delete

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims priority of US Non-provisional application No. 13/755,573, filed January 31, 2013 and titled "Distributed Storage Object Delete," which is incorporated herein by reference in its entirety.

TECHNICAL FIELD

[0002] The present disclosure relates to distributed storage systems and, in particular embodiments, to distributed storage systems offering reliable object deletion.

BACKGROUND

[0003] Deleting an object due to a DELETE Object request or PUT (Update) Object request in running storage service in a multiple datacenter (MDC) environment is a complex task. Multiple DELETE and PUT requests may arrive during a small window. In addition, the requests may be interleaved. Furthermore, subsequent requests sent to remote datacenters may arrive out of sequence and some of the requests may fail. A distributed storage system needs to handle these cases with deterministic behavior and needs to achieve eventual consistency within a predictable period.

[0004] If distributed system architecture does not handle the DELETE operation properly, there can be several use cases resulting in unintended data losses that cannot be recovered.
Summary

[0005] An embodiment method of deleting an object in a distributed storage system includes receiving, from a client, a delete request for the object, retrieving, by a primary datacenter, an object metadata list corresponding to the object to be deleted, and inserting, by the primary datacenter, a delete marker on a top of a stack data structure in the object metadata list.

[0006] An embodiment method of updating metadata in a distributed storage system includes creating, by an object storage controller, a current version of the metadata corresponding to an object in response to a command received from a client, and chronologically storing, in a storage cluster, the current version of the metadata corresponding to the object along with previous versions of the metadata corresponding to the object.

[0007] An embodiment method of storing metadata in a distributed storage system includes receiving, at an object storage controller, a command corresponding to an object from a client, loading, by the object storage controller, an object metadata list associated with the object from a storage cluster, and updating, with the object storage controller, the object metadata list associated with the object by creating a current version of the metadata and retaining previous versions of the metadata.

[0008] An embodiment datacenter in a distributed storage system includes an object storage controller configured to receive a command regarding an object from a client and to create a current version of metadata corresponding the object in response to the command, and a memory associated with the object storage controller, the memory configured to chronologically store the current version of the metadata corresponding to the object in a stack along with at least one previous version of the metadata corresponding to the object.
BRIEF DESCRIPTION OF THE DRAWINGS

[0009] For a more complete understanding of the present disclosure, and the advantages thereof, reference is now made to the following descriptions taken in conjunction with the accompanying drawing, in which:

[0010] Figure 1 illustrates an embodiment distributed storage system;

[0011] Figure 2 illustrates an embodiment datacenter from the embodiment distributed storage system of Figure 1 is illustrated in further detail;

[0012] Figure 3 illustrates an embodiment object metadata list including current and previous versions of metadata corresponding to an object;

[0013] Figure 4 illustrates an embodiment method of updating metadata in the embodiment distributed storage system of Figure 1;

[0014] Figure 5 illustrates an embodiment method of storing metadata in the embodiment distributed storage system of Figure 1; and

[0015] Figure 6 illustrates a computing platform that may be used for implementing, for example, the devices and methods described herein, in accordance with an embodiment.

[0016] Corresponding numerals and symbols in the different figures generally refer to corresponding parts unless otherwise indicated. The figures are drawn to clearly illustrate the relevant aspects of the embodiments and are not necessarily drawn to scale.
DETAILED DESCRIPTION OF ILLUSTRATIVE EMBODIMENTS

[0017] The making and using of the presently preferred embodiments are discussed in detail below. It should be appreciated, however, that the present disclosure provides many applicable inventive concepts that can be embodied in a wide variety of specific contexts. The specific embodiments discussed are merely illustrative and do not limit the scope of the disclosure.

[0018] The present disclosure will be described with respect to preferred embodiments in a specific context, namely datacenters operating in a distributed storage system. The concepts in the disclosure may also apply, however, to other types of storage systems and associated devices and components.

[0019] Referring now to Figure 1, an embodiment distributed storage system 10 is illustrated. As shown, the embodiment distributed storage system 10 includes several datacenters 12 communicating through a network 14 (e.g., the Internet). It should be recognized that a practical application of the embodiment distributed storage system 10 may include components, devices, hardware, and so on, which have not been included in Figure 1 for ease of illustration. In addition, the embodiment distributed storage system 10 may include more or fewer datacenters 12 than depicted in Figure 1. In an embodiment, one of the datacenters 12, which is labeled DC1, is considered the primary datacenter while the other datacenters, labeled DC2 and DC3, are considered secondary or back-up datacenters.

[0020] Referring now to Figure 2, an embodiment datacenter 12 is illustrated in further detail. As shown, the datacenter 12 includes an object storage controller (OSC) 16, an object integrity service (OIS) 18, and a storage cluster 20. In an embodiment, the object storage controller 16 is a server or other computing device loaded with appropriate software to perform the functions described herein. It should be recognized that the embodiment datacenter 12 may include
components, devices, hardware (e.g., routers, firewalls, cashing controller, etc.), which have not been included in Figure 1 for ease of illustration.

[0021] The object storage controller 16 is configured to interact with clients. In particular, the object storage controller 16 is configured to receive commands from one or more clients regarding an object containing data and stored within the embodiment distributed storage system 10. The command may be a delete object command, an update object command (i.e., a put command), a get command, and so on. In an embodiment, a delete command may not cause an object to be immediately deleted in the embodiment distributed storage system 10. Rather, the delete command may indicate that the object may eventually be permanently deleted from the storage cluster 20. However, the object will not be accessible to clients once the delete command has been successfully processed by the embodiment distributed storage system 10.

[0022] As will be more fully explained below, the object storage controller 16 of the primary datacenter 12 (DC1) may receive different commands (a.k.a., requests, application programming interface (API) calls, etc.) relating to the same object from different clients at or around the same time. For example, a first client may instruct the object storage controller 16 to delete an object at or about the same time that a second client instructs the object storage controller 16 to update the object. Unless handled appropriately, this and other conflict scenarios can cause data integrity problems in the embodiment distributed storage system 10.

[0023] Still referring to Figure 2, the object storage controller 16 includes an object storage controller memory 22. As will be more fully explained below, the object storage controller memory 22 stores, among other things, metadata and metadata lists. Such metadata and/or metadata lists may be uploaded from the storage cluster 20 or sent to the storage cluster 20 for storage.
The object integrity service 18 is in communication with the object storage controller 16. In an embodiment, the object integrity service 18 is a server or other computing device loaded with appropriate software to perform the functions described herein. The object integrity service 18 is configured to detect and correct errors in metadata and objects, check the integrity of data stored in the embodiment distributed storage system 10, react to tasks in a queue, perform scrubbing operations, and so on.

Still referring to Figure 2, the object integrity service 18 includes an object integrity service memory 24. As will be more fully explained below, the object integrity service memory 24 stores, among other things, metadata and metadata lists, task lists, a queue with tasks, and so on. Such metadata and/or metadata lists may be received from the object storage controller 16, from other datacenters 12 (e.g., DC2, DC3), uploaded from the storage cluster 20, and so on.

The storage cluster 20 represents a plurality of servers or other computing devices. In an embodiment, the storage cluster 20 is a collection of servers each having a storage device. The storage device may take the form of a magnetic disk, an optical drive, flash memory, and so on. The storage cluster 20 may be referred to as Sea of Disk (SoD). As will be more fully explained below, the storage cluster 20 is configured to store metadata pertaining to an object in a metadata list format as shown Figure 3.

As shown in Figure 3, the object metadata list 26 includes metadata associated with a current version of an object, which is referred to as a current version of the metadata 28. The object metadata list 26 also includes metadata associated with previous or older versions of the metadata for the same object, which is referred to as previous versions of the metadata 30. The metadata in the object metadata list 26 of Figure 3 is used within the embodiment distributed storage system 10 to coordinate the activities associated with a version of the object (e.g., parity generation, compression, multi-datacenter copying, recovery, deletion, garbage collection, etc.).
As shown in Figure 3, the current version of the metadata 28 and the previous versions of the metadata 30 are organized in a stack data structure 32. As such, the metadata in the object metadata list 26 may be referred to as generations of metadata, a list of versions of metadata, and so on. Indeed, each PUT request received from a client regarding an object creates a new "current" version of the metadata. The new "current" version of the metadata is stored in the object metadata list 26 along with the previous versions of the metadata (including the one that is no longer current) for that object. Notably, the previous versions of the metadata 30 are retained when the current version of the metadata 28 is created and placed in stack data structure 32 of the object metadata list 26.

In an embodiment, the current version of the metadata 28 is disposed at the top of the stack data structure 32 as oriented in Figure 3. Moreover, the current version of the metadata 28 and the previous versions of the metadata 30 are stored chronologically in the object metadata list 26. If a DELETE request is received from the client, a special version of the metadata, known as a delete marker, is placed on the stack data structure 32. For example, in Figure 3 the previous version of the metadata 30 labeled "Version 3 metadata DELETE" is a delete marker, which signifies that that the version of the object below the delete marker, in this case the version of the object that corresponds to "Version 2 metadata PUT," has been or will be deleted.

Still referring to Figure 3, the current version of the metadata 28 and the previous versions of the metadata 30 each include, for example, an object name, a universal unique identifier (UUID), a timestamp, an entity tag (ETag), an object size, an access control list, and so on. The UUID identifies the object itself while the ETag identifies the content of the object. Indeed, in an embodiment the ETag is a checksum or hash value of the content. The UUID is internal to the embodiment distributed storage system 10, whereas the ETag is returned as a header in response to some of the client requests.
Because the current version of the metadata 28 is situated at the top of the object metadata list 26 above the previous versions of the metadata 30, the datacenters 12 know which version of the metadata is the most current. Indeed, and as will be more fully explained below, the data centers 12 benefitting from the embodiment distributed storage architecture described herein are able to compare the information in the metadata (e.g., the ETag, the timestamp, etc.) in the current version of the metadata 28 and the previous versions of the metadata 30 to resolve conflicting commands received from different clients at or about the same time. A few examples are provided herein to illustrate the point.

To begin, when a client intends to upload an object, the client sends the following update command (i.e., PUT) or request for the object to the primary datacenter 12 (DC1) in Figure 1, which for the purposes of this example has the universal resource locator (URL) of uds.example.com.

```
PUT bucketName/objectName HTTP/1.1
Host: uds.example.com
Date: Wed 19 Dec 2012 17:50:00 GMT
Content-Type: plain/text
Content-Length: 6748
Authorization: XXX xxx:yyy

[6748 bytes of data]
```

When the update command is received, the primary datacenter (DC1) takes the following actions. First, the primary datacenter 12 (DO) retrieves the object metadata list 26 corresponding to the object being updated from the storage cluster 20 and temporarily stores it in the object storage controller memory 22. Thereafter, the primary datacenter 12 (DO) creates a new current version of the metadata 28 and places that new version of the metadata on the top of the stack data structure 32 in the object metadata list 26 shown in Figure 3.
Once the new current version of the metadata 28 has been placed at the top of the stack data structure 32 in the object metadata list 26, the primary datacenter 12 (DC1) records the system time as the timestamp in the new current version of the metadata 28. The timestamp is considered the object creation time.

The primary datacenter 12 (DC1) also generates the UUID and stores the UUID in the new current version of the metadata 28. Once the UUID has been created, the primary datacenter 12 (DC1) will begin writing the content or data corresponding to the update request in UUID segments (e.g., UUID.O, UUID.1, UUID.2, etc.) until all the content or data has been updated per the request from the client. The primary datacenter 12 (DC1) also generates an ETag for the object received from the client. The ETag, which is based on the content of the object, is saved in the new current version of the metadata 28.

If the upload is successful, the primary datacenter 12 (DC1) in Figure 1 sends the following response to the client, which includes the timestamp that was recorded and the ETag that was generated.

HTTP/1.1 200 OK
Date: Wed 19 Dec 2012 17:55:45 GMT
ETag: "ib2cf535f27731c974343645a3985328"
Server: embodiment distributed storage

Where multiple datacenters 12 are involved as illustrated in Figure 1, the primary datacenter 12 (DC1) may forward the update command to the secondary datacenters 12 (DC2, DC3). In an embodiment, the primary datacenter 12 (DC1) forwards the update command as soon as possible after it is received in order to create a synchronous copy of the object on one of the secondary datacenters 12. In an embodiment, the primary datacenter 12 (DC1) may also request that an asynchronous copy of the object be stored on one or more of the secondary datacenters 12.
In other words, when the update command is received at the primary datacenter DC1, the object may be updated at the primary datacenter DC1 and at the secondary primary datacenter DC2 immediately followed by an instruction that the object be updated at another secondary datacenter DC3 sometime in the future when, for example, system resources or other factors deem appropriate. As such, at least one backup copy of the object is made right away. In an embodiment, the synchronous copy of the object is made at the secondary datacenter DC2 prior to the response being sent to the client indicating a successful upload.

Where multiple datacenters are involved, the update request will look similar to the one depicted above. However, an additional header, x-hws-request-timestamp, is added. The value of this header is the timestamp discussed above, as shown.

```
PUT bucketName/objectName HTTP/1.1
Host: uds2.example.com
Date: Wed 19 Dec 2012 17:52:00 GMT
Content-Type: plain/text
Content-Length: 6748
Authorization: XXX abc:df
x-hws-request-timestamp: Wed 19 Dec 2012 17:50:35 GMT
[6748 bytes of data]
```

Now that the current version of the metadata in the object metadata list includes an ETag corresponding to the most recent version of the content, the ETag may be beneficially used in connection with subsequent DELETE requests. For example, when the primary datacenter DC1 sends subsequent DELETE requests to the secondary datacenters DC2, DC3, it will include the ETag value in the If-Match header. The secondary datacenters DC2, DC3 will match the ETag value with the value that they have stored. If the received and stored ETag values match, the DELETE request is successfully executed by the secondary datacenters DC2, DC3. If the
received and stored ETag values do not match, the DELETE request is ignored. In an embodiment, the client may optionally send an ETag header along with the DELETE request.

[0041] In light of the above, where multiple datacenters are involved the ETag may be used to ensure that commands are executed properly. By way of example, if the primary datacenter 12 (DC1) and the secondary datacenter 12 (DC2) each have version three of an object, but the other secondary datacenter 12 (DC3) still has version two of an object, a command from a client to delete the object will include the ETag that indicates which version, either version three or version two in this example, should be deleted. By knowing the correct version to act on, race conditions and other potentially detrimental conflicting command problems are alleviated in the multiple datacenter 12 context.

[0042] When a client intends to delete an object, the client sends the following delete command or request for the object to the primary datacenter 12 (DC1) in Figure 1, which for the purposes of this example has the universal resource locator (URL) of uds.example.com.

DELETE bucketName/objectName HTTP/1.1
Host: uds.example.com
Date: Wed 19 Dec 2012 18:50:00 GMT
Content-Type: plain/text
Authorization: XXX xxx:yyy

[0043] When the delete command is received, the primary datacenter (DC1) takes the following actions. First, the primary datacenter 12 (DO) retrieves the object metadata list 26 corresponding to the object being deleted from the storage cluster 20 and temporarily stores it in the object storage controller memory 22. Thereafter, the primary datacenter 12 (DO) inserts a delete marker on the top of the stack data structure 32 in the object metadata list 26 shown in Figure 3.
After the delete marker has been placed at the top of the stack data structure 32, the version of the object immediately below the delete mark in the stack data structure 32 is no longer available if a new READ request is received from the client.

If the delete is successful, the primary datacenter 12 (DCI) in Figure 1 sends the following response to the client, which includes the timestamp that was recorded.

HTTP/1.1 204 No Content
Date: Wed 19 Dec 2012 17:55:45 GMT
Server: embodiment distributed storage

Where multiple datacenters 12 are involved, as illustrated in Figure 1, the primary datacenter 12 (DCI) may forward the delete command to the secondary datacenters 12 (DC2, DC3). The delete request will look similar to the one depicted above. However, two additional headers, "x-hws-request-timestamp" and "If-match," are added. The value of the "x-hws-request-timestamp" header is the timestamp and the value of the "If-match" header is the value of the ETag that was generated during the update command when the object was created or uploaded, as shown.

DELETE bucketName/objectName HTTP/1.1
Host: uds2.example.com
Date: Wed 19 Dec 2012 17:52:00 GMT
Authorization: XXX abc:def
x-hws-request-timestamp: Wed 19 Dec 2012 17:50:35 GMT
If-Match: "Ib2cf535f27731c974343645a3985328"
[6748 bytes of data]

If the delete is successful, the secondary datacenter 12 (DC2, DC3) sends back the following response, which includes the ETag, to the primary datacenter 12 (DCI) in Figure 1.

HTTP/1.1 204 No Content
Date: Wed 19 Dec 2012 17:53:41 GMT
Now that the current version of the metadata 28 in the object metadata list includes an ETag corresponding to the most recent version of the content, the ETag may be beneficially used in connection with subsequent DELETE requests. For example, when the primary datacenter 12 (DC1) sends subsequent DELETE requests to the secondary datacenters 12 (DC2, DC3), it will include the ETag value in the If-Match header. The secondary datacenters 12 (DC2, DC3) will match the ETag value with the value that they have stored. If the received and stored ETag values match, the DELETE request is successfully executed by the secondary datacenters 12 (DC2, DC3). If the received and stored ETag values do not match, the DELETE request is ignored. In an embodiment, the client may optionally send an ETag header along with the DELETE request.

Notably, if any of the secondary datacenters 12 (DC2, DC3) is not using the embodiment distributed storage architecture described herein and is instead using, for example, an Amazon S3 compatible system, then the secondary datacenter 12 (DC2, DC3) would likely ignore both the x-hws-request-timestamp and If-match headers. In the alternative, the secondary datacenters 12 (DC2, DC3) may attempt to use other information, such as the timestamp, to resolve and conflicts. In either case, the secondary datacenter 12 (DC2, DC3) would not send back the ETag header in response to the primary datacenter 12 (DC1). This might cause the secondary datacenter 12 (DC2, DC3) without the embodiment distributed storage architecture described herein to delete a newer version of the object unintentionally.

For example, assume that the primary datacenter 12 (DC1) and one of the secondary datacenters 12 (DC2) have been updated to a version three of the object and another of the secondary datacenters 12 (DC3) still contains version two of the object. If the third datacenter 12 (DC3) is not equipped with the embodiment distributed storage architecture described herein, the
third datacenter 12 (DC3) will not recognize the ETag in the request to delete version three of the object. As such, the third datacenter 12 will simply delete version two of the object despite the request being intended for version three. In contrast, both the primary datacenters 12 (DC1) and the secondary datacenter 12 (DC2) that do have the embodiment distributed storage architecture described herein, will delete version three of the object as instructed and will leave version two of the object alone.

[0051] If the above unintended delete at the secondary datacenter 12 (DC3) of the second version of the object occurs, the object integrity service 18 will eventually recognize the problem and restore the second version of the object which was inappropriately deleted so that the integrity of the data in the embodiment distributed storage system is maintained. In other words, the object integrity service 18 running in the primary datacenter 12 (DC1) would take action to make the object status consistent in all the datacenters 12.

[0052] In contrast to the above, where the secondary datacenters 12 (DC2, DC3) are using the embodiment distributed storage architecture described herein, the secondary datacenters 12 (DC2, DC3) will compare the ETag value in the If-Match header of the request with the ETag value corresponding to the object. If the values don't match, the secondary datacenters (DC2, DC3) know to reject the delete request.

[0053] Referring now to Figure 4, an embodiment method 34 of updating metadata in a distributed storage system 10 is illustrated. In block 36, the object storage controller 16 creates a current version of the metadata 28 corresponding to an object in response to a command received from a client. In block 38, the object storage controller memory 22 chronologically stores the current version of the metadata 28 corresponding to the object along with at least one previous version of the metadata 30 corresponding to the object.
[0054] Referring now to Figure 5, an embodiment method 40 of storing metadata in a distributed storage system 10 is illustrated. In block 42, the object storage controller 16 receives a command corresponding to an object from a client. In block 44, the object storage controller 16 loads the object metadata list 26 associated with the object into the object storage controller memory 22. In block 46, the object storage controller 16 updates the object metadata list 26 associated with the object by creating a current version of the metadata 28 and retaining a previous version of the metadata 30.

[0055] Figure 6 is a block diagram of a processing system 80 that may be used for implementing the devices and methods disclosed herein. Specific devices may utilize all of the components shown, or only a subset of the components, and levels of integration may vary from device to device. Furthermore, a device may contain multiple instances of a component, such as multiple processing units, processors, memories, transmitters, receivers, etc. The processing system 80 may include a central processing unit (CPU) 84, memory 86 and a mass storage device 88 connected to a bus 94.

[0056] The bus 94 may be one or more of any type of several bus architectures including a memory bus or memory controller, a peripheral bus, video bus, or the like. The CPU 84 may comprise any type of electronic data processor. The memory 86 may comprise any type of system memory such as static random access memory (SRAM), dynamic random access memory (DRAM), synchronous DRAM (SDRAM), read-only memory (ROM), a combination thereof, or the like. In an embodiment, the memory 86 may include ROM for use at boot-up, and DRAM for program and data storage for use while executing programs.

[0057] The mass storage 88 device may comprise any type of storage device configured to store data, programs, and other information and to make the data, programs, and other information
accessible via the bus 94. The mass storage device 88 may comprise, for example, one or more of a solid state drive, hard disk drive, a magnetic disk drive, an optical disk drive, or the like.

[0058] The processing system 80 also includes one or more network interfaces 96, which may comprise wired links, such as an Ethernet cable or the like, and/or wireless links to access nodes or different networks. The network interface 96 allows the processing system 80 to communicate with remote systems or units via the networks. For example, the network interface 96 may provide wireless communication via one or more transmitters/transmit antennas and one or more receivers/receive antennas. In an embodiment, the processing system 80 (a.k.a., processing unit) is coupled to a local-area network 98 or a wide-area network 98 for data processing and communications with remote devices, such as other processing units, the Internet, remote storage facilities, or the like.

[0059] While the disclosure provides illustrative embodiments, this description is not intended to be construed in a limiting sense. Various modifications and combinations of the illustrative embodiments, as well as other embodiments, will be apparent to persons skilled in the art upon reference to the description. It is therefore intended that the appended claims encompass any such modifications or embodiments.
WHAT IS CLAIMED IS:

1. A method of deleting an object in a distributed storage system, comprising:
   receiving, from a client, a delete request for the object;
   retrieving, by a primary datacenter, an object metadata list corresponding to the object to be
   deleted; and
   inserting, by the primary datacenter, a delete marker on a top of a stack data structure in the
   object metadata list.

2. The method of claim 1, further comprising indicating with the delete marker that a version of
   the object below the delete marker in the stack data structure is no longer available.

3. The method of claim 1, further comprising indicating with the delete marker that a version of
   the object below the delete marker in the stack data structure has been deleted.

4. The method of claim 1, further comprising indicating with the delete marker that a version of
   the object below the delete marker in the stack data structure is scheduled for deletion.

5. The method of claim 1, further comprising forwarding, by the primary datacenter, the delete
   request for the object to a secondary datacenter, the delete request including an entity tag.

6. The method of claim 5, further comprising comparing the entity tag in the delete request
   with a second entity tag stored at the secondary datacenter.
7. The method of claim 1, further comprising forwarding, by the primary datacenter, the delete request for the object to a secondary datacenter, the delete request including an entity tag generated when the object was at least one of created or updated.

8. The method of claim 7, further comprising comparing the entity tag in the delete request with a second entity tag stored at the secondary datacenter.

9. The method of claim 7, further comprising receiving, from the secondary datacenter, a response including the entity tag.

10. The method of claim 1, further comprising forwarding, by the primary datacenter, the delete request for the object to a secondary datacenter, the secondary datacenter executing the delete request if a stored entity tag matches a received entity tag.
11. A method of updating metadata in a distributed storage system, comprising:
   creating, by an object storage controller, a current version of the metadata corresponding to
   an object in response to a command received from a client; and
   chronologically storing, in a storage cluster, the current version of the metadata
   corresponding to the object along with previous versions of the metadata corresponding to the object.

12. The method of claim 11, further comprising organizing the current version of the metadata
   and the previous versions of the metadata in a stack.

13. The method of claim 11, further comprising requesting an object integrity service implement
    updates to an object metadata list on a secondary datacenter.

14. The method of claim 11, further comprising comparing an entity tag (ETag) in the command
    to the entity tag (ETag) in the current version of the metadata and the entity tag (ETag) in one of the
    previous versions of the metadata.

15. The method of claim 14, wherein the comparing is performed to resolve conflicting
    commands received simultaneously from the client and at least one other client.

16. The method of claim 11, wherein the current version of the metadata includes an entity tag
    (ETag) that corresponds to content of the object.

17. The method of claim 11, wherein the previous versions of the metadata include an entity tag
    (ETag) that corresponds to content of the object.

18. The method of claim 11, wherein the command is one of a delete command and an update
    command.
19. The method of claim 11, further comprising chronologically storing, in the storage cluster associated with the object storage controller, the current version of the metadata corresponding to the object along with the previous versions of the metadata corresponding to the object.
20. A method of storing metadata in a distributed storage system, comprising:
   receiving, at an object storage controller, a command corresponding to an object from a
   client;
   loading, by the object storage controller, an object metadata list associated with the object
   from a storage cluster; and
   updating, with the object storage controller, the object metadata list associated with the
   object by creating a current version of the metadata and retaining previous versions of the metadata.

21. The method of claim 20, further comprising organizing the current version of the metadata
   and the previous versions of the metadata in a chronological stack.

22. The method of claim 20, further comprising storing the current version of the metadata and
   the previous versions of the metadata in the storage cluster as a chronological stack.

23. The method of claim 20, further comprising generating an entity tag for a new version of the
   object and recording the entity tag in the current version of the metadata.

24. The method of claim 23, further comprising comparing the entity tag in the command to the
   entity tag in the current version of the metadata.

25. The method of claim 20, further comprising requesting an object integrity service implement
   the object metadata list as updated on a secondary datacenter remote from a primary datacenter.

26. The method of claim 20, wherein the current version of the metadata includes a first
   universal unique identifier (UUID) and a first timestamp in a first header and each of the previous
versions of the metadata includes a second universal unique identifier (UUID) and a second
timestamp in a second header.

27. The method of claim 20, wherein the command is a delete command and the current version
of the metadata comprises a delete marker.

28. The method of claim 27, wherein the delete marker is stored above one of the previous
versions of the object to be deleted.
29. A datacenter in a distributed storage system, comprising:

an object storage controller configured to receive a command regarding an object from a
client and to create a current version of metadata corresponding the object in response to the
command; and

a memory associated with the object storage controller, the memory configured to
chronologically store the current version of the metadata corresponding to the object in a stack along
with previous versions of the metadata corresponding to the object.

30. The datacenter of claim 29, wherein the object storage controller is configured to generate an
entity tag and to record the entity tag in the current version of the metadata.

31. The datacenter of claim 29, further comprising an object integrity service in communication
with the object storage controller, the object integrity service configured to request that a secondary
datacenter, which is remotely located relative to the object storage controller, update the stack to
include the current version of the metadata.
CREATE A CURRENT VERSION OF THE METADATA CORRESPONDING TO AN OBJECT IN RESPONSE TO A COMMAND FROM A CLIENT

CHRONOLOGICALLY STORE THE CURRENT VERSION OF THE METADATA CORRESPONDING TO THE OBJECT ALONG WITH AT LEAST ONE PREVIOUS VERSION OF THE METADATA CORRESPONDING TO THE OBJECT

FIG. 4

RECEIVE A COMMAND CORRESPONDING TO AN OBJECT FROM A CLIENT

LOAD AN OBJECT METADATA LIST ASSOCIATED WITH THE OBJECT INTO AN OBJECT STORAGE CONTROLLER MEMORY

UPDATE THE OBJECT METADATA LIST ASSOCIATED WITH THE OBJECT BY CREATING A CURRENT VERSION OF THE METADATA AND RETAINING A PREVIOUS VERSION OF THE METADATA

FIG. 5
**INTERNATIONAL SEARCH REPORT**

**INTERNATIONAL APPLICATION**

PCT/CN2014/071592

**A. CLASSIFICATION OF SUBJECT MATTER**

G06F 17/30(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

**B. FIELDS SEARCHED**

Minimum documentation searched (classification system followed by classification symbols)

G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

WPI; EPODOC; CNKI; CNPAT; IEEE: distribute, delete, time, version, store, update, list, metadata

**C. DOCUMENTS CONSIDERED TO BE RELEVANT**

<table>
<thead>
<tr>
<th>Category</th>
<th>Citation of document, with indication, where appropriate, of the relevant passages</th>
<th>Relevant to claim No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>CN 101256572A (HUAWEI TECHNOLOGIES CO., LTD.) 03 September 2008 (2008-09-03)</td>
<td>11-31</td>
</tr>
<tr>
<td></td>
<td>description, page 3 to page 16</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>CN 101809571A (AMADEUS SAS.) 18 August 2010 (2010-08-18)</td>
<td>11-31</td>
</tr>
<tr>
<td></td>
<td>description, page 2 to page 5 and figures 1 to 5</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>CN 102646127A (INSPUR BEIJING ELECTRONIC INFORMATION INDUSTRY CO., LTD.) 22 August 2012 (2012-08-22)</td>
<td>1-31</td>
</tr>
<tr>
<td></td>
<td>the whole document</td>
<td></td>
</tr>
</tbody>
</table>

Further documents are listed in the continuation of Box C. See patent family annex.

* Special categories of cited documents:
  * 'A' - document defining the general state of the art which is not considered to be of particular relevance
  * 'E' - earlier application or patent but published on or after the international filing date
  * 'L' - document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
  * 'O' - document referring to an oral disclosure, use, exhibition or other means
  * 'P' - document published prior to the international filing date but later than the priority date claimed
  * 'D' - later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
  * 'X' - document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
  * 'Y' - document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
  * 'G' - document member of the same patent family

Date of the actual completion of the international search 11 April 2014

Date of mailing of the international search report 19 May 2014

Name and mailing address of the ISA/Authorized officer

STATE INTELLECTUAL PROPERTY OFFICE OF THE P.R.CHINA/ISA/CN)
6,Xitucheng Rd., Jimen Bridge, Haidian District, Beijing 100088 China

Facsimile No. (86-10)62019451

Telephone No. (86-10)62413661
<table>
<thead>
<tr>
<th>Patent document cited in search report</th>
<th>Publication date (day/month/year)</th>
<th>Patent family member(s)</th>
<th>Publication date (day/month/year)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CN 101256572A</td>
<td>03 September 2008</td>
<td>WO 2009115025A1</td>
<td>24 September 2009</td>
</tr>
<tr>
<td>CN 101809571A</td>
<td>18 August 2010</td>
<td>US 2009083056A1</td>
<td>26 March 2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WO 2009040145A1</td>
<td>02 April 2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KR 20100084520A</td>
<td>26 July 2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EP 2193459A1</td>
<td>09 June 2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CA 2699471A1</td>
<td>02 April 2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JP 2010541037A</td>
<td>24 December 2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AU 2008303807A1</td>
<td>02 April 2009</td>
</tr>
<tr>
<td>CN 102646127A</td>
<td>22 August 2012</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

Form PCT/ISA/210 (patent family annex) (July 2009)