US 20160062833A1

(19) **United States**

(12) **Patent Application Publication**

(10) **Pub. No.: US 2016/0062833 A1**

(43) **Pub. Date: Mar. 3, 2016**

Slik

(54) **REBUILDING A DATA OBJECT USING PORTIONS OF THE DATA OBJECT**

(71) Applicant: **NetApp, Inc.**, Sunnyvale, CA (US)

(72) Inventor: **David Slik**, Northridge, CA (US)

(21) Appl. No.: **14/476,620**

(22) Filed: **Sep. 3, 2014**

Related U.S. Application Data

(63) Continuation of application No. 14/475,376, filed on Sep. 2, 2014.

Publication Classification

(51) **Int. Cl.**

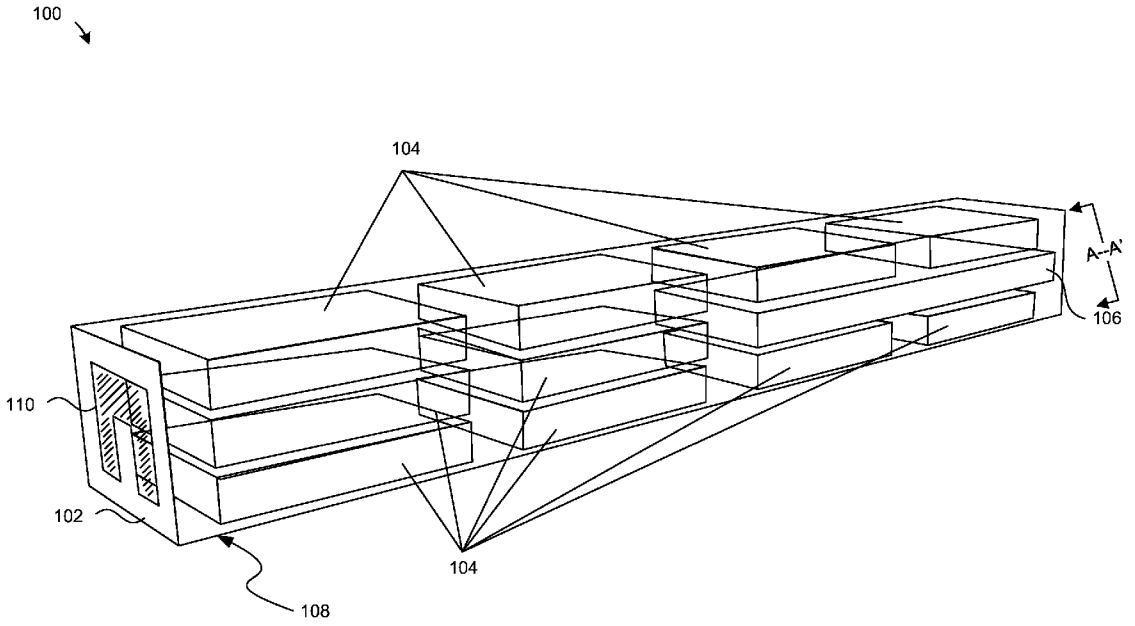
G06F 11/10	(2006.01)
G06F 3/06	(2006.01)
G06F 17/30	(2006.01)

(52) **U.S. Cl.**

CPC ***G06F 11/1076*** (2013.01); ***G06F 17/30312*** (2013.01); ***G06F 3/0619*** (2013.01); ***G06F 3/064*** (2013.01); ***G06F 3/0683*** (2013.01)

(57) **ABSTRACT**

Technology is disclosed for a data storage architecture for providing enhanced storage resiliency for a data object. The data storage architecture can be implemented in a single-tier configuration and/or a multi-tier configuration. In the single-tier configuration, a data object is encoded, e.g., based on an erasure coding method, to generate many data fragments, which are stored across many storage devices. In the multi-tier configuration, a data object is encoded, e.g., based on an erasure coding method, to generate many data segments, which are sent to one or more tiers of storage nodes. Each of the storage nodes further encodes the data segment to generate many data fragments representing the data segment, which are stored across many storage devices associated with the storage node. The I/O operations for rebuilding the data in case of device failures is spread across many storage devices, which minimizes the wear of a given storage device.



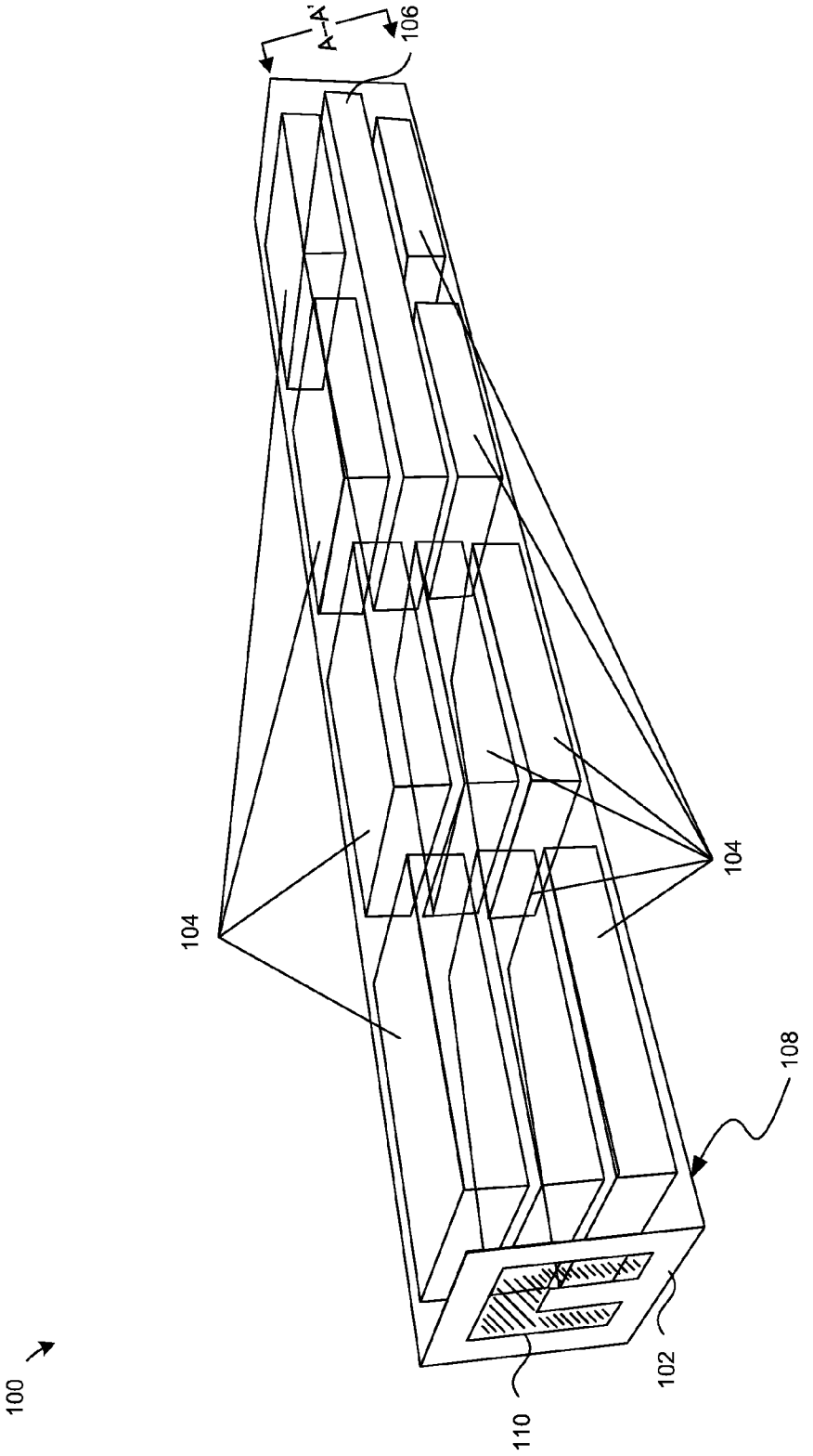


FIG. 1A

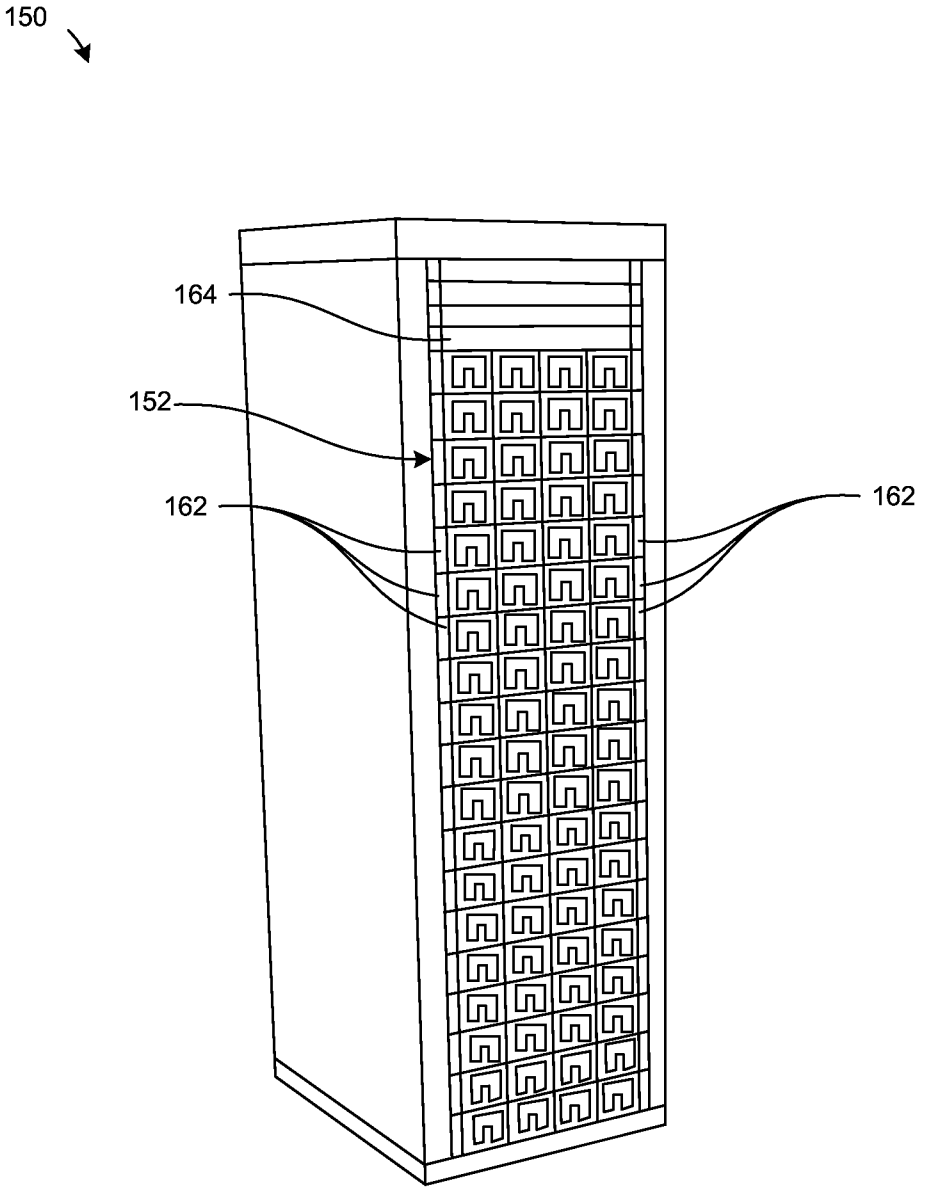


FIG. 1B

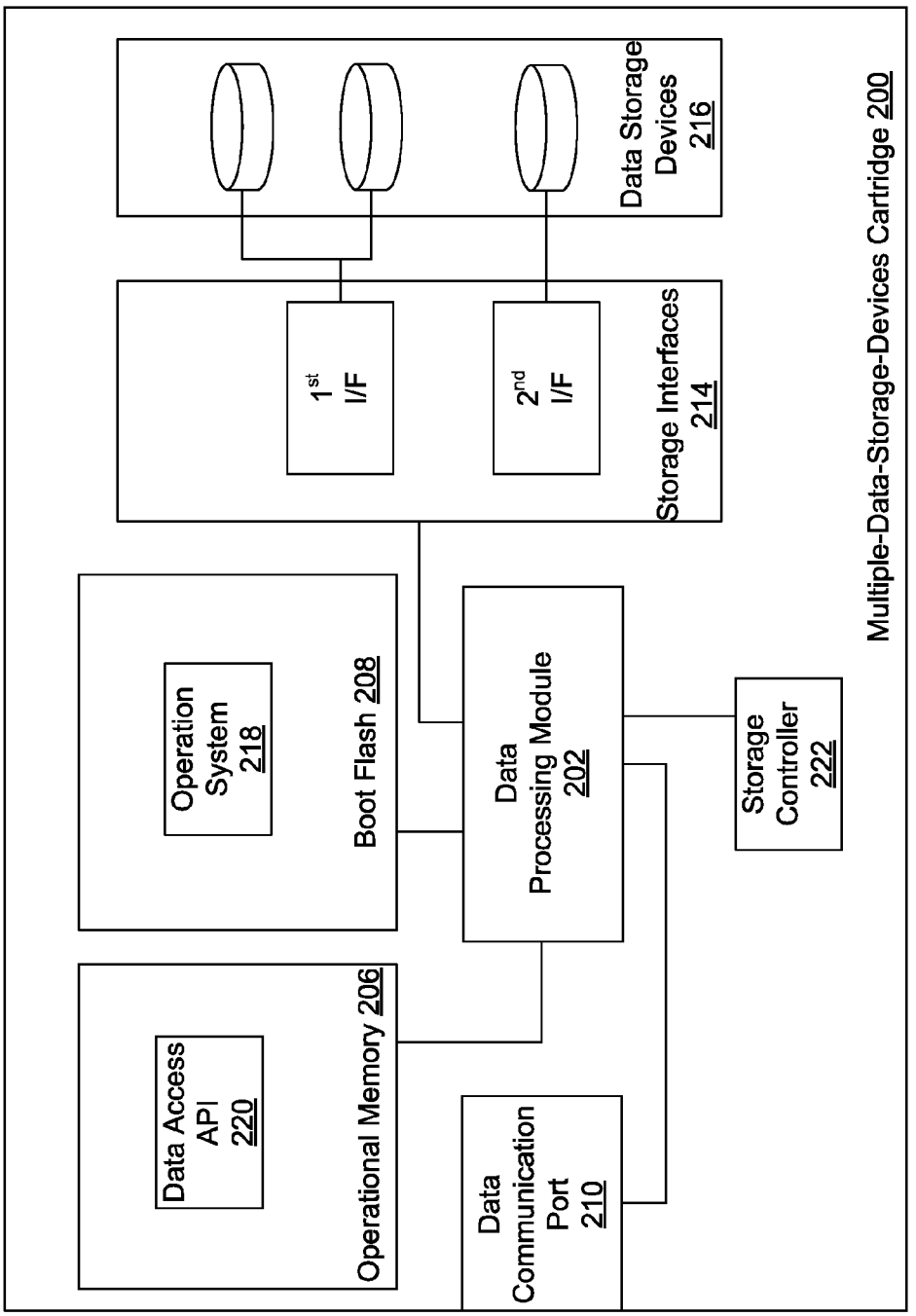

Multiple-Data-Storage-Devices Cartridge 200
Data Storage Devices 216
Storage Interfaces 214
1st I/F
2nd I/F
Boot Flash 208
Operation System 218
Data Processing Module 202
Storage Controller 222
Operational Memory 206
Data Access API 220
Data Communication Port 210

FIG. 2

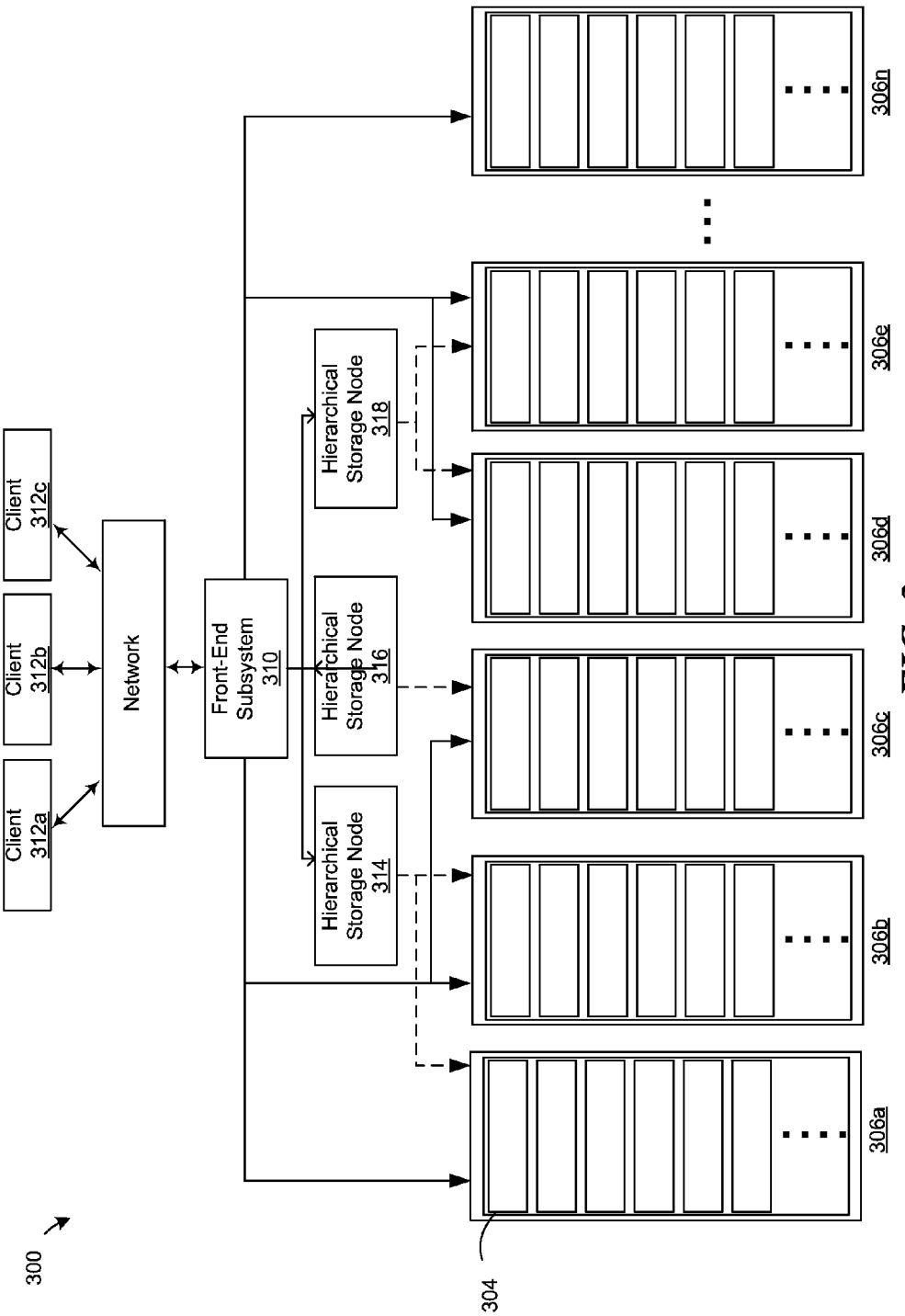

300
Client 312a
Client 312b
Client 312c
Network
Front-End Subsystem 310
Hierarchical Storage Node 314
Hierarchical Storage Node 316
Hierarchical Storage Node 318
304
306a
306b
306c
306d
306e
306n

FIG. 3

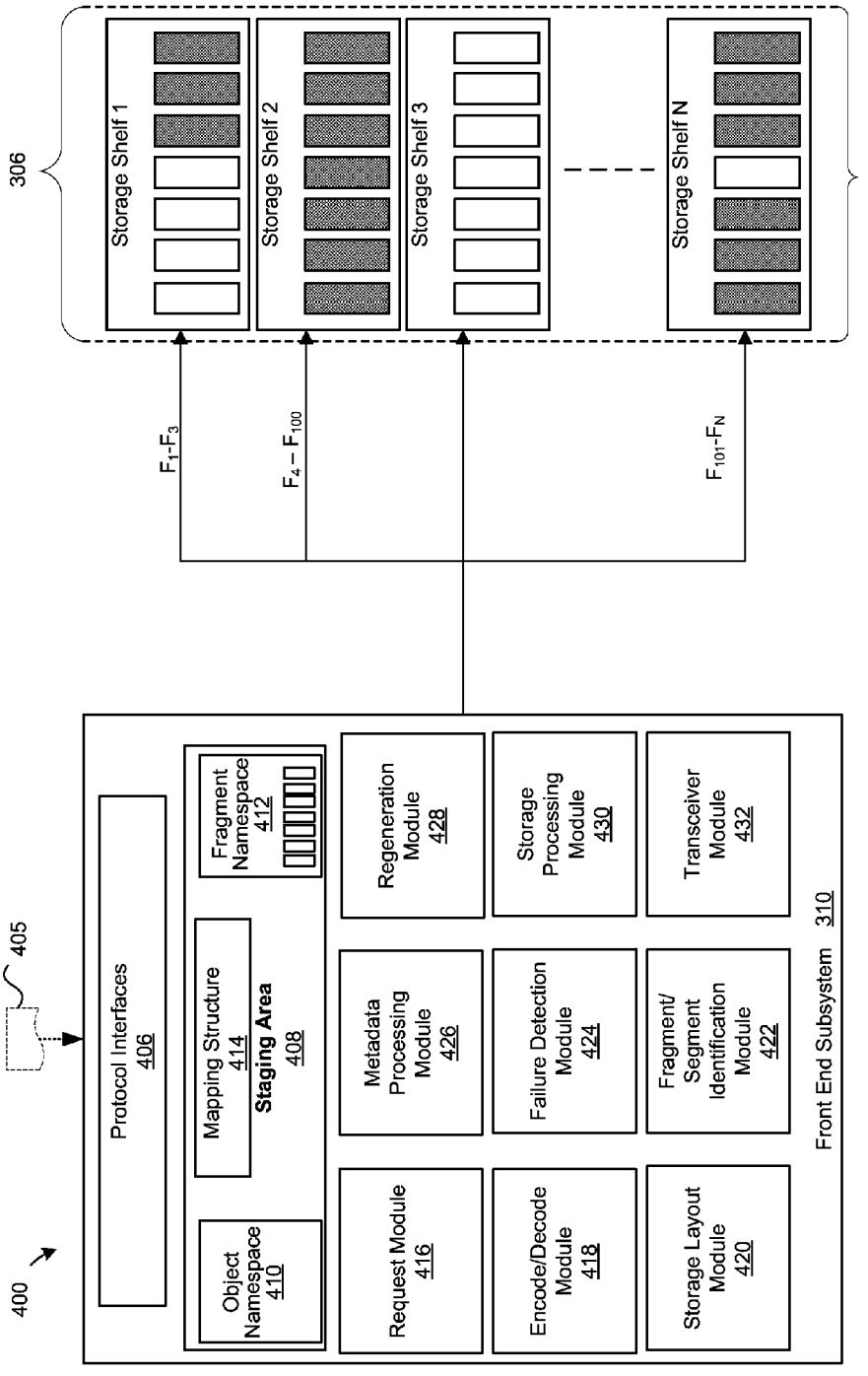

306
Storage Shelf 1
Storage Shelf 2
Storage Shelf 3
Storage Shelf N
F1-F3
F4 – F100
F101-FN
400
405
Protocol Interfaces 406
Object Namespace 410
Mapping Structure 414
Staging Area 408
Fragment Namespace 412
Request Module 416
Metadata Processing Module 426
Regeneration Module 428
Encode/Decode Module 418
Failure Detection Module 424
Storage Processing Module 430
Storage Layout Module 420
Fragment/ Segment Identification Module 422
Transceiver Module 432
Front End Subsystem 310

FIG. 4

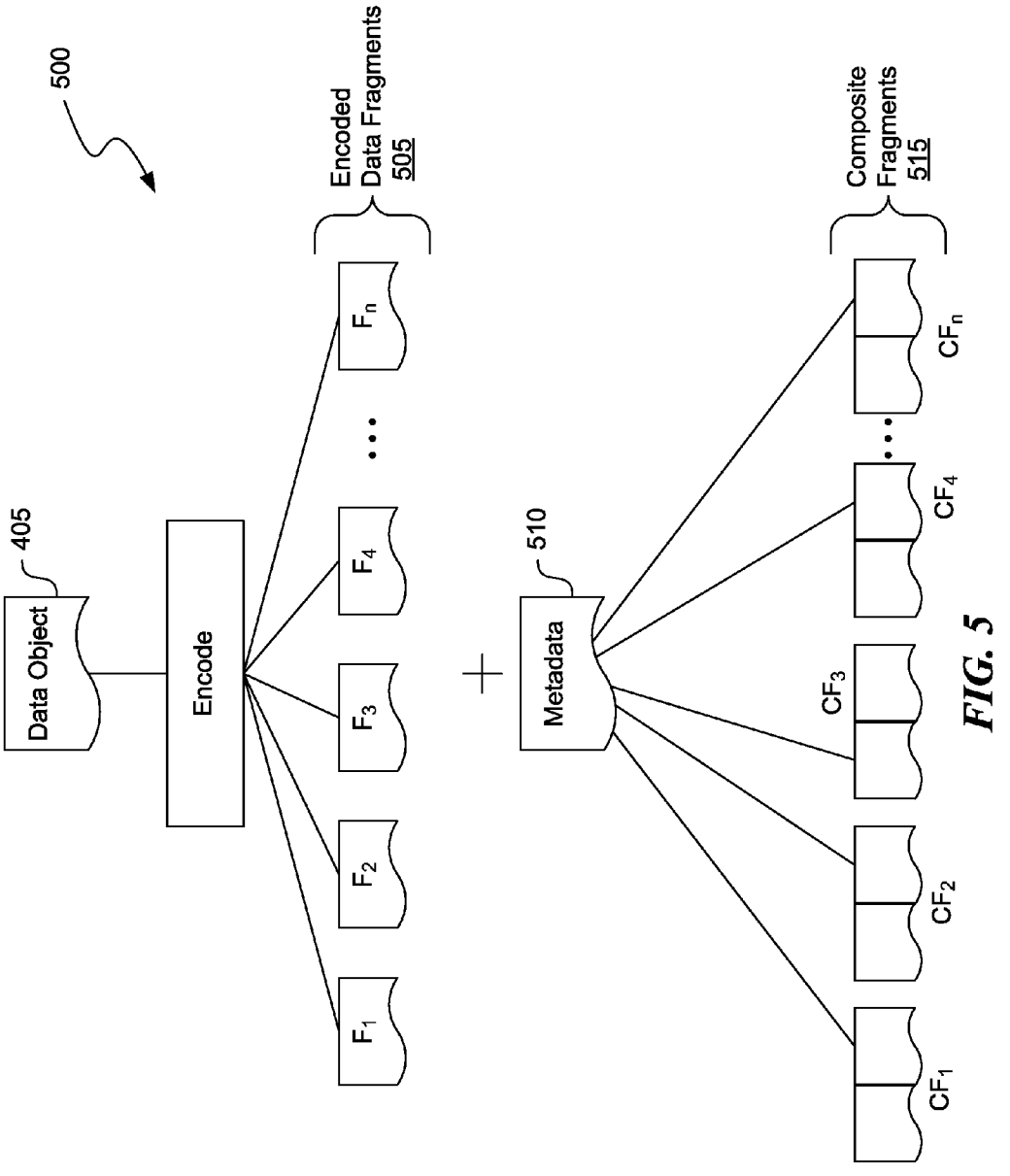

500
Encoded Data Fragments 505
Composite Fragments 515
405
Data Object
Encode
F_1
F_2
F_3
F_4
F_n
510
Metadata
CF_1
CF_2
CF_3
CF_4
CF_n

FIG. 5

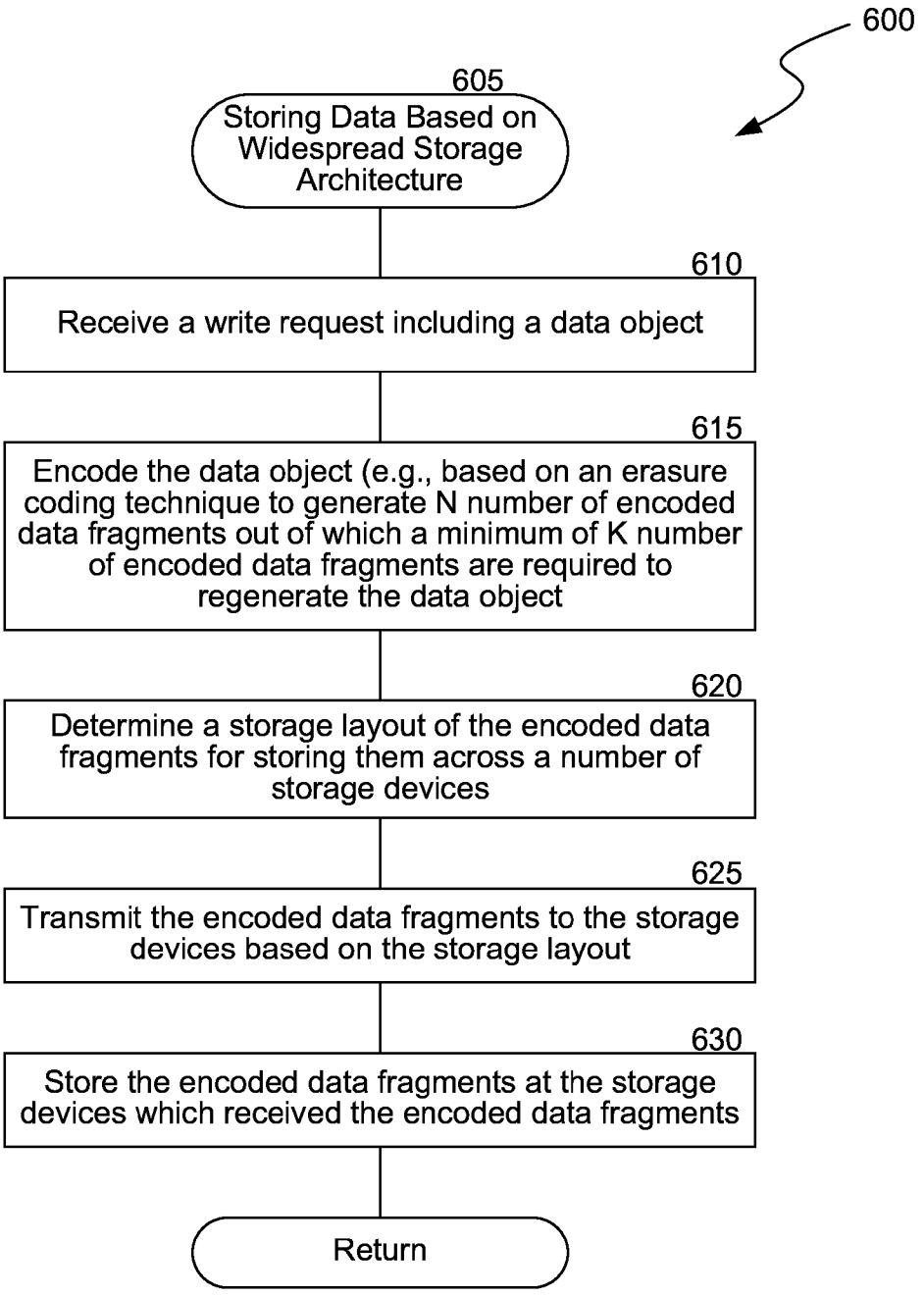

600
605
Storing Data Based on Widespread Storage Architecture
610
Receive a write request including a data object
615
Encode the data object (e.g., based on an erasure coding technique to generate N number of encoded data fragments out of which a minimum of K number of encoded data fragments are required to regenerate the data object
620
Determine a storage layout of the encoded data fragments for storing them across a number of storage devices
625
Transmit the encoded data fragments to the storage devices based on the storage layout
630
Store the encoded data fragments at the storage devices which received the encoded data fragments
Return

FIG. 6

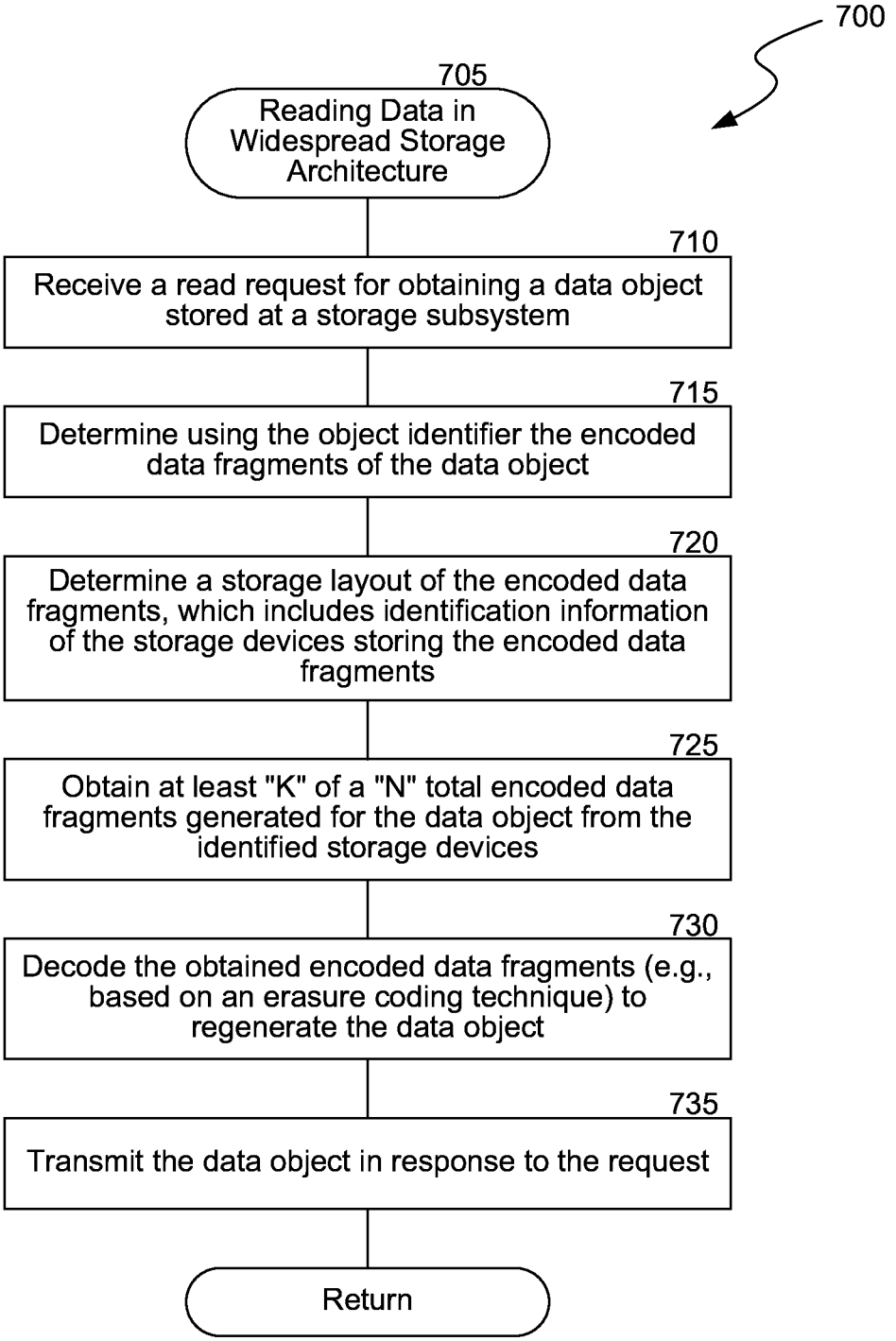

700
705
Reading Data in Widespread Storage Architecture
710
Receive a read request for obtaining a data object stored at a storage subsystem
715
Determine using the object identifier the encoded data fragments of the data object
720
Determine a storage layout of the encoded data fragments, which includes identification information of the storage devices storing the encoded data fragments
725
Obtain at least "K" of a "N" total encoded data fragments generated for the data object from the identified storage devices
730
Decode the obtained encoded data fragments (e.g., based on an erasure coding technique) to regenerate the data object
735
Transmit the data object in response to the request
Return

FIG. 7

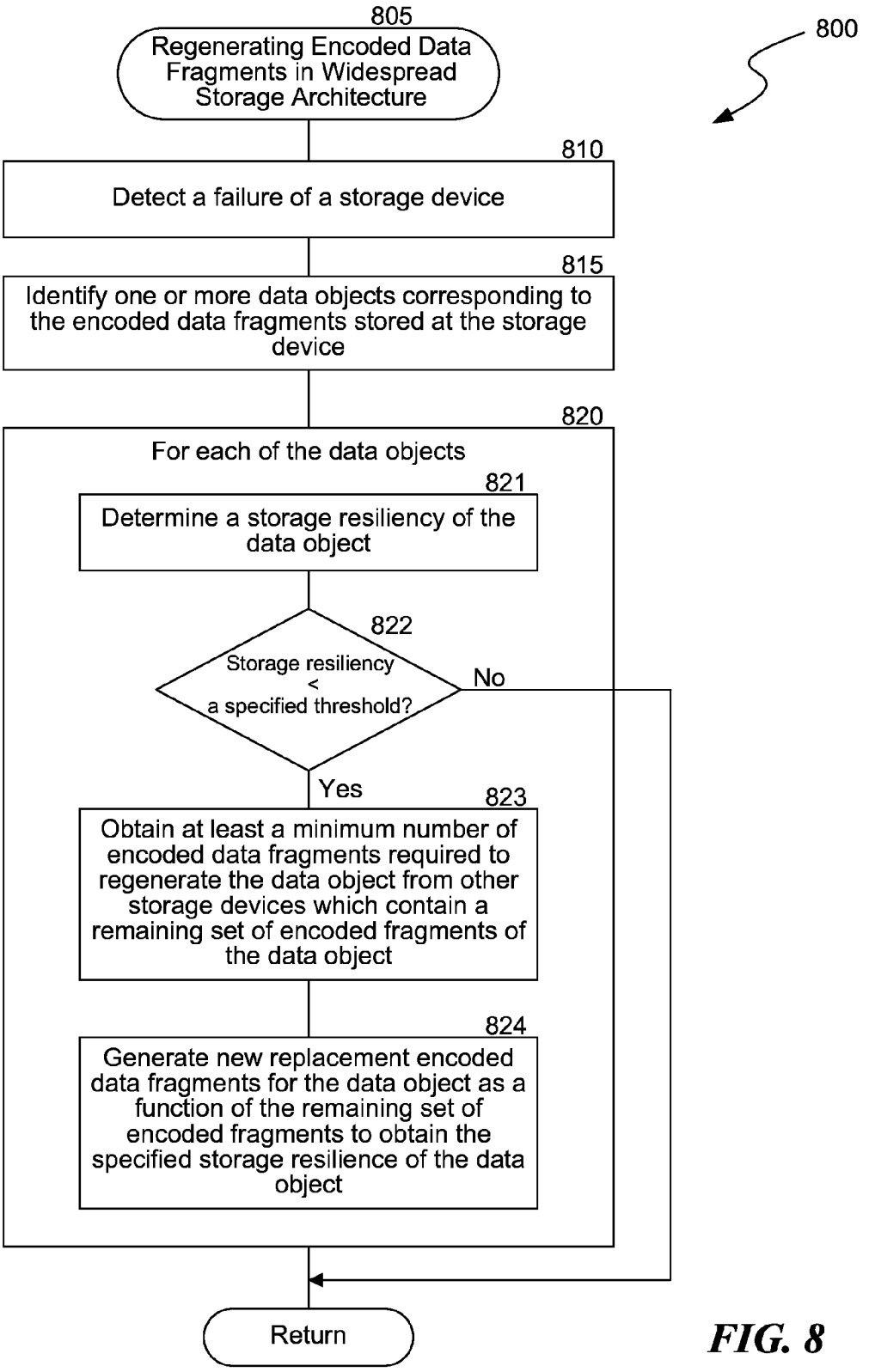

800
805
Regenerating Encoded Data Fragments in Widespread Storage Architecture
810
Detect a failure of a storage device
815
Identify one or more data objects corresponding to the encoded data fragments stored at the storage device
820
For each of the data objects
821
Determine a storage resiliency of the data object
822
Storage resiliency < a specified threshold?
No
Yes
823
Obtain at least a minimum number of encoded data fragments required to regenerate the data object from other storage devices which contain a remaining set of encoded fragments of the data object
824
Generate new replacement encoded data fragments for the data object as a function of the remaining set of encoded fragments to obtain the specified storage resilience of the data object
Return

FIG. 8

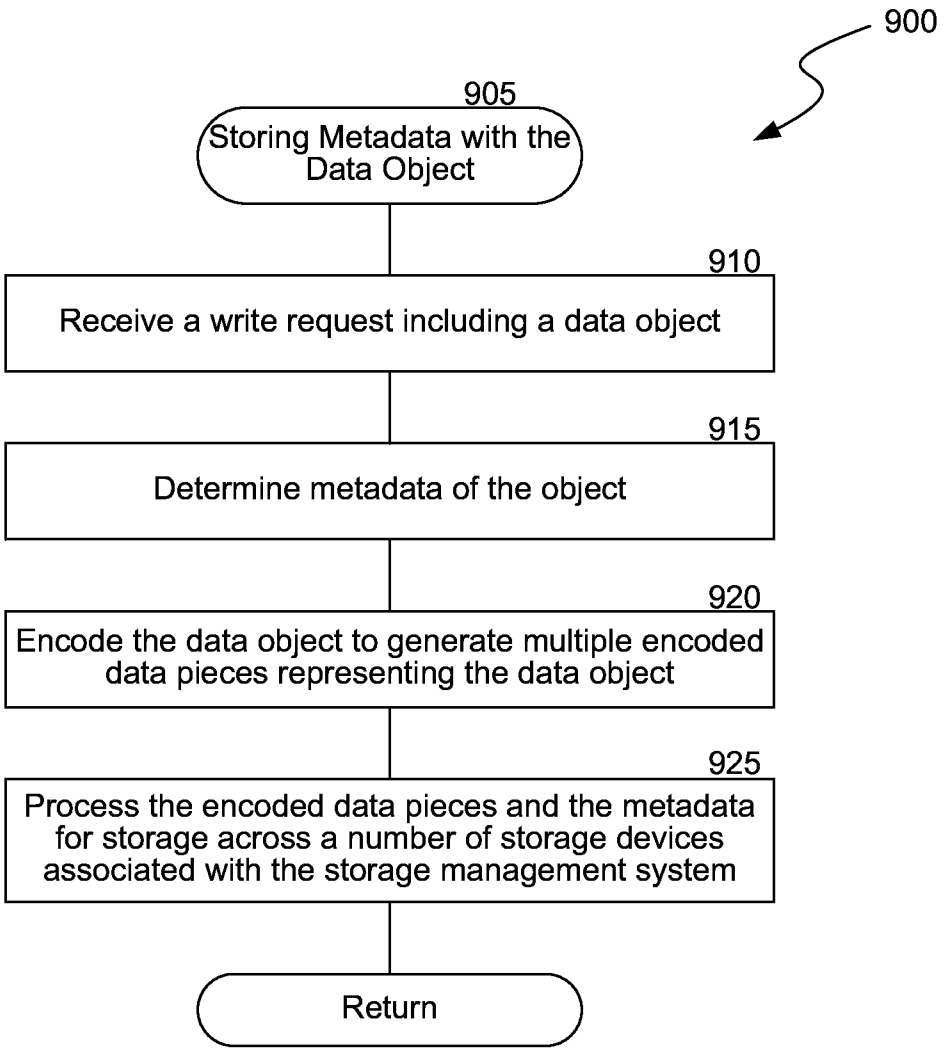

900
905
Storing Metadata with the Data Object
910
Receive a write request including a data object
915
Determine metadata of the object
920
Encode the data object to generate multiple encoded data pieces representing the data object
925
Process the encoded data pieces and the metadata for storage across a number of storage devices associated with the storage management system
Return

FIG. 9

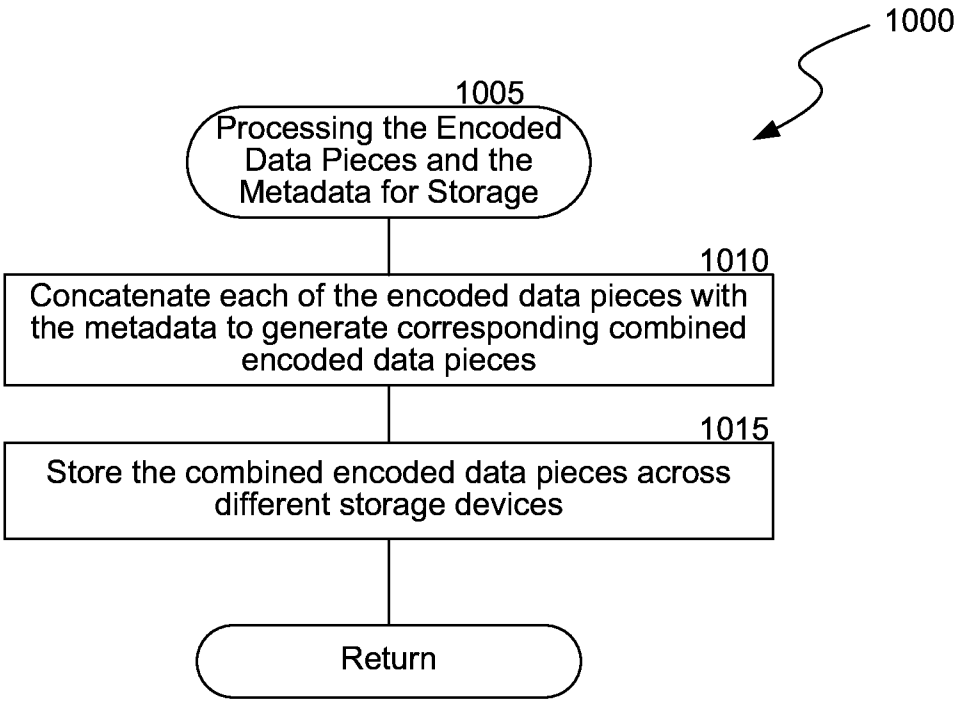

1000
1005
Processing the Encoded Data Pieces and the Metadata for Storage
1010
Concatenate each of the encoded data pieces with the metadata to generate corresponding combined encoded data pieces
1015
Store the combined encoded data pieces across different storage devices
Return

FIG. 10

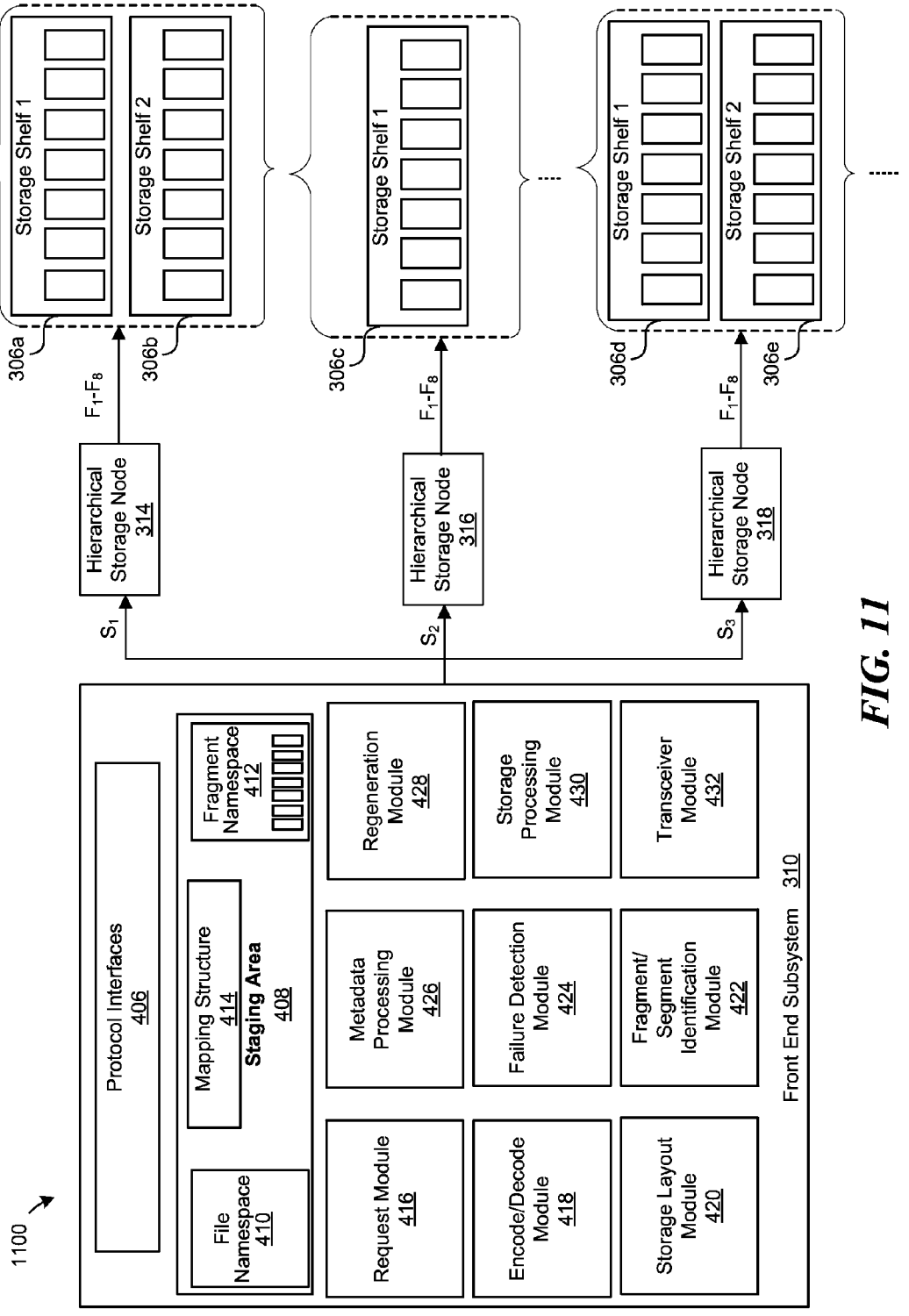

1100
Front End Subsystem 310
Protocol Interfaces 406
File Namespace 410
Staging Area 408
Mapping Structure 414
Fragment Namespace 412
Request Module 416
Encode/Decode Module 418
Storage Layout Module 420
Metadata Processing Module 426
Failure Detection Module 424
Fragment/ Segment Identification Module 422
Regeneration Module 428
Storage Processing Module 430
Transceiver Module 432
Hierarchical Storage Node 314
Hierarchical Storage Node 316
Hierarchical Storage Node 318
S1
S2
S3
F1-F8
306a
306b
306c
306d
306e
Storage Shelf 1
Storage Shelf 2

FIG. 11

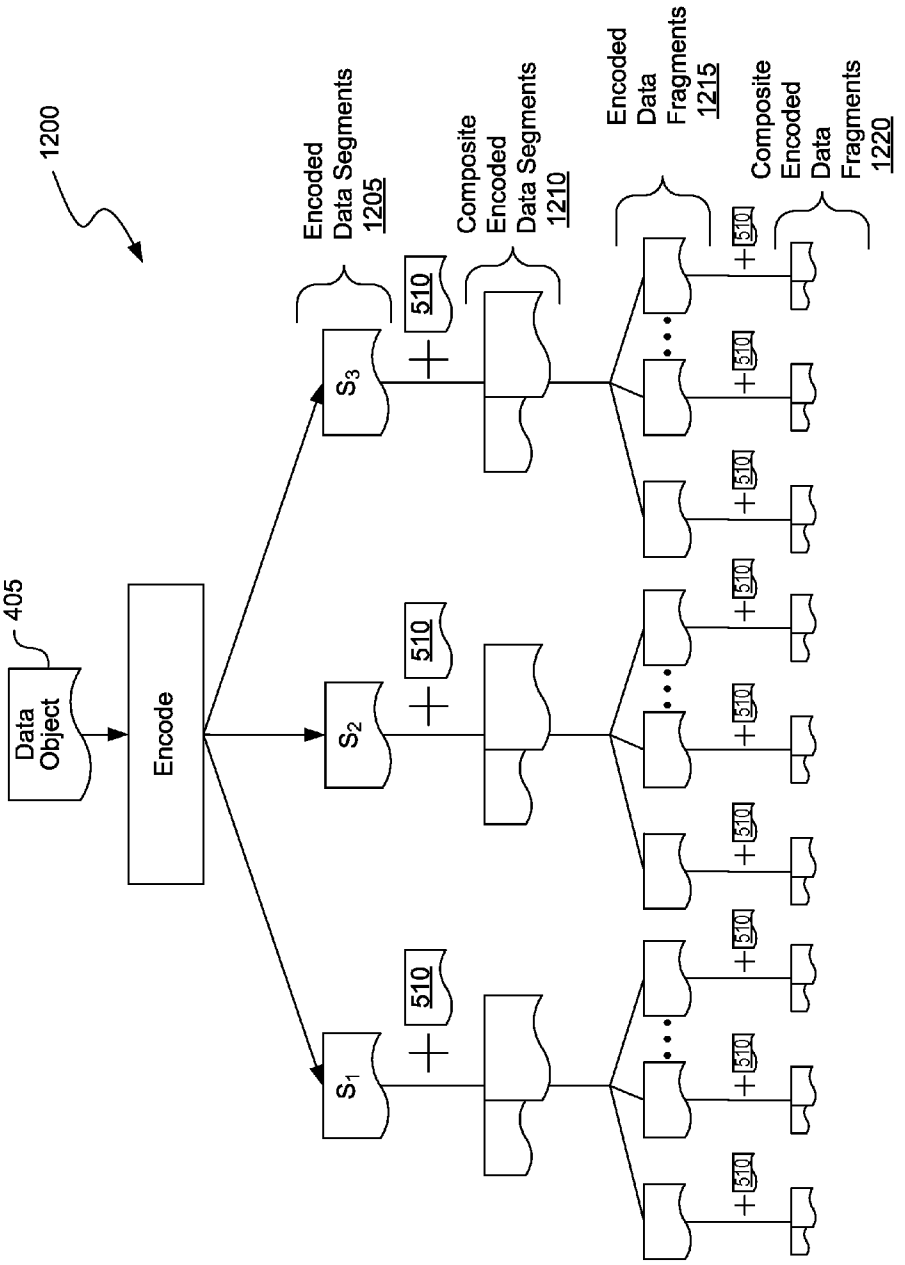

1200
405
Data Object
Encode
S_1
S_2
S_3
510
Encoded Data Segments 1205
Composite Encoded Data Segments 1210
Encoded Data Fragments 1215
Composite Encoded Data Fragments 1220

FIG. 12

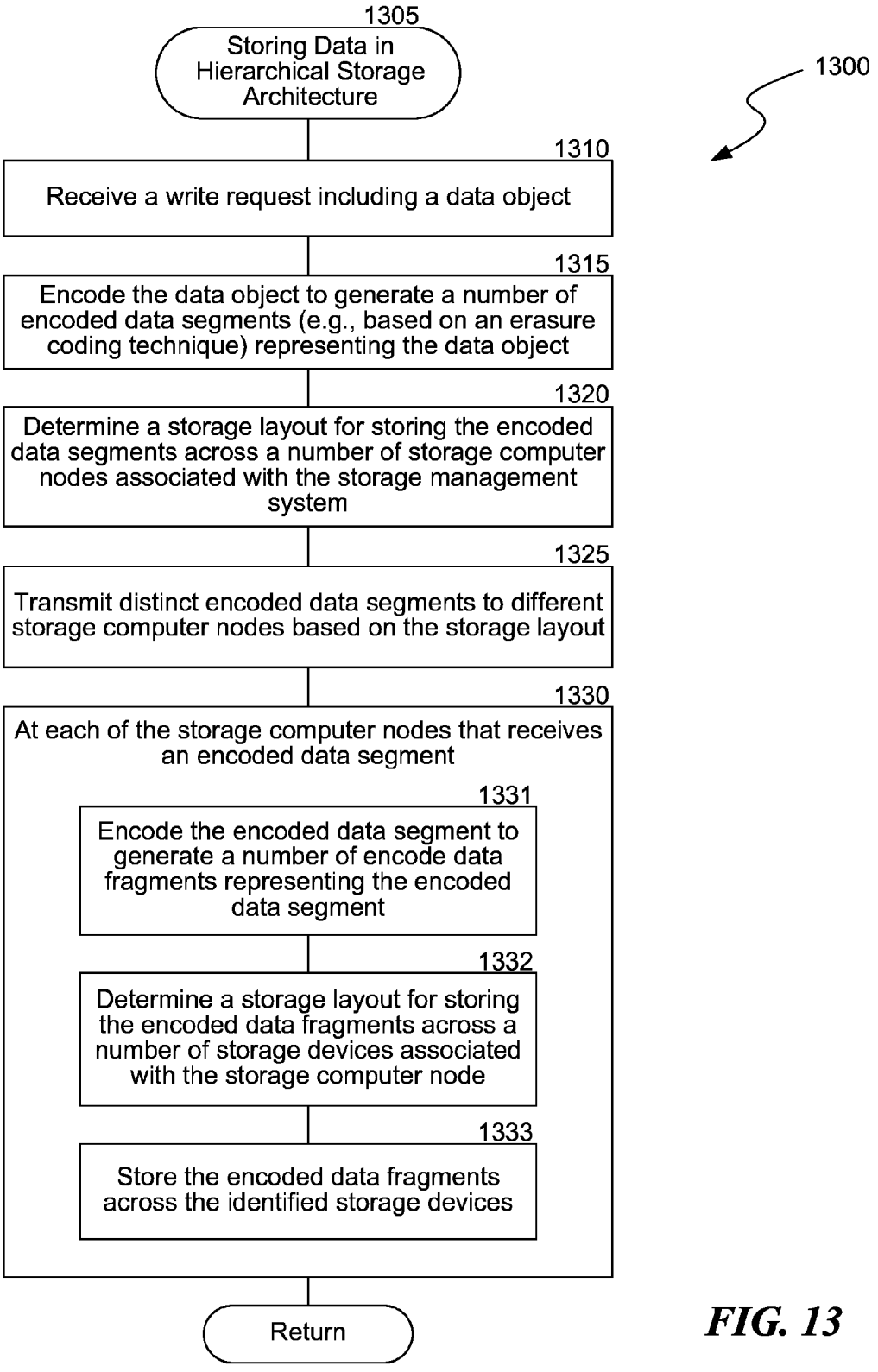

1305
Storing Data in Hierarchical Storage Architecture
1300
1310
Receive a write request including a data object
1315
Encode the data object to generate a number of encoded data segments (e.g., based on an erasure coding technique) representing the data object
1320
Determine a storage layout for storing the encoded data segments across a number of storage computer nodes associated with the storage management system
1325
Transmit distinct encoded data segments to different storage computer nodes based on the storage layout
1330
At each of the storage computer nodes that receives an encoded data segment
1331
Encode the encoded data segment to generate a number of encode data fragments representing the encoded data segment
1332
Determine a storage layout for storing the encoded data fragments across a number of storage devices associated with the storage computer node
1333
Store the encoded data fragments across the identified storage devices
Return

FIG. 13

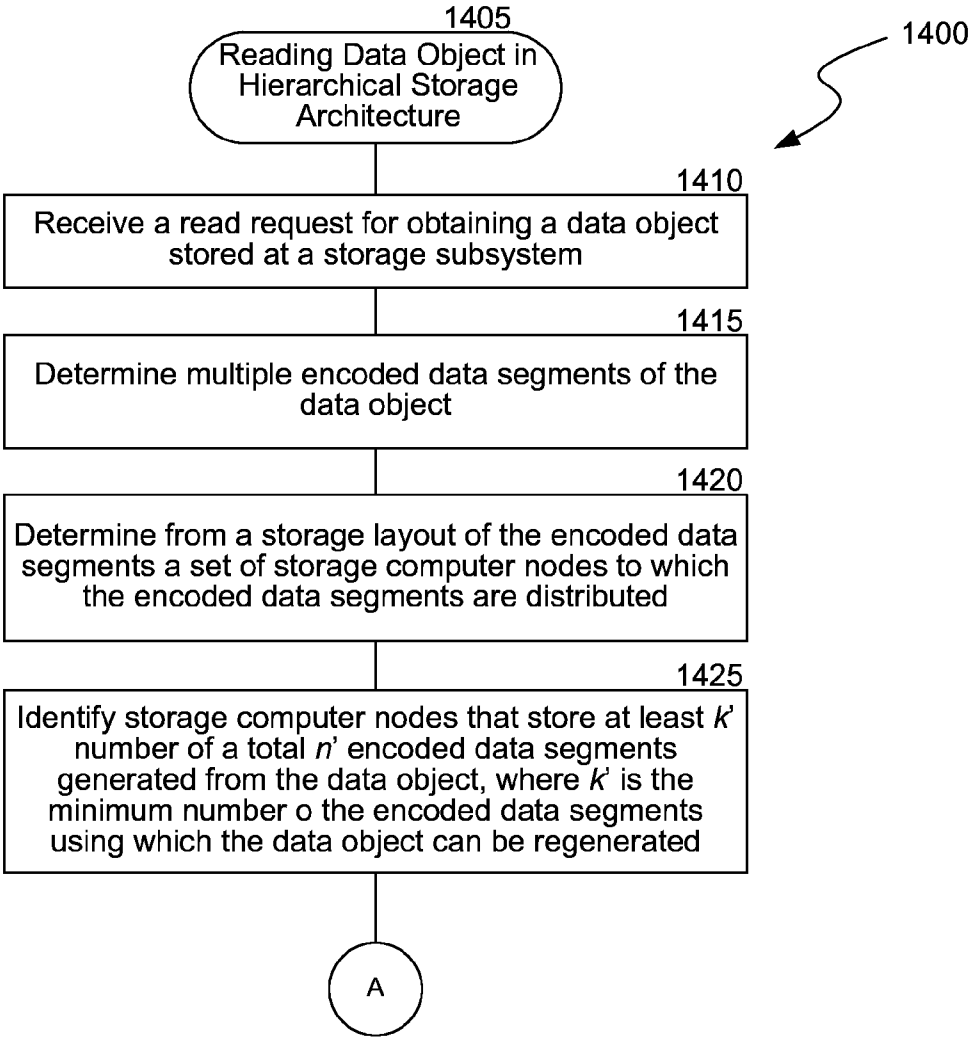

1405
Reading Data Object in Hierarchical Storage Architecture
1400
1410
Receive a read request for obtaining a data object stored at a storage subsystem
1415
Determine multiple encoded data segments of the data object
1420
Determine from a storage layout of the encoded data segments a set of storage computer nodes to which the encoded data segments are distributed
1425
Identify storage computer nodes that store at least k' number of a total n' encoded data segments generated from the data object, where k' is the minimum number o the encoded data segments using which the data object can be regenerated
A

FIG. 14

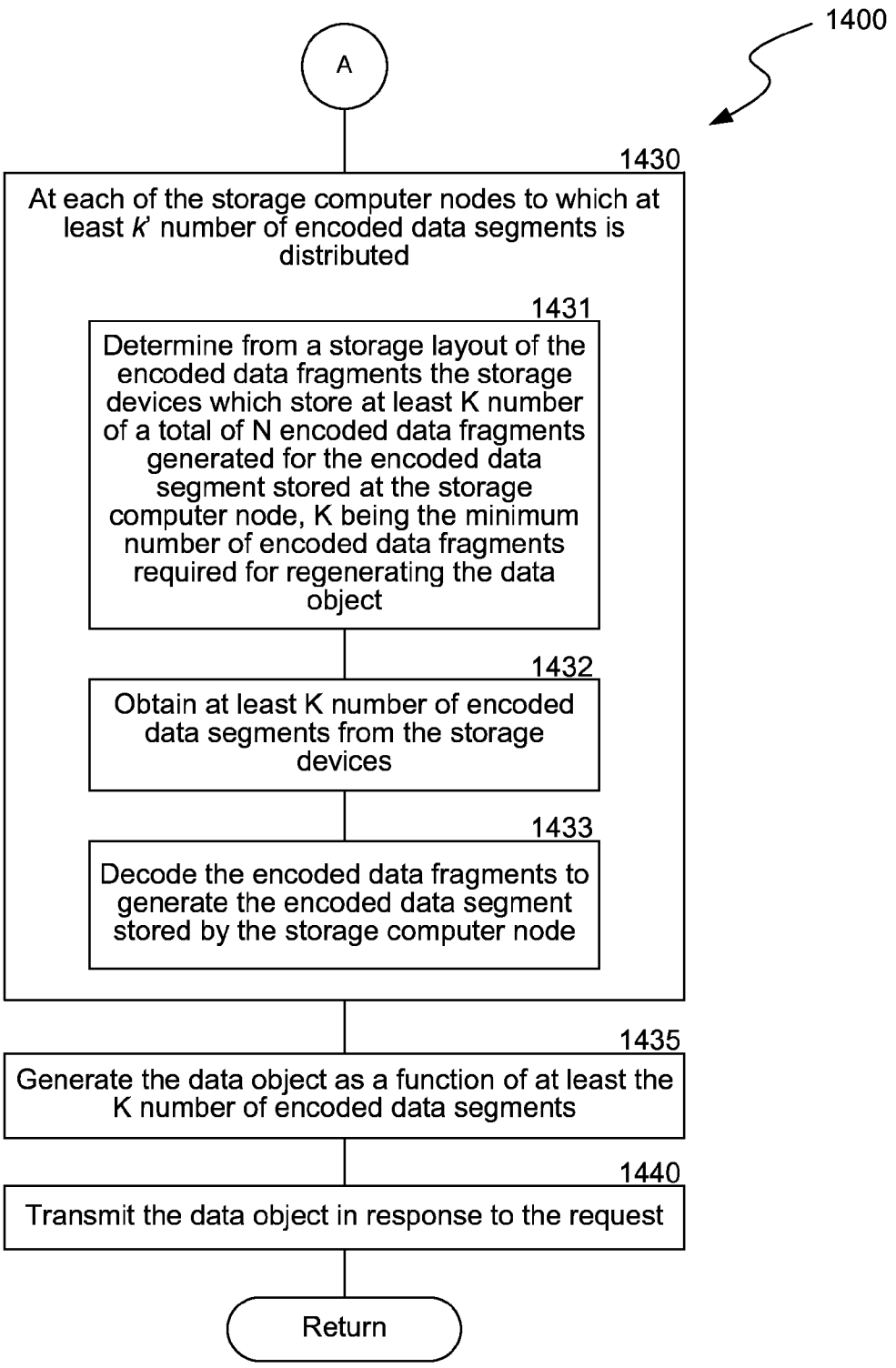


FIG. 14 (Continued)

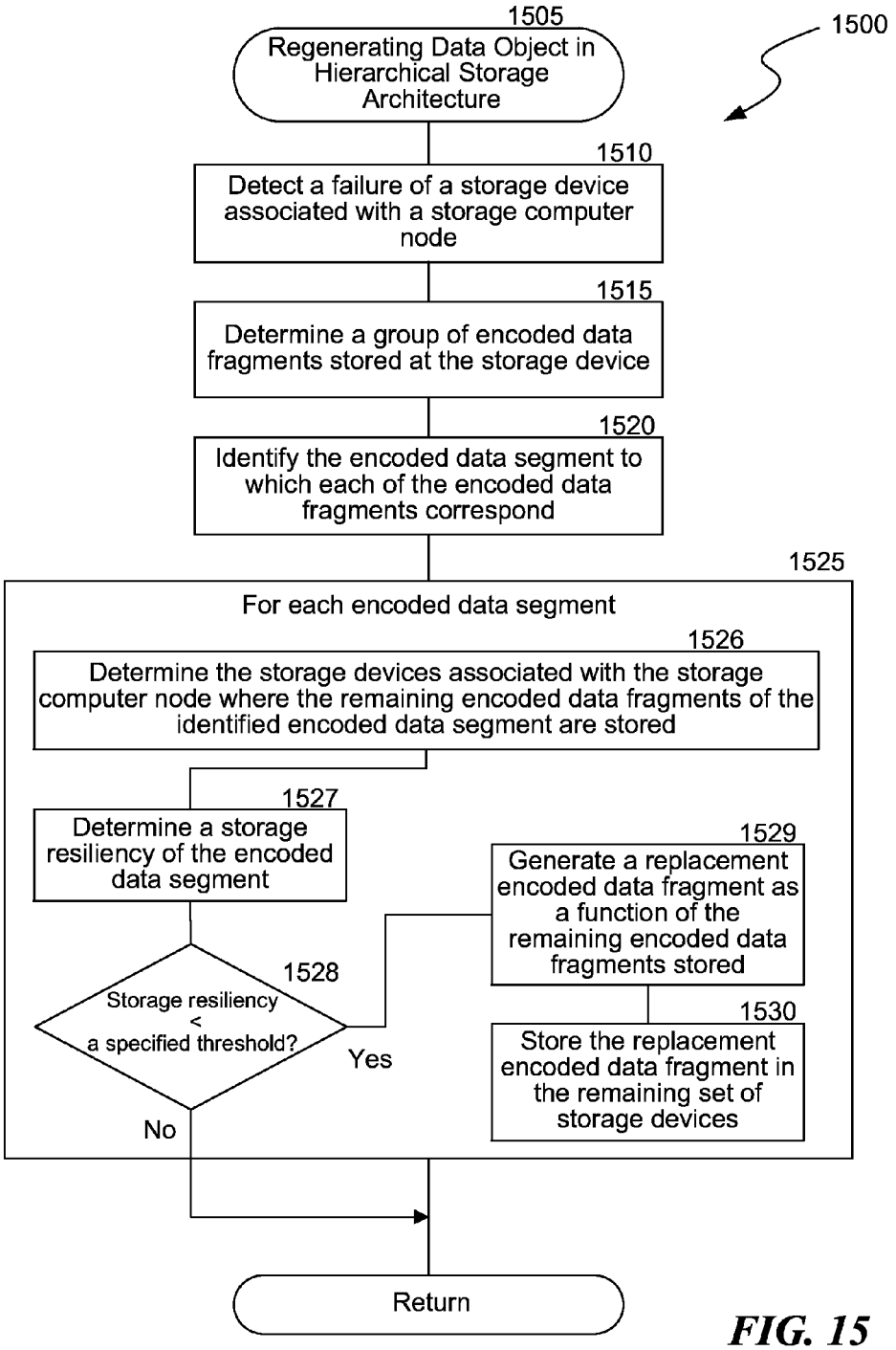

1505
Regenerating Data Object in Hierarchical Storage Architecture
1500
1510
Detect a failure of a storage device associated with a storage computer node
1515
Determine a group of encoded data fragments stored at the storage device
1520
Identify the encoded data segment to which each of the encoded data fragments correspond
1525
For each encoded data segment
1526
Determine the storage devices associated with the storage computer node where the remaining encoded data fragments of the identified encoded data segment are stored
1527
Determine a storage resiliency of the encoded data segment
1528
Storage resiliency < a specified threshold?
Yes
No
1529
Generate a replacement encoded data fragment as a function of the remaining encoded data fragments stored
1530
Store the replacement encoded data fragment in the remaining set of storage devices
Return

FIG. 15

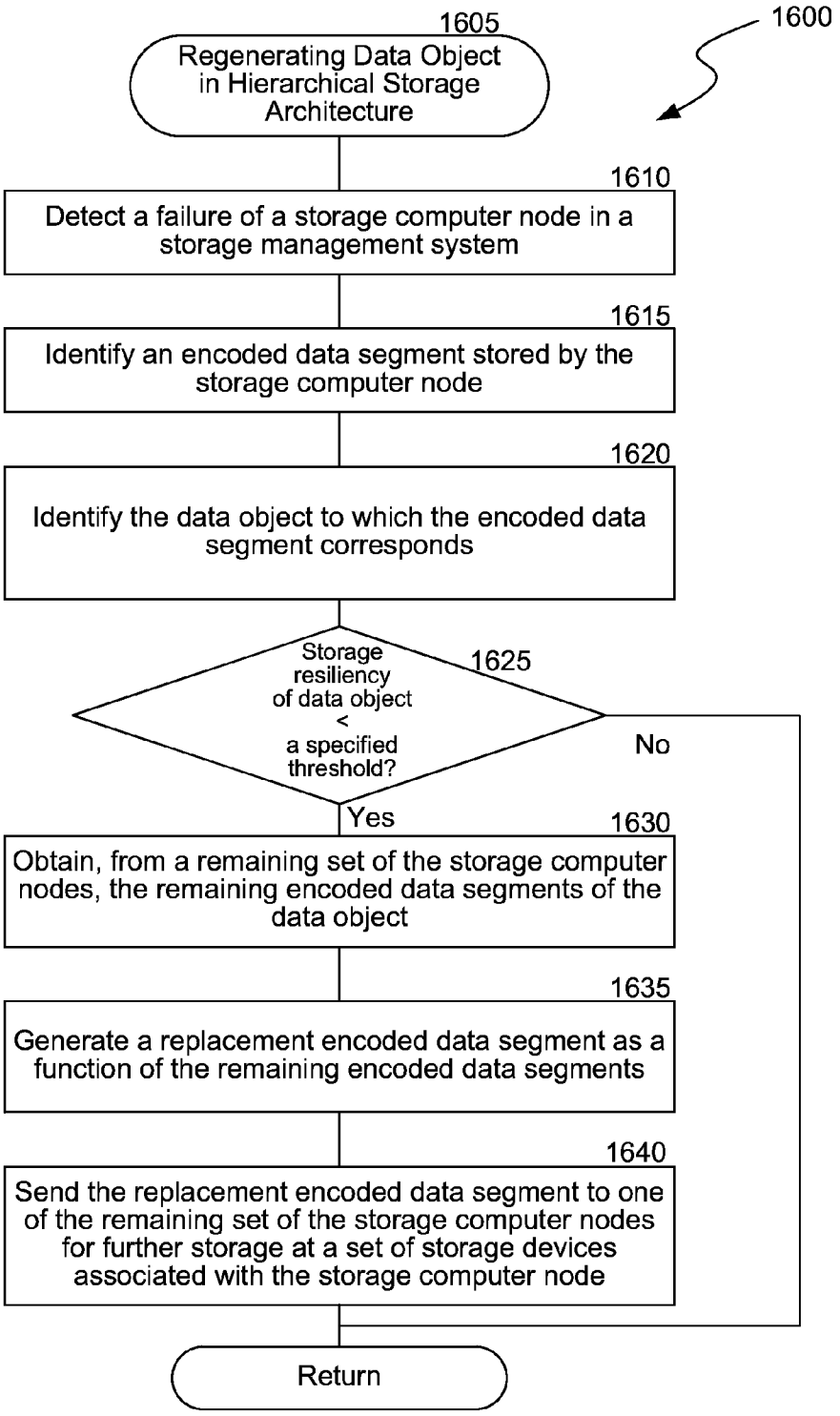

1600
1605
Regenerating Data Object in Hierarchical Storage Architecture
1610
Detect a failure of a storage computer node in a storage management system
1615
Identify an encoded data segment stored by the storage computer node
1620
Identify the data object to which the encoded data segment corresponds
1625
Storage resiliency of data object < a specified threshold?
No
Yes
1630
Obtain, from a remaining set of the storage computer nodes, the remaining encoded data segments of the data object
1635
Generate a replacement encoded data segment as a function of the remaining encoded data segments
1640
Send the replacement encoded data segment to one of the remaining set of the storage computer nodes for further storage at a set of storage devices associated with the storage computer node
Return

FIG. 16

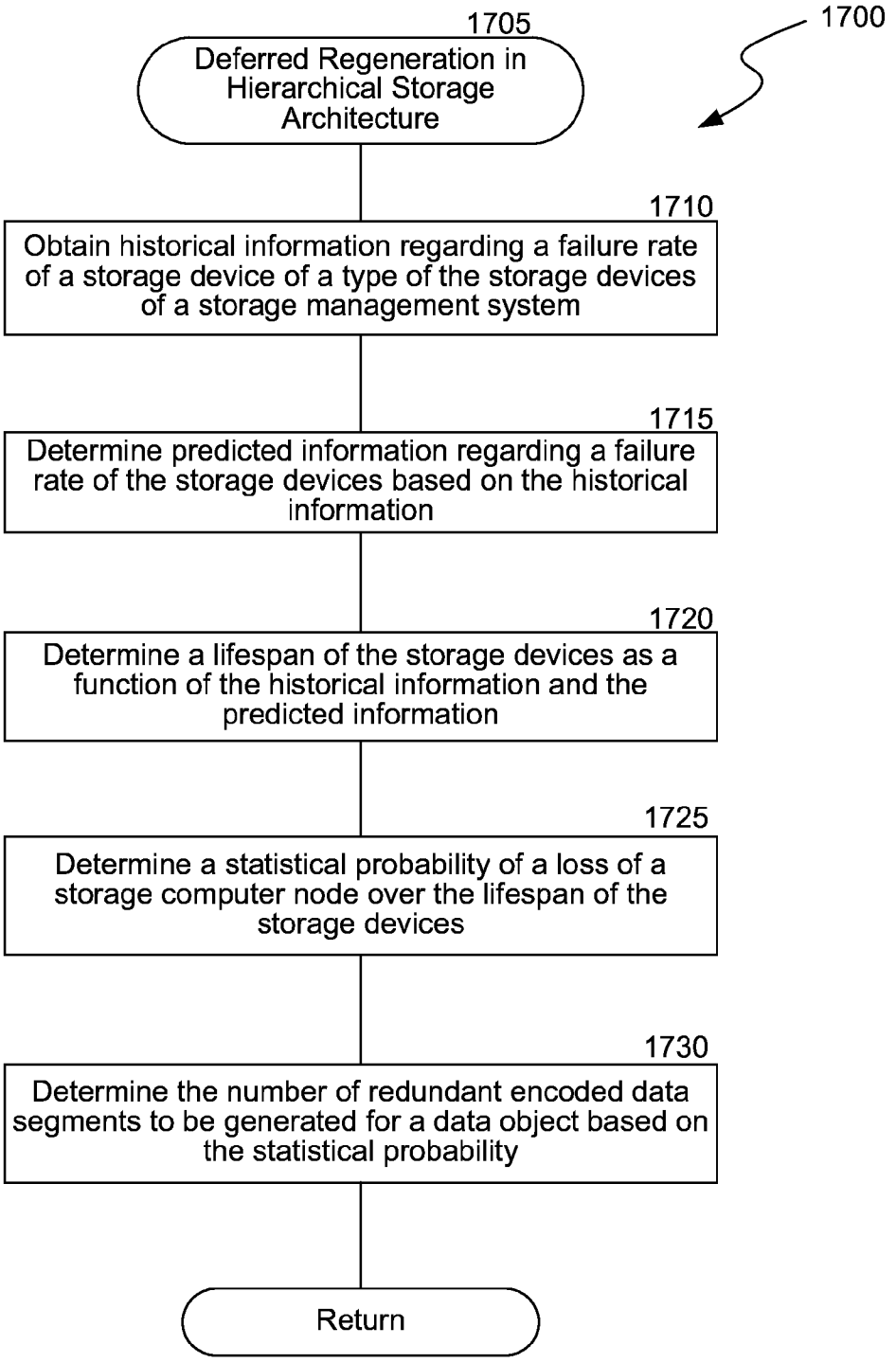

1700
1705
Deferred Regeneration in Hierarchical Storage Architecture
1710
Obtain historical information regarding a failure rate of a storage device of a type of the storage devices of a storage management system
1715
Determine predicted information regarding a failure rate of the storage devices based on the historical information
1720
Determine a lifespan of the storage devices as a function of the historical information and the predicted information
1725
Determine a statistical probability of a loss of a storage computer node over the lifespan of the storage devices
1730
Determine the number of redundant encoded data segments to be generated for a data object based on the statistical probability
Return

FIG. 17

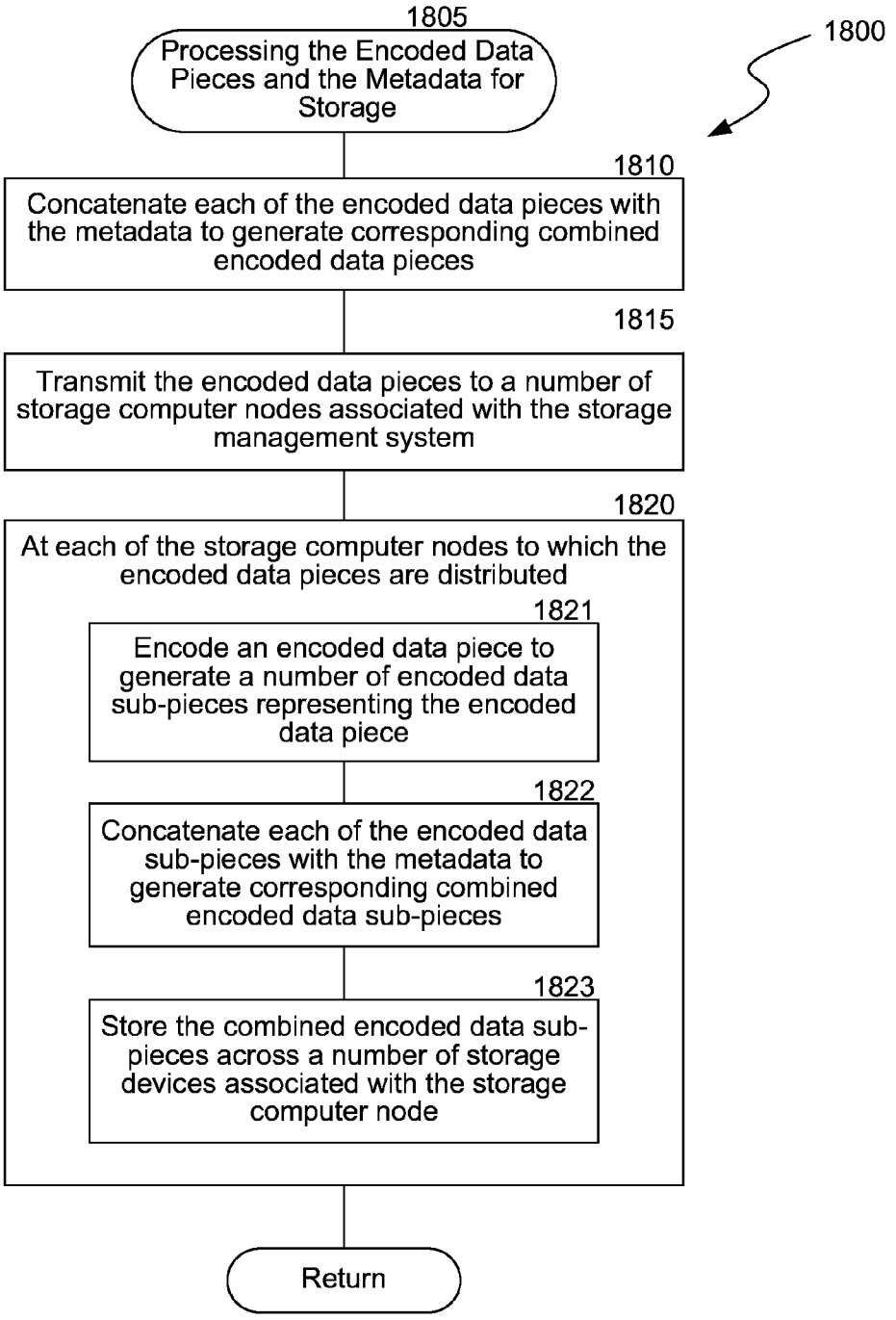

1805
Processing the Encoded Data Pieces and the Metadata for Storage
1800
1810
Concatenate each of the encoded data pieces with the metadata to generate corresponding combined encoded data pieces
1815
Transmit the encoded data pieces to a number of storage computer nodes associated with the storage management system
1820
At each of the storage computer nodes to which the encoded data pieces are distributed
1821
Encode an encoded data piece to generate a number of encoded data sub-pieces representing the encoded data piece
1822
Concatenate each of the encoded data sub-pieces with the metadata to generate corresponding combined encoded data sub-pieces
1823
Store the combined encoded data sub-pieces across a number of storage devices associated with the storage computer node
Return

FIG. 18

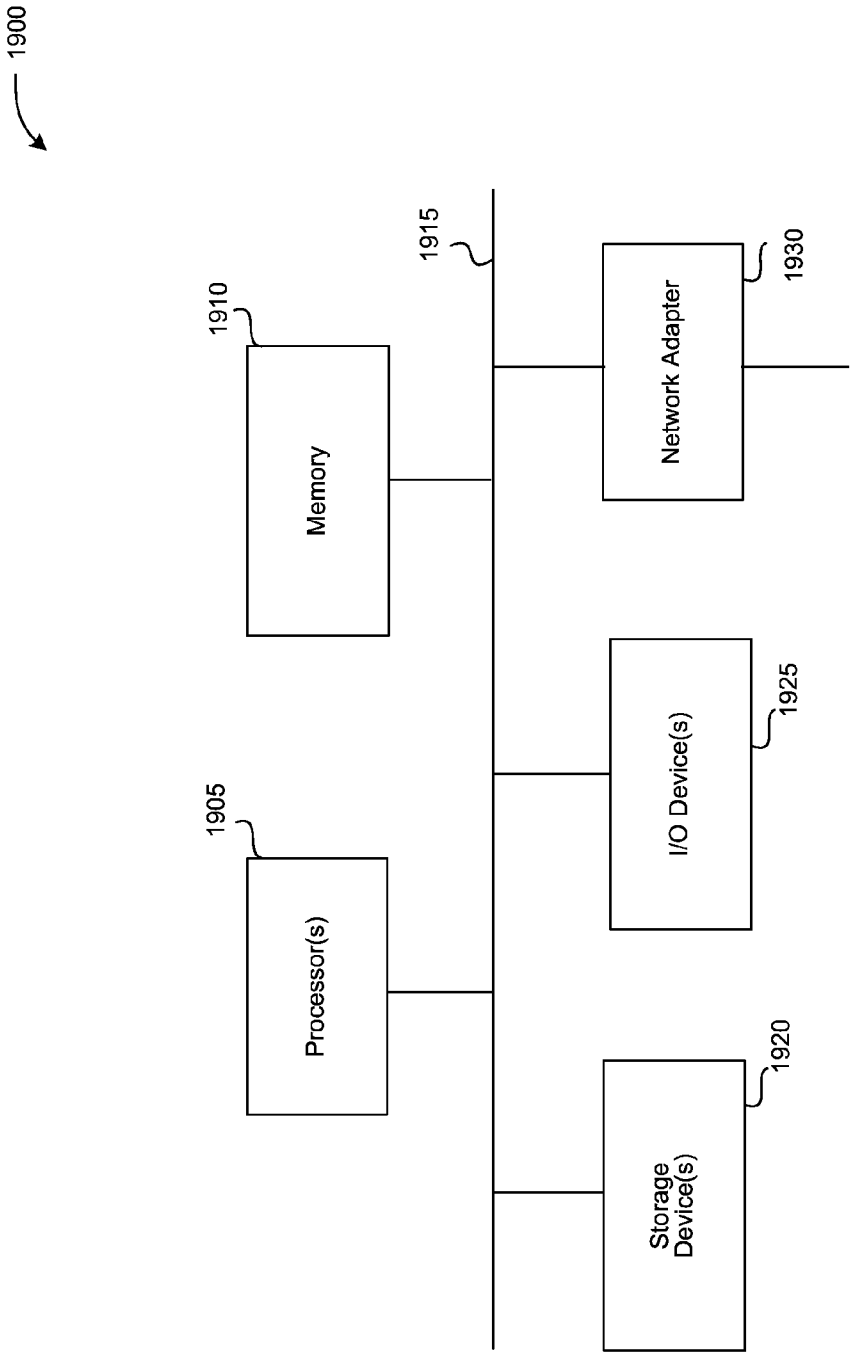

1900
1915
1910
Memory
Network Adapter
1930
1905
Processor(s)
I/O Device(s)
1925
Storage Device(s)
1920

FIG. 19

REBUILDING A DATA OBJECT USING PORTIONS OF THE DATA OBJECT

CROSS-REFERENCE TO RELATED APPLICATION(S)

[0001] This application is a continuation of U.S. patent application Ser. No. 14/475,376, entitled "WIDE SPREADING DATA STORAGE ARCHITECTURE", filed on Sep. 2, 2014, which is incorporated by reference herein in its entirety.

TECHNICAL FIELD

[0002] Several of the disclosed embodiments relate to data storage, and more particularly, to data storage architecture for enhanced storage resiliency.

BACKGROUND

[0003] Commercial enterprises (e.g., companies) and others gather, store, and analyze an increasing amount of data. The trend now is to store and archive almost all data before making a decision on whether or not to analyze the stored data. Although the per unit cost associated with storing data has declined over time, the total costs for storage has increased for many companies because of the volumes of stored data. Hence, it is important for companies to find cost-effective ways to manage their data storage environments for storing and managing large quantities of data. There are several problems with traditional approaches to capacity storage. Most traditional storage systems have difficulty scaling to support billions of values, which is far small than the trillions of objects that customers are storing today.

[0004] Traditional data protection mechanisms, e.g., RAID, are increasingly ineffective in petabyte-scale systems as a result of: larger drive capacities (without commensurate increases in throughput), larger deployment sizes (mean time between faults is reduced) and lower quality drives. The trends from the hard drive vendors are making traditional RAID increasingly difficult to implement, and are requiring complex techniques, e.g., triple parity, declustering. Some of the storage device trends that push away from traditional data protection mechanisms include: increasing drive sizes, lower I/O limits on drives, varying latency (which can slow I/O), varying capacity (within a given model/drive line, which can increase inefficiency of traditional RAID, lower drive reliability (increased failure rates, and more intense workload-triggered failures). Thus, the traditional data protection mechanisms are ill-suited for the emerging capacity storage market needs.

[0005] Further, the current data storage systems have complex data protection mechanisms, which typically involve performing a significant amount of I/O on the storage devices in order to provide a specified storage resiliency. This intensive I/O for protection purposes together with the I/O performed for providing data access to the customers wears the storage device much faster and therefore, decreases the lifespan of the device rapidly. In order to maintain the same storage resiliency, the storage devices may have to be replaced with new ones regularly, which can drive up the storage costs.

[0006] In an object based storage system, certain metadata, e.g., object size, creation date, owner, etc., are maintained for each object. In most of the current object storage systems, this metadata is kept in a database separate from the object data. Typically, this database is maintained in one or more different servers, e.g., meta-data servers. Ensuring that the objects themselves are consistent with the metadata in the metadata server is a difficult problem. The metadata servers themselves can become a bottleneck in the storage system, since they have to deal with updates every time an object is created, modified, or accessed. Typically, there is more than one meta-data server in order to address this bottleneck, but also to make sure that the meta-data is durable (not lost). The more such meta-data servers there are, the bigger the problem to keep them consistent with one another as well as the objects themselves.

BRIEF DESCRIPTION OF THE DRAWINGS

[0007] FIG. **1**A is a perspective plan view of a storage shelf and components therein, consistent with various embodiments.

[0008] FIG. **1**B is a perspective view of a storage rack of storage shelves, consistent with various embodiments.

[0009] FIG. **2** is a block diagram of a storage shelf, in accordance with various embodiments.

[0010] FIG. **3** is a block diagram illustrating an environment in which a data storage architecture can be implemented, consistent with various embodiments.

[0011] FIG. **4** is a block diagram of a storage system implementing wide spreading storage architecture, consistent with various embodiments.

[0012] FIG. **5** is a block diagram for storing metadata of a data object with the data object in a storage system of FIG. **4**, consistent with various embodiments.

[0013] FIG. **6** is a flow diagram of a process of storing data to an object-based storage system using the wide spreading storage architecture, consistent with various embodiments of the disclosed technology.

[0014] FIG. **7** is a flow diagram of a process of reading data from an object-based storage system using the wide spreading storage architecture, consistent with various embodiments of the disclosed technology.

[0015] FIG. **8** is a flow diagram of a process of rebuilding data fragments of a data object in the wide spreading storage architecture, consistent with various embodiments of the disclosed technology.

[0016] FIG. **9** is a flow diagram of a process of storing metadata of a data object with the data object in the wide spreading storage architecture, consistent with various embodiments of the disclosed technology.

[0017] FIG. **10** is a flow diagram of a process of processing metadata and data fragments of a data object in the wide spreading storage architecture, consistent with various embodiments of the disclosed technology.

[0018] FIG. **11** is a block diagram of a storage system implementing hierarchical spreading storage architecture, consistent with various embodiments.

[0019] FIG. **12** is a block diagram for storing metadata of a data object with the data object in a storage system of FIG. **11**, consistent with various embodiments.

[0020] FIG. **13** is a flow diagram of a process of storing data to an object-based storage system using the hierarchical spreading storage architecture, consistent with various embodiments of the disclosed technology.

[0021] FIG. **14** is a flow diagram of a process of reading data from an object-based storage system using the hierarchical spreading storage architecture, consistent with various embodiments of the disclosed technology.

[0022] FIG. **15** is a flow diagram of a process of rebuilding data fragments of a data object in the hierarchical spreading storage architecture, consistent with various embodiments of the disclosed technology.

[0023] FIG. **16** is a flow diagram of a process of rebuilding data segments of a data object in the hierarchical spreading storage architecture, consistent with various embodiments of the disclosed technology.

[0024] FIG. **17** is a flow diagram of a process of deferred rebuilding of data segments of a data object in the hierarchical spreading storage architecture, consistent with various embodiments of the disclosed technology.

[0025] FIG. **18** is a flow diagram of a process of processing metadata and data fragments of a data object in the hierarchical spreading storage architecture, consistent with various embodiments of the disclosed technology.

[0026] FIG. **19** is a block diagram of a computer system as may be used to implement features of some embodiments of the disclosed technology.

DETAILED DESCRIPTION

[0027] Technology is related to a data storage architecture for providing enhanced storage resiliency. Storage resiliency or data durability can be defined as a resistance to loss of one or more storage devices storing a portion of a data object or as a resistance to loss of one or more portions of the data object. The data storage architecture can be implemented in a single-tier configuration (also referred to as "wide spreading storage architecture") and/or a multi-tier configuration (also referred to as "hierarchical spreading storage architecture"). In either of the architecture, additional redundant portions of the data object are generated and stored across a number of storage devices, e.g., to provide storage resiliency for the data object. In some embodiments, the number of redundant portions generated depends on a specified storage resiliency. In some embodiments, the redundant portions are generated by encoding the data object based on an erasure coding method. The encoding of the data object generates a number of data object fragments, which include redundant fragments. The encoded data fragments are stored across various storage devices.

[0028] In the single-tier configuration of the data storage architecture, a storage system includes a number of storage devices, for example, hundreds or thousands of storage devices. A data object can be split into a number of fragments and stored across the storage devices. In some embodiments, the data object is encoded based on an erasure coding method to generate a number of fragments. The fragments are distributed across the storage devices. In some embodiments, the storage resiliency of the data object depends on a storage layout of the fragments. For example, if most of the fragments are stored on the same storage device or storage devices in a same storage shelf, the storage resiliency can be lower, as loss of the storage device or the storage shelf can result in higher probability of data loss. In another example, spreading the fragments widely across a large number of storage devices or storage shelves can have a better storage resiliency.

[0029] The number of encoded data fragments generated depends on a specified storage resiliency. In some embodiments, a ratio of the total number of fragments "n" generated to a minimum number of fragments "k" required for reconstructing the object is a function of the specified storage resiliency. For example, if n/k is 130%, then the storage resiliency is 30%. That is, the storage system can tolerate or resist loss of 30% of the data fragments without losing the data object. If the number of storage devices is more than n, the storage system can tolerate or resist loss of up to n of storage devices without losing the data. To obtain a storage resiliency of 30%, the storage system generates 30% redundant fragments for the purposes of data protection. For example, if the minimum number of fragments, k, is "1000," then the total number of fragments generated, n, is "1300", and the same system above would be able to tolerate "300" storage devices failing before data can be lost. This illustrates the importance to data protection of having a large n. The n data fragments are then spread widely across the storage devices. The storage resiliency can also be represented in the form of equation, n=k+m, where "k" is the original amount of data fragments or the minimum number of data fragments required to regenerate or rebuild the data object, and variable "m" stands for the extra or redundant fragments that are added to provide protection from failures. The variable "n" is the total number of fragments created after the encoding process. The data object can be reconstructed, e.g., in response to a request from a client system, by obtaining at least k encoded data fragments and decoding those to regenerate the data object.

[0030] In some embodiments, such storage resiliency can also be provided to metadata of the data object. The metadata of the data object can be stored with the data object and spread across various storage devices. This eliminates the need to store the metadata of the data objects in a separate repository from that of the data objects.

[0031] The single-tier storage architecture provides a number of benefits over existing architectures, e.g., RAID storage architecture. For example, in the single-tier architecture a write and/or read is spread across a large number of storage devices as opposed to a small set of storage devices in RAID. The writes and reads of the data fragments can be performed in parallel across the storage devices. Additionally, the number of reads performed on the storage devices can be further minimized as only a subset of the total number of data fragments is required to be read for regenerating the data object, thereby increasing a lifespan of the storage devices and lowering latency of access.

[0032] Further, the number of read-write operations performed on a particular storage device to regenerate the data fragments due to loss of one or more storage devices is minimized as the reads and writes are spread across the storage devices. For example, if a set of data fragments are lost due to failure of a storage device, the set of data fragments can be reconstructed by obtaining at least k data fragments from the remaining of the storage devices and generating the replacement data fragments as a function of the obtained data fragments. In some embodiments, the k data fragments are obtained from a first set of storage devices and the replacement data fragments are stored on a different set of storage devices, which distributes the read/write operations across different set of storage devices, thereby minimizing the read-write operations on a particular storage device and increasing the lifespan of the particular storage device.

[0033] Additionally, in the single-storage architecture, the mean-time-to-repair, which is how quickly the failed drive has to be repaired and the data stored in the failed drive to be reconstructed in order to provide a certain storage resiliency, is lower than that of current storage systems, e.g., RAID. Continuing with the above example of 30% storage resiliency with m equal to "300", the storage system can withstand loss

of up to "300" drives. So the repair process can defer operation until a high percentage of those drives have failed. Similarly, the mean time between failure, which is a statistical measure of the time until a failure occurs, in the single-tier storage architecture is higher than that of current storage systems, e.g., RAID. For example, as described above since the storage system distributes the read/write operations across different sets of storage devices, the read-write operations on a particular storage device is minimized, which increases the lifespan of the particular storage device.

[0034] In the multi-tier configuration of the data storage architecture, the storage system includes a number of storage computer nodes which are each associated with a set of storage devices. The storage system encodes a data object into a number of data segments and distributes them to a number of storage computer nodes. Each of the storage computer nodes further encodes the data segment into a number of fragments and stores the fragments across storage devices associated with the storage computer node. For example, the storage system can encode the data object into "16" segments and send each of the "16" segments to different storage computer nodes. Each of the storage computer nodes can encode, independent of the other storage computer nodes, the segment into "16" fragments and store them across a set of storage devices associated with the storage computer node. The storage system can distribute the segments to a selected set of storage computer nodes and store the fragments at a selected set of storage devices based on a storage layout of the data object. The storage layout can be specified by a user, e.g., an administrator of the storage system, or calculated automatically based on operational characteristics of the storage system, e.g., capacity, load, wear, age and health.

[0035] The storage resiliency in multi-tier configuration of the data storage architecture is distributed between the tiers. For example, if storage resiliency in two level storage architecture is 30%, then the first tier of storage computer nodes could offer 15% storage resiliency, with the second tier of storage devices offering 15% storage resiliency. In some embodiments, this can mean that the storage system can generate 15% extra segments and 15% extra fragments for protection purposes.

[0036] In some embodiments, such storage resiliency can also be provided to metadata of the data object. The metadata of the data object can be stored with the data object and spread across various storage devices, which eliminates the need to store the metadata of the data objects in a separate repository from that of the data objects. For example, the metadata can be prefixed to the segments and/or fragments and stored across various storage devices.

[0037] One of the advantages of multi-tier storage architecture is localized data regeneration process. For example, if a storage device of a particular storage computer node fails, a fragment of a particular segment stored on the failed storage device can be regenerated using other fragments of the segment stored at other storage devices of the storage computer node. The storage system may not have to obtain fragments from other storage computer nodes. After the replacement fragment is generated, it can be stored at one of the remaining storage devices of the storage computer node. The reads and writes are restricted to the storage devices of a particular storage computer node. By restricting the reads and writes to the local storage devices of a storage computer node, the data traffic in the network, e.g., between storage computer nodes, is minimized, as is the amount of data that must be read from storage devices.

[0038] The storage system can store the data object across two or more tiers. For example, the storage system can have two tiers of storage computer nodes, where a first tier storage computer node can be associated with a number of second tier storage computer nodes and each of the second tier storage computer nodes can be associated with a set of storage devices. The data object is split into number of segments and the segments are sent to first tier storage computer nodes, where each first tier storage computer node splits the corresponding data segment into a number of fragments and distributes the fragments to a number of second tier storage computer nodes. Each of the second tier computer storage nodes splits the data fragment to a number of sub-fragments and stores the sub-fragments across a set of storage devices associated with the second tier storage computer node.

[0039] The storage devices of the storage system can be organized as storage shelves and storage racks, where each storage rack includes a number of storage shelves and each storage shelf includes a number of storage devices. The storage racks/shelves/devices can be distributed across various geographical locations.

Environment

[0040] FIG. **1**A is a perspective plan view of a storage shelf **100** and components therein, consistent with various embodiments. The storage shelf **100** includes an enclosure shell **102** (partially shown) that encloses and protects multiple data storage devices **104**. The data storage devices **104** may be hard drives, solid-state drives, flash drives, tape drives, or any combination thereof. It is noted that the term "enclose" does not necessarily require sealing the enclosure and does not necessarily require enveloping all sides of the enclosure.

[0041] The storage shelf **100** further includes control circuitry **106** that manages the power supply of the storage shelf **100**, the data access to and from the data storage devices **104**, and other storage operations to the data storage devices **104**. The control circuitry **106** may implement each of its functions as a single component or a combination of separate components.

[0042] As shown, the storage shelf **100** is adapted as a rectangular prism that sits on an elongated surface **108** of the rectangular prism. Each of the data storage devices **104** may be stacked within the storage shelf **100**. For example, the data storage devices **104** can stack on top of one another into columns. The control circuitry **106** can stack on top of one or more of the data storage devices **104** and one or more of the data storage devices **104** can also stack on top of the control circuitry **106**.

[0043] In various embodiments, the enclosure shell **102** encloses the data storage devices **104** without providing window openings to access individual data storage devices or individual columns of data storage devices. In these embodiments, each of the storage shelves **100** is disposable such that after a specified number of the data storage devices **104** fail, the entire cartridge can be replaced as a whole instead of replacing individual failed data storage devices. Alternatively, the storage shelf **100** may be replaced after a specified time, e.g., corresponding to an expected lifetime.

[0044] The illustrated stacking of the data storage devices **104** in the storage shelf **100** enables a higher density of standard disk drives (e.g., 3.5 inch disk drives) in a standard

shelf (e.g., a 19 inch width rack shelf). Each storage shelf **100** can store ten of the standard disk drives. In the cases that the data storage devices **104** are disk drives, the storage shelf **100**A can hold the disk drives "flat" such that the spinning disks are parallel to the gravitational field.

[0045] The storage shelf **100** may include a handle **110** on one end of the enclosure shell **102** and a data connection port **112** (not shown) on the other end. The handle **110** is attached on an outer surface of the enclosure shell **102** to facilitate carrying of the storage shelf **100**. The enclosure shell **102** exposes the handle **110** on its front surface. For example, the handle **110** may be a retractable handle that retracts to fit next to the front surface when not in use.

[0046] FIG. **1**B is a perspective view of a storage rack **150** of storage shelves, consistent with various embodiments. The storage shelves may be instances of the storage shelf **100** illustrated in FIG. **1**A. The storage rack **150**, as illustrated, includes a tray structure **152** (e.g., a rack shelf) securing four instances of the storage shelf **100**. The tray structure **152** can be a standard 2U 19" deep rack mount. The storage rack **150** may include a stack of tray structures **152**, each securely attached to a set of rails **162**. Management devices **164** may be placed at the top shelves of the rack **150**. For example, the management devices **164** may include network switches, power regulators, front-end storage appliances, or any combination thereof.

[0047] FIG. **2** is a block diagram of a storage shelf **200**, in accordance with various embodiments. In some embodiments, the storage shelf **200** is the storage shelf **100** of FIG. **1**A. The storage shelf **200** includes a processor **202**, an operational memory **206**, a boot flash **208**, a data communication port **210**, a power management module **212**, storage interfaces **214**, and data storage devices **216**.

[0048] The processor **202** can be a microprocessor, a controller, an application specific integrated circuit, a field programmable gate array, or any combination thereof. The boot flash **208** is a memory device storing an operating system **218**. The processor **202** can load the operating system **218** into the operational memory **206** and run the operating system **218**. A data access application programming interface (API) service **220** can execute on this operating system to provide data access over a network to the data storage devices **216** for clients (e.g., devices, applications, or systems).

[0049] The data communication port **210** enables the storage shelf **200** to connect with the network. For example, the data communication port **210** can be a Power-over-Ethernet module that connects to an Ethernet cable to both establish a network connection with the network and power the storage shelf **200**.

[0050] In various embodiments, the storage shelf **200** only turns on a subset (hereinafter the "active set") of data storage devices **216** at a time. The active set can be a single data storage device or more than one data storage devices. The data access API service **220** can determine the membership of the active set depending on client requests received through the network. A client can either specifically request access to a data storage device or request a data range for the data access API service **220** to determine which data storage device stores the data range.

[0051] The power management module **212** provides electronic circuitry to switch on and off components of the storage shelf **200**, e.g., to activate only one subset of the data storage devices at a time. The power management module **212** can receive instructions from the data processing module **202** (e.g., as part of the data access API service **220**) to provide power to the designated active set, including a subset of the storage interfaces **214** that enables data access to the active set. Once power is supplied to the designated active set, the storage controller **222** can facilitate communicate between the data processing module **202** through the storage interface **214** to the data storage devices.

[0052] FIG. **3** is a block diagram illustrating an environment in which the data storage architecture can be implemented, consistent with various embodiments. The environment **300** includes a number of storage devices, e.g., storage device **304**, which are organized as a number of storage shelves **306***a-n* (collectively referred to as "storage subsystem **306**"). In some embodiments, each of the storage shelves in the storage subsystem **306** can be similar to the storage shelf **100** of FIG. **1**A and each of the storage devices, including the storage device **304**, can be similar to the data storage devices **104** or the data storage devices **216** of FIG. **2**. Further, the storage shelves **306***a-n* can be part of one or more storage racks, e.g., storage rack **150**. The storage subsystem **306** can be spread across various geographical locations.

[0053] The environment **300** includes one or more front-end subsystem **310** that facilitates storing and/or retrieving data from the storage subsystem **306**. The front-end subsystem **310** processes the read/write requests from clients **312***a-c* (collectively referred to as "clients **312**"). In some embodiments, the storage subsystem **306** is implemented as an object storage system, which manages data as data objects. The front-end subsystem **310** stores the data received from the clients as data objects in the storage subsystem **306**. The front-end subsystem **310** can receive the data from the clients as data objects or in other formats. If the front-end subsystem **310** receives the data in other formats, it can convert the data into data objects before storing the data in the storage subsystem **306**. In some embodiments, the front-end subsystem **310** also stores the metadata of the data with the data objects.

[0054] The environment **300** supports both single-tier configuration and multi-tier configuration of the data storage architecture. In the single-tier storage architecture, the front-end subsystem **310** encodes the data object, e.g., received from a client, to generate a number of data fragments and stores the data fragments across one or more of the storage devices of the storage subsystem **306**. In some embodiments, the front-end subsystem encodes the data object based on an erasure coding method. In some embodiments, an erasure coding method encodes the data object to generate n fragments. The n fragments include some redundant fragments which are generated for storage resiliency/data protection purpose. The erasure coding requires at least k out of n fragments to generate the data object. In some embodiments, the ratio of n to k indicates a storage resiliency of the data object.

[0055] In the multi-tier storage configuration, the environment **300** includes one or more tiers of hierarchical storage nodes, e.g., hierarchical storage nodes **314**-**318**. Each of the hierarchical storage nodes **314**-**318** can be associated with a set of storage devices. For example, the hierarchical storage node **314** is associated with storage devices from storage shelves **306***a* and **306***b*, the hierarchical storage node **316** is associated with storage devices from storage shelf **306***c*, and the hierarchical storage node **318** is associated with storage devices from storage shelves **306***d* and **306***e*.

[0056] In the multi-tier storage configuration, the front-end subsystem **310** encodes the data object, e.g., based on erasure coding, to generate a number of data segments and distributes

them to a number of hierarchical storage nodes, e.g., hierarchical storage nodes **314-318**. Each of the hierarchical storage nodes **314-318** further splits the data segment into a number of fragments and stores the fragments across storage devices associated with the hierarchical storage node. For example, the front-end subsystem **310** can split the data object into "3" segments and send each of the "3" segments to different hierarchical storage nodes **314-318**. Each of the hierarchical storage nodes **314-318**, e.g., hierarchical storage nodes **314** can split, independent of the other hierarchical storage nodes, the segment into "16" fragments and store them across a set of associated storage devices, e.g., storage devices from storage shelves **306***a* and **306***b*. The segments and fragments are distributed to a selected set of hierarchical storage nodes and storage devices, respectively, based on a storage layout of the data object. The storage layout can be specified by a user, e.g., an administrator of the storage system, or calculated automatically based on operational characteristics of the storage system, such as capacity, load, wear, age and health.

[0057] When a client system, e.g., client **312***a*, requests to access the data object, a front-end subsystem **310** determines the storage layout of the data segments, requests the identified hierarchical storage nodes, e.g., one or more of the hierarchical storage nodes **314-318**, to obtain the fragments of a segment from the storage devices and decode them to generate the segment, and decodes the segments to generate the data object. The front-end subsystem **310** returns the data object to the client **312***a*. In some embodiments, the front-end subsystem **310** obtains at least the minimum number of segments required to regenerate the data object and the hierarchical storage nodes obtain at least the minimum number of fragments required to regenerate the data segment.

[0058] In some embodiments, both the single-tier configuration and multi-tier configuration of the data storage architecture can be implemented in the same storage system as illustrated in the environment **300**. Further, in some embodiments, one of the two configurations is automatically and/or dynamically chosen for performing the read/write operations. A particular configuration can be selected based on a number of factors, e.g., type of data to be written, a client from whom the data is received, included metadata, etc. In some embodiments, the front-end subsystem **310** is configured to select the particular configuration based on the above factors.

[0059] FIG. **4** is a block diagram of storage system **400** implementing wide spreading storage architecture, consistent with various embodiments. In some embodiments, the storage system **400** can be implemented in the environment **300** of FIG. **3**. The storage system **400** includes the front-end subsystem **310** that facilitates data storage and retrieval from the storage subsystem **306**. The front-end subsystem **310** can be one or more computer systems (e.g., the computing device **1800** of FIG. **18**), having either a shared nothing architecture or a shared database architecture, connected to the storage subsystems **306** over a network (e.g., a global network or a local network). The front-end subsystem **310** can be on a separate rack from the storage subsystem **306**, or can be combined with the hierarchical storage node **314** or storage shelf **306**.

[0060] The front-end subsystem **310** includes a protocol interfaces module **406**. The protocol interfaces module **406** defines one or more functional interfaces that applications and devices use to store, retrieve, update, and delete data elements from the storage system **400**. For example, the protocol interfaces module **406** can implement a Cloud Data Management Interface (CDMI), a Simple Storage Service (S3) interface, or both. The front-end subsystem **310** includes a staging area **408**. The staging area **408** is a memory space implemented by one or more data storage devices within or accessible to the front-end subsystem **310**. For example, the staging area **408** can be implemented by solid-state drives, hard disks, volatile memory, or any combination thereof. The staging area **408** can maintain an object namespace **410** to facilitate client interactions through the protocol interfaces module **406**. The object namespace **410** manages a set of data container identifiers, e.g., object identifiers of data received from clients of the front-end subsystem **310**. The staging area **408** also maintains a fragment namespace **412** corresponding to the object namespace **410**. The fragment namespace **412** manages a set of fragment identifiers, each corresponding to a data fragment stored in the storage subsystem **306**. The staging area **408** can store a mapping structure **414** that stores associations between the data container identifiers of the object namespace **410** and the fragment identifiers of the fragment namespace **412**.

[0061] In some embodiments, the front-end subsystem **310** can be implemented as a distributed computing network including multiple computing nodes (e.g., computer servers). Each computing node can include an instance of the staging area **408**. The namespaces (e.g., the object namespace **410** and the fragment namespace **412**) of each staging area **408** can be implemented either as a share-nothing database or a shared database.

[0062] The staging area **408** can also serve as a temporary cache to process payload data from a write request received at the protocol interfaces module **406**. The request module **416** receives read/write requests from the clients of the storage system **400**. The front-end subsystem **310** processes an incoming write request by performing a number of storage efficiency processes on the payload data of the write request prior to sending the payload data into persistent storage in the storage subsystem **306**. In some embodiments, the storage efficiency processes include deduplication, compression, fragmentation, erasure coding and fragment encryption of the payload data.

[0063] The storage processing module **430** performs the deduplication process on the payload data, which removes duplicate data portions from the payload data. The storage processing module **430** can use a number of deduplication techniques for deduplicating the payload data. The storage processing module **430** can compress the payload data, e.g., to reduce the storage space occupied by the payload data. The storage processing module can implement one or more compression algorithms for compressing the payload data.

[0064] The encode/decode module **418** fragments the payload data into a number of fragments, which includes redundant fragments for the purpose of data protection. In some embodiments, the encode/decode module **418** performs the encoding based on one or more erasure coding techniques. In some embodiments, erasure coding is a method of data protection in which payload data is broken into fragments, expanded and encoded with redundant data fragments. For example, payload data can be broken into k fragments and erasure coded data to generate n fragments, where n>k, such that the payload data can be recovered from a subset of the n, e.g., at least k fragments.

[0065] The storage processing module **430** can further encrypt the data fragments using one or more encryption

techniques to generate encrypted data fragments. In some embodiments, the storage processing module **430** encrypts the fragments for data security purposes.

[0066] Note that the order of execution of storage efficiency processes is not restricted to the order described above. Alternative embodiments may perform these storage efficiency processes in a different order, and some processes may be removed, moved, added, subdivided, combined, and/or modified to provide alternatives or sub combinations.

[0067] The storage layout module **420** determines the storage layout of the data fragments. The storage layout identifies one or more of the storage racks, storage shelves of a rack and storage devices of a storage shelf the data fragments have to be stored in. In some embodiments, the storage layout module **420** determines the optimal layout of fragments to meet the service level object (SLO) promised to the client and/or to maximize storage resiliency, and sends the fragments to the selected storage devices of the storage subsystem **306** for storage. In some embodiments, a best storage layout stores each of the data fragments in a different storage device of the storage subsystem **306** to provide the best storage resiliency. In some embodiments, a worst storage layout stores all of the data fragments in the same storage device of the storage subsystem **306**. Typically, the storage layout module **420** is configured to distribute the fragments across the storage devices as widely as possible, that is, to store distinct fragments on distinct storage devices.

[0068] In some embodiments, the storage layout module **420** selects the storage devices on a random basis. In some embodiments, the storage layout module **420** selects the storage devices on a random weighted basis. The storage layout module **420** can weigh the storage devices based on a number of factors, e.g., available storage capacity, a write latency of the storage device, a read latency of the storage device, a type of the storage device. For example, the storage layout module **420** can randomly select the storage devices from a set of storage devices that have at least some specified percentage of storage capacity free. In some embodiments, the random weighted basis attempts to store the data fragments evenly across the available storage devices. For example, one type of weighting is to decrease the weight if there are already a specified number of fragments stored on the storage device. In some embodiments, the random weighted basis randomly identifies the storage devices at which the encoded data fragments are to be stored as a function of decreasing the risk of data loss. For example, if a particular geographical region is prone to higher number of device failures, then the storage devices in that geographical region may be weighted less so that a lower number of fragments are written to the storage devices in that geographical region.

[0069] In some embodiments, the storage layout module **420** can select the storage devices based on parameters defined by a user, e.g., metadata, a client of the storage system **400**, and/or an administrator of the storage system **400**.

[0070] The following paragraphs describe additional details of writing data to the storage subsystem **306** in wide spreading storage architecture.

[0071] When a client, e.g., client **312***a*, sends a write request to the storage system **400**, the request module **416** receives the request and extracts the data object to be written from the request. The storage processing module **430** performs a number of processes on the data object, e.g., as described above. The encode/decode module **418** encodes the data to generate n fragments. The encode/decode module **418** can use an erasure coding method, e.g., Reed-Solomon, FEC code, Fountain code, Raptor code, Tornado code.

[0072] In FIG. **4**, the encode/decode module **418** splits the data object **405** into n fragments, F_1 to F_N. The storage layout module **420** determines the storage layout of the fragments and spreads the fragments, F_1 to F_N across the storage devices of the storage subsystem **306**. For example, the storage layout module **420** determines that the fragments, F_1 to F_{99} have to be sent to the storage devices of "storage shelf 1," fragments, F_{100} to F_{199} to the storage devices of "storage shelf 2," and fragments, F_{200} to F_N to the storage devices of "storage shelf N." In some embodiments, the storage layout module **420** also determines the storage devices of the storage shelves where the fragments have to be stored. After the storage layout module **420** determines the storage layout, the transceiver module **432** transmits the data fragments to the corresponding storage shelves, which store the data fragments at the storage devices. In some embodiments, the fragments can be written to the different storage devices in parallel.

[0073] The number of fragments generated by the encode/decode module **418** depends on the required storage resiliency. The storage resiliency offered can be represented as n=k+m, where variable "k" is the original amount of data fragments or the minimum number of data fragments required to regenerate or rebuild the data object, and variable "m" stands for the extra or redundant fragments that are added to provide protection from failures. The variable "n" is the total number of fragments created after the encoding process.

[0074] Typically, in the wide spreading data storage architecture, the width to which the data object is split is wider, and the degree to which the data fragments are spread across the storage devices is wider, e.g., compared to current storage architecture such as RAID. For example, the number of fragments to which the data object is split into can be in hundreds and the number of storage devices across which the hundreds of fragments are spread across can be in the thousands to tens of thousands.

[0075] In some embodiments, a ratio of "n" to "k" indicates the storage resiliency provided for the data object. For example, if n/k is 130%, then the storage resiliency is 30%. That is, the storage system can tolerate or resist loss of 30% of the data fragments without losing the data object. If the number of storage devices is more than n, the storage system can tolerate or resist loss of up to n of storage devices without losing the data. For example, if the minimum number of fragments, k, is "1000," then the total number of fragments generated, n, is "1300.", and the same system above would be able to tolerate "300" storage devices failing before data can be lost. This illustrates the importance to data protection of having a large n. To obtain a storage resiliency of 30%, the storage system generates 30% redundant fragments for the purposes of data protection. For example, if the minimum number of fragments, k, is "1000," then "m" is "300" and n is "1300." The n data fragments are then spread widely across "4000" storage devices.

[0076] The object identifier of the data object and the fragment identifiers of the fragments are stored in the staging area **408** at the object namespace **410** and the fragment namespace **412**, respectively. Further, a mapping of the object identifier to the fragment identifiers can be stored in the mapping structure **414** of the staging area **408**.

[0077] When a read request arrives at the storage system **400** from the client **312***a* for the data object, the data object can be reconstructed by obtaining at least k number of the F_N

data fragments and decoding them to regenerate the data object. The transceiver module **432** obtains the storage layout of the fragments from the storage layout module **420** and obtains the data fragments from the identified storage devices of the storage subsystem **306**. The storage layout module **420** can use the mapping structure **414** to obtain the fragment identifiers of the data object and then determine the storage devices at which the corresponding fragments are stored.

[0078] The transceiver module **432** can obtain from k to n number of fragments. For example, the transceiver module **432** can stop fetching the fragments after obtaining the first k fragments. In another example, the transceiver module **432** can fetch all the n fragments but use only the first k fragments for regenerating the data object.

[0079] Further, the transceiver module **432** can preferentially select a subset of the storage devices identified by the storage layout module **420** to obtain the fragments from. The transceiver module **432** selects a storage device based on a number of factors, e.g., read latency of storage device, type of the storage device, number of pending read requests ahead of the current read request in a read request queue of the storage device, how far away the storage device is. Accordingly, the transceiver module **432** may not even read some of the storage devices that contain the data fragments of the data object, thereby minimizing read/write operation on the storage device. In some embodiments, the transceiver module **432** can obtain the fragments from different storage devices in parallel.

[0080] After obtaining the data fragments, the encode/decode module **418** decodes the data fragments, e.g., based on the erasure coding used to encode the data object, to generate the data object. In some embodiments, the storage processing module **430** may perform additional processes on the decoded object before returning the data object to the client **312***a*. For example, the storage processing module **430** can perform decompression and de-deduplication on the decoded data object if the data object was deduplicated and compressed.

[0081] The wide spreading storage architecture provides a robust storage resiliency to the data stored in the storage subsystem **306**. The wide spreading storage architecture also provides an efficient way to rebuild the data fragments in case of storage device failures. When a storage device fails, the data fragments stored at the storage device may be lost. When a failure detection module **424** detects a failure or impending failure of a storage device, the failure detection module **424** requests the regeneration module **428** to evacuate readable fragments or rebuild unreadable or lost data fragments to compensate for the ones that are no longer reliably stored. The regeneration module **428** facilitates rebuilding of new data fragments of a data object using the remaining data fragments of the data object stored at other storage devices. For example, if a storage device in "storage shelf 2" storing the data fragments F_4-F_{10} fails, the regeneration module **428** can rebuild up to new six data fragments and writes the new data fragments to any of the remaining set of storage devices. In some embodiments, the regeneration module **428** rebuilds the data fragments using sufficient number of the remaining data fragments F_1-F_3 and F_{11}-F_N. The regeneration module **428** can use the encoding method used to generate the initial fragments to generate the new replacement fragments.

[0082] The failed storage device can store data fragments of one or more data objects. The fragment/segment identification module **422** can determine the fragments stored on the storage device that failed, e.g., using the storage layout. The regeneration module **428** can rebuild the data fragments of all the data objects whose fragments are lost or for only a set of data objects that have lost the data fragments. For example, the regeneration module **428** can rebuild the data fragments of a data object whose current storage resiliency is lesser than a specified threshold for minimum storage resiliency. The current storage resiliency is determined as a function of the remaining of "n" number of fragments and "k." For example, if the specified threshold for minimum storage resiliency of a data object is 10% and the current storage resiliency is less than 10%, then the data fragments can be rebuilt for the data object. Further, the regeneration module **428** can start rebuilding the data fragments of the data object whose current storage resiliency is lesser than the specified threshold instantaneously, e.g., in response to the failure of the storage device. The regeneration module **428** can rebuild the data fragments of other data objects whose current storage resiliency exceeds the specified threshold at a later time. In some embodiments, the regeneration module **428** executes the rebuilding process as a background process of the front-end subsystem **310**. In some embodiments, a user, e.g., administrator of the storage system **400** can manually execute the rebuilding process.

[0083] The wide spreading storage architecture can resist higher number of storage device failures than that of current storage systems, e.g., RAID storage system. For example, if the storage system **400** offers a storage resiliency of 30% and has a k of 1000, then the storage system **400** can resist a failure of "300" storage devices before the data is lost. So if one or more storage devices are lost, or even if an entire storage shelf/storage rack is lost, there may not be much impact on the storage resiliency. This provides a number of advantages. First, the rebuilding process may not have to be started immediately; it can be done at a later time. The storage resiliency of the lost data fragments can be repaired over time, e.g., when the work load (data read-write operations) on the storage system **400** is below a threshold, or when the current storage resiliency drops below the specified threshold, e.g., when the current storage resiliency is less than 10%—which means the storage system **400** can only tolerate failure of "200" more storage devices. That is, the wide spreading storage architecture offers a high mean time to repair, e.g., compared to RAID storage architecture.

[0084] Second, the wide spreading storage architecture separates the rebuilding of data fragments from replacement of the failed storage devices. That is, the storage system **400** may not have to wait until the failed storage devices are replaced to rebuild the data fragments. The rebuilding process reads the data fragments of the data object from the remaining storage devices, generates new data fragments as a function of the data fragments obtained from the other storage devices, and writes the new data fragments on one or more of the remaining storage devices. Accordingly, in the wide spreading storage architecture, the storage system **400** does not have to wait for the failed storage device to be replaced to rebuild the data fragments, unlike current storage architectures, e.g., RAID storage architecture without hot spares, where a failed storage device may have to be replaced immediately upon failure.

[0085] However, if the failed storage device is replaced immediately upon failure, the storage system **400** can use the replacement storage device as additional capacity, e.g., to

store new data. Further, the replacement storage device can be of different storage capacity and/or type from that of the failed storage device.

[0086] The wide spreading storage architecture also minimizes the number of read-write operations required per storage device for rebuilding the data fragments of a particular data object. The regeneration module **428** obtains the remaining data fragments of the particular data object from other storage devices of the storage subsystem **306**. Since the data fragments are spread over a number of storage devices, the number of read operations performed for the rebuilding process is spread across many storage devices and therefore, the number of read operations performed on a particular storage device is limited. Further, in some embodiments, the regeneration module **428** obtains less than the remaining number of fragments, e.g., k fragments of the remaining fragments, to rebuild the lost data fragments, which further minimizes the read operations performed on the storage devices. By minimizing the read operations on a given storage device, the wear of the storage device is minimized and the lifespan of the storage device is therefore, increased. Further, as rebuild can be deferred and performed after many failures have occurred, rebuild operations are minimized compared to architectures were rebuilds are initiated for each failure operation.

[0087] Furthermore, after rebuilding the new data fragments, the new data fragments are written to a set of storage devices. In some embodiments, the set of storage devices to which the data is written is different from the set of storage devices from which the data fragments are read to rebuild the data fragments. Accordingly, the read-write operations performed on any given storage device is minimized, which minimizes the wear of the storage device and therefore, increases the lifespan of the storage device.

[0088] As described above, the wide spreading storage architecture provides optimum storage resiliency to data stored in the storage devices of the storage subsystem **306** while minimizing the wear of the storage devices.

[0089] The wide spreading storage architecture can also be used to store metadata of the data object. FIG. **5** is a block diagram **500** for storing metadata of a data object with the data object in a storage system **400** of FIG. **4**, consistent with various embodiments. The wide spreading storage architecture can provide the same storage resiliency to the metadata of a data object that is provided to the data object. Examples of metadata can include, object ID, object size, object owner, creation time, created by, modified by, etc. The metadata can also include client-specified metadata, e.g., author of an object, name of entity, etc. Typically, current storage architectures store metadata separate from the data object. The wide spreading storage architecture enables storing the metadata with data object, thereby eliminating the need to have a separate database for the metadata, the need to have specific infrastructure to ensure the metadata is consistent with the data, etc.

[0090] When a write request is received, the payload data in the write request is analyzed to obtain the metadata **510** and the data portion, e.g., data object **405**. The data object **405** is then encoded, e.g., using encode/decode module **418** as described with reference to FIG. **4**, to generate a number of fragments **505**. The metadata **510** is combined with some or each of the fragments **505**, e.g., concatenated or prefixed to each of the fragments **505**, to generate composite fragments **515**. The composite fragments **515** can then be stored in the storage subsystem **306** by spreading them across a number of storage devices, e.g., similar to storing the data fragments as described with reference to FIG. **4**. In some embodiments, the metadata **510** can be a subset of the metadata of the data object **405**.

[0091] In some embodiments, by including the metadata **510** with the data object, the possibility of inconsistency between the metadata **510** and the data object **405** is eliminated. Further, since the metadata **510** is attached to the fragments **505**, the composite fragments **515** can be moved across locations/storage devices without having to update the metadata **510** and without risking the consistency between the metadata **510** and the data object **405**.

[0092] Another benefit of storing the metadata **510** with the data object **405** is that since a separate database and/or metadata server is not needed to maintain the metadata **510**, the read and write operations are relatively faster since no separate read/write is required to read/write the metadata **510**. In some embodiments, metadata retrieval is also simplified since a method call that is used for retrieving the data object **405** can be modified to use retrieve the metadata **510**, which can simplify a number of functions performed related to the metadata **510**.

[0093] FIG. **6** is a flow diagram of a process **600** of storing data to an object-based storage system using wide spreading storage architecture, consistent with various embodiments of the disclosed technology. In some embodiments, the process **600** may be implemented in environment **300** of FIG. **3**, and using the storage system **400** of FIG. **4**. The process **600** begins at block **605**, and at block **610**, a request module **416** of the frontend subsystem **310** receives a write request including payload data. In some embodiments, the payload data includes data portion and metadata of the data. If the data portion is not in a format suitable for storing in an object storage system, e.g., storage subsystem **306**, the frontend subsystem **310** converts the data portion to the suitable format, e.g., as the data object.

[0094] At block **615**, the encode/decode module **418** encodes the data object to generate a number of encoded data fragments, e.g., encoded data fragments F**1**-FN. In some embodiments, the encode/decode module **418** encodes the data object based on an erasure coding technique. The number of encoded data fragments generated can be expressed as a function, e.g., n=k+m, where variable "k" is the original amount of data fragments or the minimum number of data fragments required to regenerate or rebuild the data object, and variable "m" is the number of extra or redundant fragments added to provide protection from storage device failures. The variable "n" is the total number of fragments created after the encoding process.

[0095] After the encoded data fragments are generated, a mapping of the object identifier of the data object and fragment identifiers of the encoded data fragments are stored in the mapping structure **414**.

[0096] In some embodiments, apart from encoding the data object to generate the fragments, various other processes may be performed on the data object, e.g., deduplication, compression, encryption. One or more of these processes can be performed by the storage processing module.

[0097] At block **620**, the storage layout module **420** determines a storage layout for storing the encoded data fragments across a number of storage devices, e.g., storage devices of storage subsystem **306**. In some embodiments, the storage layout module **420** is configured to spread the encoded data fragments across as many storage devices as possible, e.g., to

provide better storage resiliency to the data object. That is, the storage layout module **420** attempts to identify different storage devices for storing different encoded data fragments. In some embodiments, the storage layout module **420** selects the storage devices on a random basis. In some embodiments, the storage layout module **420** selects the storage devices on a random weighted basis.

[0098] At block **625**, the transceiver module **432** transmits the encoded data fragments to the identified storage devices. For example, the transceiver module **432** can transmit the encoded data fragments to the storage shelves and/or the storage racks which contain the storage devices.

[0099] At block **630**, the storage shelves and/or the storage racks store the encoded data fragments at the identified storage devices, and the process **600** returns. In some embodiments, the front-end subsystem **310** also stores the metadata of the data object with the data object. Additional details with respect to the process of storing the metadata are described at least with reference to FIGS. **9** and **10**.

[0100] FIG. **7** is a flow diagram of a process **700** of reading data from an object-based storage system using wide spreading storage architecture, consistent with various embodiments of the disclosed technology. In some embodiments, the process **700** may be implemented in environment **300** of FIG. **3**, and using the storage system **400** of FIG. **4**. The process **700** begins at block **705**, and at block **710**, a request module **416** of the frontend subsystem **310** receives a read request, e.g., from a client system **312***a*, for obtaining a data object. In some embodiments, the read request includes an object identifier of the data object.

[0101] At block **715**, the fragment/segment identification module **422**, determines the encoded data fragments of the data object using the object identifier. In some embodiments, a mapping of the object identifier and the fragment identifiers of the encoded data fragments are stored in the mapping structure **414**.

[0102] At block **720**, the storage layout module **420** determines the storage layout of the encoded data fragments using the mapping obtained from the mapping structure. The storage layout can include identification information of the storage devices where each of the encoded data fragments is stored. In some embodiments, the storage layout information can also include identification information of the storage racks and/or storage shelves of the storage devices where the encoded data fragments are stored.

[0103] At block **725**, the transceiver module **432** obtains sufficient number of the encoded data fragments required to generate the data object from the identified storage devices. In some embodiments, the sufficient number of encoded data fragments is k number of the encoded data fragments. In some embodiments, the transceiver module **432** can obtain k to n number of fragments. For example, the transceiver module **432** can stop fetching the fragments after obtaining the first k fragments. In another example, the transceiver module **432** can fetch all the n fragments but use only the first k fragments for regenerating the data object.

[0104] Further, the transceiver module **432** can preferentially select a subset of the identified storage devices to obtain the fragments from. The transceiver module **432** can select a storage device based on a number of factors, e.g., read latency of a storage device, type of the storage device, number of pending read requests ahead of the current read request in a read request queue of the storage device, a geographical location of the storage device. In some embodiments, the transceiver module **432** can obtain the fragments from different storage devices in parallel.

[0105] After obtaining the encoded data fragments, at block **730**, the encode/decode module **418** decodes the encoded data fragments, e.g., based on the erasure coding method used to encode the data object, to generate the data object.

[0106] At block **735**, the transceiver module **432** transmits the data object in response to the read request, e.g., to the client system **312***a*, and the process **700** returns. In some embodiments, additional processes may be performed before decoding the data fragments. For example, the storage processing module **430** can decrypt the encoded data fragments if they were encrypted before being stored. In some embodiments, additional processes may be performed on the decoded data object before returning the data object to the client **312***a*. For example, the storage processing module **430** can perform decompression and de-deduplication on the decoded data object if the data object was deduplicated and compressed.

[0107] FIG. **8** is a flow diagram of a process **800** of rebuilding data fragments of a data object in wide spreading storage architecture, consistent with various embodiments of the disclosed technology. In some embodiments, the process **800** may be implemented in environment **300** of FIG. **3**, and using the storage system **400** of FIG. **4**. In some embodiments, the data fragments stored in the storage subsystem **306** may be lost due to a failure of a storage device. The process **800** begins at block **805**, and at block **810**, a failure detection module **424** of the frontend subsystem **310** detects a failure of a storage device, e.g., storage device **304**. In some embodiments, the failure can be one or more of the storage device being not accessible, the storage device being physically damaged, etc.

[0108] At block **815**, the fragment/segment identification module **422** identifies the encoded data fragments that were stored at the storage device. For example, the fragment/segment identification module **422** can refer to the storage layout module **420** to determine the fragments stored at the storage device that has failed. Further, the fragment/segment identification module **422** identifies the one or more data objects corresponding to the identified encoded data fragments. For example, the fragment/segment identification module **422** can refer to the mapping structure **414** to determine the data objects associated with the identified encoded data fragments.

[0109] At block **820**, the regeneration module **428** rebuilds some or all of the encoded data fragments that was stored at the storage device that failed. In some embodiments, rebuilding the data fragments include performing the method described in association with blocks **821-824** for each of the identified data objects. At block **821**, the regeneration module **428** computes the current storage resiliency of the data object. In some embodiments, storage resiliency is defined as a resistance to loss of one or more storage devices storing a portion of a data object or resistance to loss of one or more portions of the data object. In some embodiments, a current storage resiliency of a data object is determined as a function of the number of fragments remaining out of "n" fragments and "k." For example, if n is "130," k is "100," then the number of redundant fragments, m is "30," and therefore, the storage resiliency can be calculated as 30% (100*m/k). Note that the

storage resiliency can be calculated using other functions and based on several other parameters.

[0110] The storage system **400** may guarantee a storage resiliency range to the clients of the storage system, for example, a minimum storage resiliency and a maximum storage resiliency. In some embodiments, the storage resiliency range is part of the SLO guaranteed to the clients. In some embodiments, the storage system **400** may not rebuild the lost data fragments until the current storage resiliency of the data object drops below the minimum storage resiliency.

[0111] At determination block **822**, the regeneration module **428** determines if the current storage resiliency of the data object is less than the minimum storage resiliency. Continuing with the above example of a storage resiliency of 30%, if the minimum storage resiliency is 10%, then the storage system **400** can withstand loss of "20" data fragments, in which case m is "10."

[0112] Responsive to a determination that the current storage resiliency of the data object is not less than the minimum storage resiliency, the process **800** returns. On the other hand, responsive to a determination that the current storage resiliency is less than the minimum storage resiliency, at block **823**, the transceiver module **432** obtains sufficient number of fragments of the data object from remaining of the storage devices. The transceiver module **432** may use the storage layout to identify the storage devices that store the data fragments of the data object. In some embodiments, the transceiver module **432** can obtain the minimum number of fragments required to rebuild the data fragments.

[0113] At block **824**, the regeneration module **428** regenerates the data fragments as a function of the obtained data fragments and stores the regenerated data fragments in at least a subset of the remaining storage devices. In some embodiments, the regeneration module **428** regenerates as many data fragments as required to meet a specified storage resiliency, which can be up to the maximum storage resiliency. In some embodiments, regenerating the data fragments as a function of the obtained data fragments includes encoding the obtained data fragments to generate the new/replacement/additional data fragments. In some embodiments, regenerating the data fragments as a function of the obtained data fragments includes decoding the obtained data fragments to generate the data object and encoding the generated data object to generate the specified number of data fragments.

[0114] FIG. **9** is a flow diagram of a process **900** of storing metadata of a data object with the data object in wide spreading storage architecture, consistent with various embodiments of the disclosed technology. In some embodiments, the process **900** may be implemented in environment **300** of FIG. **3**, and using the storage system **400** of FIG. **4**. The process **900** begins at block **905**, and at block **910**, a request module **416** of the frontend subsystem **310** receives a write request including payload data. In some embodiments, the payload data includes data portion and metadata of the data. If the data portion is not in a format suitable for storing in an object storage system, e.g., storage subsystem **306**, the frontend subsystem **310** converts the data portion to the suitable format, e.g., as the data object.

[0115] At block **915**, the metadata processing module **426** analyzes the payload data to obtain the metadata of the data object, e.g., metadata **510** of FIG. **5**. Examples of metadata can include, object ID, object size, object owner, creation time, created by, modified by, etc. The metadata can also include client-specified metadata, e.g., author of an object, name of entity, etc.

[0116] At block **920**, the encode/decode module **418** encodes the data object to generate a number of encoded data pieces, e.g., segments and/or fragments. In some embodiments, the encode/decode module **418** encodes the data object as described at least with reference to FIGS. **4-6**.

[0117] At block **925**, after the encoded data pieces are generated, the metadata processing module **426** processes the encoded data pieces and the metadata for storage across a number of storage devices, e.g., storage devices of the storage subsystem **306**, and the process **900** returns. Additional details with respect to the method of processing the metadata are described at least with reference to FIG. **10**.

[0118] FIG. **10** is a flow diagram of a process **1000** of processing metadata and data fragments of a data object in wide spreading storage architecture, consistent with various embodiments of the disclosed technology. In some embodiments, the process **1000** may be implemented in environment **300** of FIG. **3**, and using the storage system **400** of FIG. **4**. In some embodiments, the process **1000** implements the method of block **925** of FIG. **9**. The data piece generated in the process **900** of FIG. **9**, e.g., in block **920**, can be considered as a data fragment in the wide spreading storage architecture. The process **1000** begins at block **1005**, and at block **1010**, the metadata processing module **426** combines each of the data fragments of the data object with the metadata, e.g., metadata **510**, to generate composite encoded data fragments, e.g., composite encoded data fragments **515**. In some embodiments, combining the metadata with each of the fragments includes concatenating or prefixing the metadata to each of the fragments.

[0119] After the composite fragments are generated, at block **1015**, the transceiver module **432** transmits the composite fragments to the storage subsystem **306** for storing across a number of storage devices, e.g., similar to storing the data fragments as described at least with reference to blocks **620-630** of FIG. **6**, and the process **1000** returns. Prior to transmitting the composite fragments to the storage subsystem **306**, the storage layout module **420** determines a storage layout for storing the composite data fragments across the number of storage devices, e.g., similar to determining the storage layout for storing the data fragments as described at least with reference to FIG. **4** and block **620** of FIG. **6**. The transceiver module **432** then transmits the composite data fragments to the identified storage devices.

[0120] FIG. **11** is a block diagram of storage system **1100** implementing hierarchical spreading storage architecture, consistent with various embodiments. In some embodiments, the storage system **1100** can be implemented in the environment **300** of FIG. **3**. Further, in some embodiments, the storage system **1100** includes at least some of the characteristics, behavior/functionalities of the storage system **400** of FIG. **4**. In some embodiments, the wide spreading storage architecture of storage system **400** can also be implemented in the storage system **1100**. The storage system **1100** includes the front-end subsystem **310** and a tier of hierarchical storage nodes, e.g., hierarchical storage nodes **314-318** that facilitate data storage and retrieval from the storage subsystem **306**, which includes storage shelves **306***a-n*. The hierarchical storage nodes can be implemented in a similar configuration to that of the front-end subsystem **310**. For example, a hierarchical storage node can include the modules/components of the front-end subsystem **310** depicted in FIG. **3**. Note that

although FIG. **11** depicts one tier of hierarchical storage nodes, the hierarchical spreading storage architecture can have more than one tier of hierarchical storage nodes.

[0121] Each of the hierarchical storage nodes **314-318** can be associated with a set of storage devices. For example, the hierarchical storage node **314** is associated with storage devices from storage shelves **306***a* and **306***b*, the hierarchical storage node **316** is associated with storage devices from storage shelf **306***c*, and the hierarchical storage node **318** is associated with storage devices from storage shelves **306***d* and **306***e*. In some embodiments, the hierarchical storage nodes are spread across various geographical locations. In other embodiments, the hierarchical storage nodes are integrated into each storage shelf.

[0122] The following paragraphs describe additional details of writing data to the storage subsystem **306** in hierarchical spreading storage architecture.

[0123] When a client, e.g., client **312***a*, sends a write request to the storage system **1100**, the request module **416** receives the request and extracts the data object to be written from the request. The encode/decode module **418** encodes the data object to generate a number of segments, e.g., "S**1**," "S**2**," and "S**3**". In some embodiments, the encode/decode module **418** can use wide spreading, or an erasure coding method directly, e.g., Reed-Solomon, FEC coding, Fountain code, Raptor code, Tornado code, to generate the segments. In some embodiments, the number of segments generated is a function of the number of hierarchical storage nodes.

[0124] The transceiver module **432** distributes the data segments to a number of hierarchical storage nodes, e.g., hierarchical storage nodes **314-318**. The storage layout module **420** determines the storage layout of the segments, that is, the hierarchical storage nodes to which the segments have to be distributed, and the transceiver module **432** spreads the segments to the identified the hierarchical storage nodes. In some embodiments, the storage layout module **420** is configured to select different hierarchical storage nodes for different segments, e.g., to maximize storage resiliency of the data object. However, in some embodiments, more than one segment may be transmitted to a hierarchical storage node. In some embodiments, the storage layout module **420** determines the hierarchical storage nodes to which the segments have to be distributed on a random basis. The storage layout can also be specified by a user, e.g., an administrator of the storage system **1100**. In FIG. **11**, the segment, "S**1**" is sent to the hierarchical storage node **314**, the segment "S**2**" is sent to the hierarchical storage node **316** and the segment "S**3**" is sent to the hierarchical storage node **318**. In some embodiments, the segments are transmitted to the hierarchical storage nodes in parallel.

[0125] The number of segments generated by the encode/decode module **418** can also depend on the required storage resiliency. The storage resiliency offered can be represented as n'=k+m', where variable k' is the original amount of data segments or the minimum number of data segments required to rebuild the data object, and variable m' stands for the extra or redundant segments added to provide protection from failures, e.g., failures of hierarchical storage nodes and/or storage devices associated with hierarchical storage nodes. The variable n' is the total number of segments created after the encoding process.

[0126] The segment identifiers of the data object may be stored in the fragment namespace **412**. The mapping structure **414** can store a mapping of the object identifier of the data object to the segment identifiers of the segments of the data object.

[0127] In some embodiments, prior to encoding the data object, the storage processing module **430** can perform a number of storage efficiency processes on the data object, e.g., as described at least with reference to FIG. **4**.

[0128] Each of the hierarchical storage nodes **314-318** can encode, independent of the other hierarchical storage nodes, the segment, e.g., based on an erasure coding method, to generate a number of fragments of the segment. In some embodiments, the hierarchical storage node encodes the segment using an encode/decode module similar to the encode/decode module **418**. In FIG. **11**, the segments "S**1**," "S**2**," and "S**3**," are each encoded to generate eight fragments F**1**-F**8**. Each of the hierarchical storage node stores the fragments, F**1** to F**8**, across the storage devices of the storage subsystem **306**. In some embodiments, the techniques involved in encoding a data segment to generate the fragments of a segment and storing the fragments across the storage devices is similar to the techniques involved in encoding a data object to generate the fragments of the data object and storing the fragments across the storage devices in wide spreading storage architecture, e.g., as described at least with reference to FIGS. **4** and **6**.

[0129] For storing the fragments across a set of storage devices, the hierarchical storage node determines a storage layout of the fragments. The storage layout identifies one or more of the storage racks, storage shelves of a rack and storage devices of a storage shelf the data fragments have to be stored in. In some embodiments, the hierarchical storage node determines the storage layout of the fragments using a storage layout module similar to the storage layout module **420**. After the storage layout is determined, the hierarchical storage node stores the fragments in the identified storage devices. In some embodiments, the hierarchical storage node writes the fragments to the different storage devices in parallel. In the hierarchical spreading storage architecture, the writes are more efficient than current storage systems. For example, in addition to writing the fragments of a particular segment in parallel, all the hierarchical storage nodes can write the fragments of their corresponding segments in parallel.

[0130] The hierarchical storage node stores the segment identifier of the data segment and the fragment identifiers of the fragments of the data segment in a staging area similar to the staging area **408**. Further, the hierarchical storage node stores a mapping of the segment identifier of a segment to the fragment identifiers of the segment in a mapping structure similar to the mapping structure **414**.

[0131] In the hierarchical spreading storage architecture, the storage resiliency provided for a data object is split across the tiers of a storage system. For example, if the storage resiliency offered for a data object by the storage system **1100** is 30%, then the first tier—hierarchical storage node **314-318** provides 15% of the storage resiliency and the second tier—storage devices provided the other 15%. The amount of storage resiliencies provided by each of the tiers can be configurable. However, the sum of storage resiliencies offered by the tiers may not exceed the total storage resiliency offered by the storage system **1100**.

[0132] Referring to the read requests, when a read request arrives at the storage system **1100** from the client **312***a* for a particular data object, the data object can be reconstructed by

obtaining at least k' number of the n'data segments and decoding them to regenerate the data object. The transceiver module **432** obtains the storage layout of the segments from the storage layout module **420** and obtains the data segments from the identified hierarchical storage nodes. The storage layout module **420** can obtain the segment identifiers of the segments of the data object from the mapping structure **414** and then determine from the storage layout the hierarchical storage nodes at which the corresponding segments are stored.

[0133] After the hierarchical storage nodes are identified, the transceiver module **432** requests the hierarchical storage nodes to return the data segments of the data object. The transceiver module **432** can obtain k' to n' number of segments for generating the data object. For example, the transceiver module **432** can stop fetching the segments after obtaining the first k'segments. In another example, the transceiver module **432** can fetch all the n'segments but use only the first k'segments for regenerating the data object. Further, the transceiver module **432** can preferentially select a subset of identified the hierarchical storage nodes to obtain the segments from. The transceiver module **432** selects a hierarchical storage node based on a number of factors, e.g., a latency of the hierarchical storage node, a workload of the hierarchical storage node, a geographical location of the storage device. In some embodiments, the transceiver module **432** can obtain the segments from different storage nodes in parallel.

[0134] When a particular hierarchical storage node receives a request from the front-end subsystem **310** for a data segment, the hierarchical storage node obtains the fragments of the data segment from the storage devices associated with the hierarchical storage node. The hierarchical storage node determines the storage layout of the fragments and obtains a sufficient number of the data fragments, e.g., the minimum number data fragments required to generate the data segment, from the identified storage devices.

[0135] Further, the hierarchical storage node can preferentially select a subset of the storage devices to obtain the fragments from. The hierarchical storage node selects a storage device based on a number of factors, e.g., read latency of storage device, type of the storage device, number of pending read requests ahead of the current read request in a read request queue of the storage device, how far the storage device is. Accordingly, the hierarchical storage node may not even read some of the storage devices that contain the data fragments of the data object, thereby minimizing read/write operations on a particular storage device. In some embodiments, the hierarchical storage node can obtain the fragments in parallel.

[0136] After obtaining the data fragments, the hierarchical storage node decodes the data fragments, e.g., based on the erasure coding used to encode the data segment, to generate the data segment, and then returns the data segment to the front-end subsystem **310**. In some embodiments, the hierarchical storage node may perform additional processes on the decoded data segment before returning it to the front-end subsystem **310**. For example, the hierarchical storage node can perform decompression and de-deduplication on the decoded data segment if the data segment was deduplicated and compressed.

[0137] After the front-end subsystem **310** obtains sufficient number of the data segments from the hierarchical storage nodes, the front-end subsystem **310** decodes the data segments to generate the data object, and returns the data object to the client system **312***a*. In some embodiments, the storage processing module **430** may perform additional processes on the decoded data object before returning the data object to the client **312***a*. For example, the storage processing module **430** can perform decompression and de-deduplication on the decoded data object if the data object was deduplicated and compressed.

[0138] As described above, the hierarchical spreading storage architecture distributes the storage resiliency provided to the data across the storage tiers—hierarchical storage nodes **314-318** and storage devices of the storage subsystem **306**. One of the advantages of such a distributed storage resiliency is that the storage system **1100** can withstand the loss of either some of the hierarchical storage nodes or some of the storage devices of a hierarchical storage node, or in some cases, both.

[0139] Another advantage of the hierarchical spreading storage architecture is that the rebuilding process can be localized in some cases. That is, when a storage device associated with a particular hierarchical storage node fails, the data fragments of a segment stored at the failed storage device may be rebuilt using the remaining data fragments of the segment stored within the storage shelves of the particular hierarchical storage node. The storage system **1100** may not have to obtain the fragments from the storage devices associated with another hierarchical storage node. For example, when a fragment F**1** of the segment S**1** is lost due to a failure of a storage device in the storage shelves **306***a-b*, the hierarchical storage node rebuilds a new data fragment for the data segment S**1** using the remaining data fragments, F**2**-F**8**, stored at other storage devices within the storage shelves **306***a-b*. In some embodiments, the hierarchical storage node uses sufficient number of the data fragments, e.g., k number of the remaining data fragments to rebuild the new data fragment. The hierarchical storage node can use the encoding method used to generate the initial fragments to regenerate the new data fragment.

[0140] Localizing the rebuilding process to a particular hierarchical storage node minimizes the network traffic, e.g., between the hierarchical storage nodes and the front-end subsystem **310**, between the hierarchical storage nodes, that might otherwise occur if the fragments are to be read from storage devices apart from that of the particular hierarchical storage node. This saves the time required for the fragments to traverse the network and therefore, can make the rebuilding process faster and more efficient. Further, localizing the rebuilding process to the storage devices of the particular hierarchical storage node, the read-write operations performed on storage devices of other hierarchical storage nodes is minimized, and therefore the wear of other storage devices is minimized.

[0141] The hierarchical storage node can rebuild the data fragments of all the data segments whose storage resiliency is affected or a subset of those data segments. In some embodiments, the hierarchical storage node rebuilds the data fragments for a particular data segment if the current storage resiliency of the data segment is below the minimum storage resiliency to be provided for the data segment, e.g., as described with reference to rebuilding the data fragments in FIGS. **4** and **8**.

[0142] However, when a particular hierarchical storage node fails or a current storage resiliency of a data segment stored by the particular hierarchical storage node drops below the minimum storage resiliency the storage system **1100** uses the fragments from other hierarchical storage nodes to rebuild

the lost fragments. For example, when the hierarchical storage node **314** fails, the front-end subsystem **310** obtains all or some of the remaining segments S**2** and S**3** from the remaining hierarchical storage nodes, generates a new segment S**4** (not illustrated) and transmits it to another hierarchical storage node or one of the hierarchical storage nodes **316** and **318**, which further encodes the new segment into fragments and stores them at its associated storage devices.

[0143] The hierarchical spreading storage architecture can also be used to store metadata of the data received from a client of the storage system **1100**. FIG. **12** is a block diagram **1200** for storing metadata of a data object with the data object in a storage system **1100** of FIG. **11**, consistent with various embodiments. The hierarchical spreading storage architecture can provide the same storage resiliency to the metadata of a data object that is provided to the data object. Examples of metadata can include, object ID, object size, object owner, creation time, created by, modified by, client-specified metadata, etc. Typically, metadata is stored separate from the data object. The hierarchical spreading storage architecture enables storing the metadata with the data object, thereby eliminating the need to have a separate database for metadata, the need to have specific infrastructure in place to ensure the metadata is consistent with the data, etc.

[0144] When a write request is received at the storage system **1100**, the payload data in the write request is analyzed to obtain the metadata **510** and the data portion, e.g., data object **405**. The data object **405** is then encoded, e.g., using encode/decode module **418**, to generate a number of segments **1205**, e.g., as described with reference to FIG. **11**. The metadata **510** is combined with each of the segments **1205**, e.g., concatenated or prefixed to each of the segments **1205**, to generate composite segments **1210**. In some embodiments, the metadata **510** can be a subset of the metadata of the data object **405**. The composite segments **1210** can then be sent to a number of hierarchical storage nodes, e.g., as described with reference to FIG. **11** for further storage at a set of storage devices associated with the hierarchical storage nodes.

[0145] When a particular hierarchical storage node receives a composite data segment, it encodes the composite data segment to generate a number of data fragments such as fragments **1215**. The metadata **510** is combined with each of the fragments **1215**, e.g., concatenated or prefixed to each of the fragments **1215**, to generate composite fragments **1220**. The composite fragments **1220** can then be stored at the storage devices associated with the hierarchical storage node, e.g., as described with reference to FIG. **11**.

[0146] Note that though FIG. **12** illustrates combining metadata **510** with both the data segments and the fragments, the metadata **510** can be combined with either the data segments or the data fragments.

[0147] In some embodiments, by storing the metadata **510** with the data object **405**, the possibility of inconsistency between the metadata **510** and the data object **405** is eliminated. Further, since the metadata **510** is attached to the segments **1205** and/or fragments **1215**, the composite segments **1210** can be moved across hierarchical storage nodes and the composite fragments **1220** can be moved across storage devices without having to update the metadata **510** and without risking the consistency between the metadata **510** and the data object **405**.

[0148] In some embodiments, another benefit of storing the metadata **510** with the data object **405** is that since a separate database and/or metadata server is not needed to maintain the metadata **510**, the read and write operations are relatively faster since no separate read/write is required to read/write the metadata **510**. In some embodiments, metadata retrieval is also simplified since a method call that is used for retrieving the data object **405** can be modified to use retrieve the metadata **510**, which can simplify a number of functions performed related to the metadata **510**.

[0149] FIG. **13** is a flow diagram of a process **1300** of storing data to an object-based storage system using hierarchical spreading storage architecture, consistent with various embodiments of the disclosed technology. In some embodiments, the process **1300** may be implemented in environment **300** of FIG. **3**, and using the storage system **1100** of FIG. **11**. The process **1300** begins at block **1305**, and at block **1310**, a request module **416** of the frontend subsystem **310** receives a write request including payload data. In some embodiments, the payload data includes data portion and metadata of the data. If the data portion is not in a format suitable for storing in an object storage system, e.g., storage subsystem **306**, the frontend subsystem **310** converts the data portion to the suitable format, e.g., as the data object.

[0150] At block **1315**, the encode/decode module **418** encodes the data object to generate a number of encoded data segments, e.g., encoded data segments S**1**-S**3**. In some embodiments, the encode/decode module **418** encodes the data object based on an erasure coding technique. The number of encoded data segments generated can be expressed as a function, e.g., n'=k'+m', where variable k' is the original amount of data segments or the minimum number of data segments required to regenerate or rebuild the data object, and variable m'stands for the extra or redundant segments that are added to provide protection from storage device/storage node failures. The variable n' is the total number of segments created after the encoding process.

[0151] After the encoded data segments are generated, a mapping of the object identifier and the segment identifiers of the encoded data segments are stored in the mapping structure **414** in the staging area **408**.

[0152] In some embodiments, apart from encoding the data object to generate the fragments, various other storage efficiency processes may be performed on the data object, e.g., deduplication, compression, encryption. One or more of these processes can be performed by the storage processing module **430**.

[0153] At block **1320**, the storage layout module **420** determines a storage layout for sending the encoded data segments across a number of hierarchical storage nodes, e.g., hierarchical storage nodes **314**-**318**. In some embodiments, the storage layout module **420** is configured to spread the encoded data segments across as many hierarchical storage nodes as possible, e.g., to provide better storage resiliency to the data object. That is, the storage layout module **420** attempts to identify different hierarchical storage nodes for storing different encoded data segments. In some embodiments, the storage layout module **420** selects the hierarchical storage nodes on a random basis. In some embodiments, the storage layout module **420** selects the hierarchical storage nodes on a random weighted basis. In some embodiments, the random weighted basis attempts to store the data segments evenly across the hierarchical storage nodes. For example, one type of weighting is to decrease the weight if there are already a specified number of segments stored at the hierarchical storage node. In some embodiments, the random weighted basis randomly identifies the hierarchical storage nodes at which

the encoded data segments are to be stored as a function of decreasing the risk of data loss. For example, if a particular geographical region is prone to higher number of device failures, then the storage nodes in that geographical region may be weighted less so that a lower number of segments are written to the storage nodes in that geographical region.

[0154] At block **1325**, the transceiver module **432** transmits the encoded data segments to the identified hierarchical storage nodes. For example, the transceiver module **432** can transmit the encoded data segments S**1**-S**3** to hierarchical storage nodes **314**-**318**, respectively.

[0155] At block **1330**, each of the hierarchical storage that receives an encoded data segment, processes the encoded data segment to store it at a set of storage devices associated with the hierarchical storage node, and the process **1300** returns. The processing can include encoding the data segment to generate a number of data fragments (block **1331**). For example, the hierarchical storage node **314** encodes the data segment to generate fragments F**1**-F**8**. In some embodiments, the hierarchical storage node encodes the data segment based on an erasure coding technique. Also, the erasure coding technique used to generate the data segments can be different from that used for generating the fragments from the segment.

[0156] The hierarchical storage node includes a storage layout module, e.g., similar to the storage layout module **420**, that determines a storage layout for storing the data fragments at a set of storage devices associated with the hierarchical storage node (block **1332**). In some embodiments, the storage layout module is configured to spread the encoded data fragments across as many storage devices as possible, e.g., to provide better storage resiliency to the data object. After the storage layout is determined, the hierarchical storage node stores the encoded data fragments at the identified storage devices (block **1333**).

[0157] In some embodiments, the front-end subsystem **310** also stores the metadata of the data object with the data segments and/or fragments. Additional details with respect to the process of storing the metadata is described at least with reference to FIGS. **9** and **17**.

[0158] FIG. **14** is a flow diagram of a process **1400** of reading data from an object-based storage system using hierarchical spreading storage architecture, consistent with various embodiments of the disclosed technology. In some embodiments, the process **1400** may be implemented in environment **300** of FIG. **3**, and using the storage system **1100** of FIG. **11**. The process **1400** begins at block **1405**, and at block **1410**, a request module **416** of the frontend subsystem **310** receives a read request, e.g., from a client system **312***a*, for obtaining a data object. In some embodiments, the read request includes an object identifier of the data object.

[0159] At block **1415**, the fragment/segment identification module **422**, determines the encoded data segments of the data object using the object identifier. In some embodiments, a mapping of the object identifier and the encoded data segments are stored in the mapping structure **414** in the staging area **408**.

[0160] At block **1420**, the storage layout module **420** determines the storage layout of the encoded data segments using the mapping obtained from the mapping structure **414**. The storage layout can include identification information of the hierarchical storage nodes where each of the encoded data segments are stored.

[0161] At block **1425**, the transceiver module **432** identifies the hierarchical storage nodes that store sufficient number of the encoded data segments required to generate the data object. In some embodiments, the sufficient number of encoded data segments is k' number of the encoded data segments. In some embodiments, the transceiver module **432** can obtain k' to n' number of segments. For example, the transceiver module **432** can stop fetching the segments after obtaining the first k'segments. In another example, the transceiver module **432** can fetch all the n' segments but use only the first k'segments for regenerating the data object.

[0162] Further, the transceiver module **432** can preferentially select a subset of the identified hierarchical storage nodes to obtain the segments from. The transceiver module **432** can select a hierarchical storage node based on a number of factors, e.g., a read latency of the hierarchical storage node, type of the storage devices associated with hierarchical storage node, number of pending read requests ahead of the current read request in a read request queue of the hierarchical storage node, a geographical location of the hierarchical storage node.

[0163] After the hierarchical storage nodes are identified, the transceiver module **432** requests each of the hierarchical storage nodes for the data segment.

[0164] At block **1430**, each of the identified hierarchical storage nodes performs a number of steps, e.g., **1431**-**1433**, to obtain the data segment. At block **1431**, the hierarchical storage node determines from a storage layout of the fragments, the set of storage devices that store sufficient number of the encoded data fragments required to generate the data segment. In some embodiments, the sufficient number of encoded data fragments is k number of the encoded data fragments. In some embodiments, the hierarchical storage node can obtain k to n number of fragments. For example, the hierarchical storage node can stop fetching the fragments after obtaining the first k fragments. In another example, the hierarchical storage node can fetch all the n fragments but use only the first k fragments for regenerating the data segment.

[0165] Further, the hierarchical storage node can preferentially select a subset of the identified storage devices to obtain the fragments from. The hierarchical storage node can select a storage device based on a number of factors, e.g., a read latency of the storage device, a type of the storage device, number of pending read requests ahead of the current read request in a read request queue of the storage device, a geographical location of the storage device. At block **1432**, the hierarchical storage node obtains the sufficient number of fragments from the identified set of storage devices.

[0166] At block **1433**, after obtaining the encoded data fragments, the hierarchical storage node decodes the encoded data fragments, e.g., based on the erasure coding method used to encode the data segment, to generate the data segment. After generating the data segment, the hierarchical storage node returns the data segment to the front-end subsystem **310**. In some embodiments, additional processes may be performed before decoding the data fragments. For example, the hierarchical storage node can decrypt the encoded data fragments if they were encrypted before being stored. In some embodiments, additional processes may be performed on the decoded data segment before the data segment is returned to the front-end subsystem **310**. For example, the hierarchical storage node can perform decompression and dededuplication on the decoded data segment if the data segment was deduplicated and compressed.

[0167] After obtaining sufficient number of the encoded data segments, at block **1435**, the encode/decode module **418**

of the front-end subsystem **310** decodes the encoded data segments, e.g., based on the erasure coding method used to encode the data object, to generate the data object.

[0168] At block **1440**, the transceiver module **432** transmits the data object in response to the read request, e.g., to the client system **312***a*, and the process **1400** returns. In some embodiments, additional processes may be performed before decoding the data segments. For example, the storage processing module **430** can decrypt the encoded data segments if they were encrypted before being stored. In some embodiments, additional processes may be performed on the decoded data object before it is returned to the client **312***a*. For example, the storage processing module **430** can perform decompression and de-deduplication on the decoded data object if the data object was deduplicated and compressed.

[0169] FIG. **15** is a flow diagram of a process **1500** of rebuilding data fragments of a data object in hierarchical spreading storage architecture, consistent with various embodiments of the disclosed technology. In some embodiments, the process **1500** may be implemented in environment **300** of FIG. **3**, and using the storage system **1100** of FIG. **11**. In some embodiments, the data fragments stored in the storage subsystem **306** may be lost due to a failure of a storage device. The process **1500** begins at block **1505**, and at block **1510**, a hierarchical storage node detects a failure of a storage device, e.g., storage device **304**, associated with the hierarchical storage node. In some embodiments, the failure can be one or more of the storage device being not accessible, the storage device being physically damaged, the storage device determined to fail in a specified period, the storage device determined to fail in a specified number of read/write operations, etc.

[0170] At block **1515**, the hierarchical storage node identifies the encoded data fragments that were stored at the storage device. For example, the hierarchical storage node can refer to the storage layout to determine the fragments stored at the storage device that has failed.

[0171] At block **1520**, the hierarchical storage node identifies the one or more data segments corresponding to the identified encoded data fragments. For example, the hierarchical storage node can refer to the mapping structure to determine the data segments associated with the identified encoded data fragments.

[0172] At block **1525**, the hierarchical storage node rebuilds some or all of the encoded data fragments that was stored at the storage device that failed. In some embodiments, rebuilding the data fragments include performing the method described in association with blocks **1526-1530** for each of the identified data segments.

[0173] At block **1526**, the hierarchical storage node identifies the storage devices where the data fragments of the identified data segment are stored. The hierarchical storage node may use the storage layout determined by the storage layout module of the node to identify the storage devices that store the data fragments of the data segment. At block **1527**, the hierarchical storage node computes the current storage resiliency of the data segment. In some embodiments, storage resiliency is defined as a resistance to loss of one or more storage devices storing a portion of a data segment or resistance to loss of one or more fragments of the data segment. In some embodiments, a current storage resiliency of a data segment is determined as a function of the number of fragments remaining out of n fragments and k. For example, if n is "10," k is "8," the number of redundant fragments, m is "2," and therefore, the storage resiliency can be calculated as 25% (m/k*100). Note that the storage resiliency can be calculated using other functions and based on several other parameters. The storage system **1100** may guarantee a storage resiliency range to the clients of the storage system, for example, a minimum storage resiliency and a maximum storage resiliency. In some embodiments, the storage resiliency range is part of the SLO guaranteed to the clients. In some embodiments, the storage system **1100** may not rebuild the lost data fragments until the current storage resiliency of the data segment is or below the minimum storage resiliency.

[0174] At determination block **1528**, the hierarchical storage node determines if the current storage resiliency of the data segment is less than the minimum storage resiliency. Responsive to a determination that the current storage resiliency of the data segment is not less than the minimum storage resiliency, the process **1500** returns. On the other hand, responsive to a determination that the current storage resiliency is less than the minimum storage resiliency, at block **1529**, the hierarchical storage node obtains sufficient number of fragments of the data segment stored at the identified storage devices (e.g., identified in block **1526**). In some embodiments, the hierarchical storage node can obtain the minimum number of fragments required to rebuild the data fragments.

[0175] At block **1529**, the hierarchical storage node generates the replacement data fragments as a function of the obtained data fragments, and at block **1530**, the hierarchical storage node stores the regenerated data fragments in at least a subset of the remaining storage devices. In some embodiments, the hierarchical storage node regenerates as many data fragments as required to meet a specified storage resiliency, which can be up to maximum storage resiliency. In some embodiments, regenerating the data fragments as a function of the obtained data fragments includes decoding the obtained data fragments to generate the data segment and encoding the generated data segment to generate the specified number of data fragments. In some embodiments, the hierarchical spreading storage performs the encoding and decoding using an erasure coding method.

[0176] FIG. **16** is a flow diagram of a process **1600** of rebuilding data segments of a data object in hierarchical spreading storage architecture, consistent with various embodiments of the disclosed technology. In some embodiments, the process **1600** may be implemented in environment **300** of FIG. **3**, and using the storage system **1100** of FIG. **11**. In some embodiments, the data segments stored by a hierarchical storage node may be lost due to a failure of a storage device and/or a hierarchical storage node. The process **1600** begins at block **1605**, and at block **1610**, a failure detection module **424** of front-end subsystem **310** detects a failure of a hierarchical storage node and/or a failure of one or more storage devices of the hierarchical storage node that caused the storage resiliency of a particular data segment to drop. In some embodiments, the failure can be one or more of the storage device being not accessible, the storage device being physical damaged, the hierarchical storage node not being accessible, the storage device determined to fail in a specified period, the storage device determined to fail in a specified number of read/write operations, etc.

[0177] At block **1615**, the fragment/segment identification module **422** identifies the encoded data segment stored by the hierarchical storage device. For example, the fragment/seg-

ment identification module **422** can refer to the storage layout to determine the segments stored at the particular hierarchical storage node that has failed.

[0178] At block **1620**, the fragment/segment identification module **422** identifies the data object to which the encoded data segment corresponds. For example, the fragment/segment identification module **422** can refer to the mapping structure to determine the data segments associated with the identified data object.

[0179] At determination block **1625**, the regeneration module **428** computes the current storage resiliency of the data object and determines if the storage resiliency of the object is below the specified minimum storage resiliency. In some embodiments, a current storage resiliency of a data object is determined as a function of the number of segments remaining out of n'segments and k'. For example, if n' is "10," k' is "8," the number of redundant segments, m' is 2, and therefore, the storage resiliency can be calculated as 25% (m/k*100). Note that the storage resiliency can be calculated using other functions and based on several other parameters. In some embodiments, the storage system **1100** may not rebuild the lost data segments until the current storage resiliency of the data object is or below the minimum storage resiliency.

[0180] Responsive to a determination that the current storage resiliency of the data object is not less than the minimum storage resiliency, the process **1600** returns. On the other hand, responsive to a determination that the current storage resiliency is less than the minimum storage resiliency, at block **1630**, the transceiver module **432** obtains sufficient number of segments of the data object stored at other hierarchical storage nodes. In some embodiments, the transceiver module **432** obtains the segments of the data object stored at other hierarchical storage nodes as described with at least with reference to blocks **1425-1433** of FIG. **14**.

[0181] At block **1635**, the regeneration module **428** generates the replacement data segment as a function of the obtained data segments. In some embodiments, the regeneration module **428** generates as many data segments as required to meet a specified storage resiliency for the data object, which can be up to a specified maximum storage resiliency of the data object. In some embodiments, regenerating the data segments as a function of the obtained data segments includes decoding the obtained data segments to generate the data object and encoding the generated data object to generate the specified number of data segments. In some embodiments, the hierarchical spreading storage performs the encoding and decoding using an erasure coding method.

[0182] At block **1640**, the transceiver module **432** sends the regenerated data segments to one or more of the remaining storage devices for storage at their associated storage devices. In some embodiments, the transceiver module **432** transmits the replacement data segments of the data object to other hierarchical storage nodes as described with at least with reference to blocks **1320-1333** of FIG. **13**.

[0183] FIG. **17** is a flow diagram of a process **1700** of deferred rebuilding of data segments of a data object in the hierarchical spreading storage architecture, consistent with various embodiments of the disclosed technology. In some embodiments, the process **1700** may be implemented in environment **300** of FIG. **3**, and using the storage system **1100** of FIG. **11**. The rebuilding/regeneration process **1600** can consume significant system resources for regenerating the encoded data segments, e.g., network resources for reading at least k' number of encoded data segments from other hierarchical storage nodes, computing resources of the corresponding hierarchical storage nodes in obtaining the fragments of the corresponding data segment and decoding them to generate the encoded data segment, etc. In some embodiments, the consumption of the system resources can be minimized by postponing or deferring the regeneration process **1600** until a later time, e.g., when the storage devices are replaced with new storage devices, when the data in the storage devices is migrated, etc.

[0184] In some embodiments, the generation of replacement data segments for the lost data segments is deferred until after one or more of the failed storage devices and/or one or more of the hierarchical storage nodes is replaced. That is, the regeneration process may not be executed during the lifetime of the storage devices and/or the hierarchical storage nodes. In some embodiments, the timing of the regeneration process is controlled based on m', the number of redundant encoded data segments to be generated. As described above at least with reference to the regeneration process **1600**, the regeneration process **1600** is triggered when the current storage resiliency of the data object drops below the minimum storage resiliency. The storage resiliency of a data object is a function of the total number of encoded data segments, n', stored across the hierarchical storage nodes, which is a function of m'. The m' can be determined such that the storage resiliency of the data object does not drop below the minimum storage resiliency during the lifespan of one or more of the storage devices. In other words, the number of encoded data segments generated are such that a loss of a subset of the encoded data segments does not drop the storage resiliency of the data object below the minimum storage resiliency during the lifespan of one or more of the storage devices. The following paragraphs describe the process **1700** in further detail.

[0185] The process **1700** begins at block **1705**, and at block **1710**, the regeneration module **428** obtains the historical information regarding a failure rate of storage devices of the type of the storage devices in the environment **300**. The historical information can include a number of parameters that can describe and/or help determine the failure information of a storage device, e.g., an annual failure rate (AFR) of the storage device of a particular type, an AFR of the storage device based on a particular workload on the storage device, how long a storage device is expected to survive based on a particular workload. Such historical information can be gathered from various sources, gathered from the environment **300** over a period and/or can be input by a user such as an administrator of the environment **300**.

[0186] At block **1715**, the regeneration module **428** predicts the failure rate of the storage devices in the environment **300** and generates the predicted information. The regeneration module **428** can interpolate the historical information with various parameters of the storage devices in the environment **300**, e.g., the number of storage devices in the environment **300**, a workload of the storage devices, the number of read/write operations performed on the storage devices, a remaining life of the storage devices, and determine the predicted failure rate of the storage devices.

[0187] At block **1720**, the regeneration module **428** determines the lifespan of the storage devices as a function of the historical information and the predicted information. At block **1725**, the regeneration module **428** determines a statistical probability of a loss of a failure of one or more hierarchical storage nodes based on the determined lifespan of the storage devices. In some embodiments, a failure/loss of a

hierarchical storage node is a function of the lifespan of the set of storage devices associated with the hierarchical storage node since a failure of one or more storage devices from the set can result in a failure of the hierarchical storage node. Further, a failure of the hierarchical storage node can result in a loss of the encoded data segment stored at the hierarchical storage node.

[0188] At block **1730**, the regeneration module **428** determines the redundant number of encoded data segments, m', to be generated for the data object based on the statistical probability of the loss of the hierarchical storage node. The regeneration module **428** notifies the encode/decode module **418** regarding the determined m', and the encode/decode module **418** encodes the data object to generate the encoded data segments accordingly.

[0189] In some embodiments, the regeneration module **428** may continuously adjust m', e.g., based on a specified schedule or certain events such as when storage devices are added or removed, to factor in any change in the parameters of the environment **300**, e.g., change in workload on the storage devices, addition or removal or storage devices, etc.

[0190] Note that although the process **1700** is described as being performed by the regeneration module **428**, the process **1700** can be performed by a combination of modules of the front-end subsystem **310** and/or sub-modules of the regeneration module **428** (not illustrated).

[0191] FIG. **18** is a flow diagram of a process **1800** of processing metadata and data fragments of a data object in hierarchical spreading storage architecture, consistent with various embodiments of the disclosed technology. In some embodiments, the process **1800** may be implemented in environment **300** of FIG. **3**, and using the storage system **1100** of FIG. **11**. In some embodiments, the process **1800** is an implementation of the method of block **925** of FIG. **9**. The data piece generated in the process **900** of FIG. **9**, e.g., in block **920**, can be considered as a data segment in the hierarchical spreading storage architecture. The process **1800** begins at block **1805**, and at block **1810**, the metadata processing module **426** combines the metadata of a data object, e.g., metadata **510**, with each of the segments, e.g., segments **1205**, to generate composite segments, e.g., composite segments **1210**. In some embodiments, combining the metadata with data segment can include concatenating the metadata with segment or prefixing a segment with the metadata. In some embodiments, the metadata **510** combined with segment can be a subset of the metadata of the data object **405**.

[0192] After the composite segments are generated, at block **1815**, the transceiver module **432** transmits the composite segments to a number of hierarchical storage nodes, e.g., as described at least with reference to blocks **1320** and **1325** of FIG. **13** for further storage at a set of storage devices associated with the hierarchical storage nodes.

[0193] At block **1820**, when a particular hierarchical storage node receives a composite data segment, it encodes the composite data segment to generate a number of data fragments, e.g., fragments **1215** (block **1821**). In some embodiments, the composite data segment is encoded to generate a number of data fragments as described at least with reference to block **1331** of FIG. **13**.

[0194] At block **1822**, the particular hierarchical storage node combines each of the fragments with the metadata, e.g., concatenates or prefixes the fragments **1215** with the metadata **510**, to generate the composite fragments, e.g., composite fragments **1220**.

[0195] After the composite fragments are generated, at block **1823**, the particular hierarchical storage node stores the composite fragments at a set of storage devices associated with the hierarchical storage node, e.g., as described with reference to blocks **1332** and **1333** of FIG. **13**.

[0196] Note that although FIG. **18** illustrates combining metadata **510** with both the data segments and the fragments, the metadata **510** can be combined with either the data segments or the data fragments.

[0197] FIG. **19** is a block diagram of a computer system as may be used to implement features of some embodiments of the disclosed technology. The computing system **1900** may be used to implement any of the entities, components or services depicted in the examples of FIGS. **1-17** (and any other components described in this specification). The computing system **1900** may include one or more central processing units ("processors") **1905**, memory **1910**, input/output devices **1925** (e.g., keyboard and pointing devices, display devices), storage devices **1920** (e.g., disk drives), and network adapters **1930** (e.g., network interfaces) that are connected to an interconnect **1915**. The interconnect **1915** is illustrated as an abstraction that represents any one or more separate physical buses, point to point connections, or both connected by appropriate bridges, adapters, or controllers. The interconnect **1915**, therefore, may include, for example, a system bus, a Peripheral Component Interconnect (PCI) bus or PCI-Express bus, a HyperTransport or industry standard architecture (ISA) bus, a small computer system interface (SCSI) bus, a universal serial bus (USB), IIC (I2C) bus, or an Institute of Electrical and Electronics Engineers (IEEE) standard 1394 bus, also called "Firewire".

[0198] The memory **1910** and storage devices **1920** are computer-readable storage media that may store instructions that implement at least portions of the described technology. In addition, the data structures and message structures may be stored or transmitted via a data transmission medium, such as a signal on a communications link. Various communications links may be used, such as the Internet, a local area network, a wide area network, or a point-to-point dial-up connection. Thus, computer readable media can include computer-readable storage media (e.g., "non-transitory" media) and computer-readable transmission media.

[0199] The instructions stored in memory **1910** can be implemented as software and/or firmware to program the processor(s) **1905** to carry out actions described above. In some embodiments, such software or firmware may be initially provided to the computing system **1900** by downloading it from a remote system through the computing system **1900** (e.g., via network adapter **1930**).

[0200] The technology introduced herein can be implemented by, for example, programmable circuitry (e.g., one or more microprocessors) programmed with software and/or firmware, or entirely in special-purpose hardwired (non-programmable) circuitry, or in a combination of such forms. Special-purpose hardwired circuitry may be in the form of, for example, one or more ASICs, PLDs, FPGAs, etc.

Remarks

[0201] The above description and drawings are illustrative and are not to be construed as limiting. Numerous specific details are described to provide a thorough understanding of the disclosure. However, in some instances, well-known details are not described in order to avoid obscuring the description. Further, various modifications may be made

without deviating from the scope of the embodiments. Accordingly, the embodiments are not limited except as by the appended claims.

[0202] Reference in this specification to "one embodiment" or "an embodiment" means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment of the disclosure. The appearances of the phrase "in one embodiment" in various places in the specification are not necessarily all referring to the same embodiment, nor are separate or alternative embodiments mutually exclusive of other embodiments. Moreover, various features are described which may be exhibited by some embodiments and not by others. Similarly, various requirements are described which may be requirements for some embodiments but not for other embodiments.

[0203] The terms used in this specification generally have their ordinary meanings in the art, within the context of the disclosure, and in the specific context where each term is used. Some terms that are used to describe the disclosure are discussed below, or elsewhere in the specification, to provide additional guidance to the practitioner regarding the description of the disclosure. For convenience, some terms may be highlighted, for example using italics and/or quotation marks. The use of highlighting has no influence on the scope and meaning of a term; the scope and meaning of a term is the same, in the same context, whether or not it is highlighted. It will be appreciated that the same thing can be said in more than one way. One will recognize that "memory" is one form of a "storage" and that the terms may on occasion be used interchangeably.

[0204] Consequently, alternative language and synonyms may be used for any one or more of the terms discussed herein, nor is any special significance to be placed upon whether or not a term is elaborated or discussed herein. Synonyms for some terms are provided. A recital of one or more synonyms does not exclude the use of other synonyms. The use of examples anywhere in this specification including examples of any term discussed herein is illustrative only, and is not intended to further limit the scope and meaning of the disclosure or of any exemplified term. Likewise, the disclosure is not limited to various embodiments given in this specification.

[0205] Those skilled in the art will appreciate that the logic illustrated in each of the flow diagrams discussed above, may be altered in various ways. For example, the order of the logic may be rearranged, substeps may be performed in parallel, illustrated logic may be omitted; other logic may be included, etc.

[0206] Without intent to further limit the scope of the disclosure, examples of instruments, apparatus, methods and their related results according to the embodiments of the present disclosure are given below. Note that titles or subtitles may be used in the examples for convenience of a reader, which in no way should limit the scope of the disclosure. Unless otherwise defined, all technical and scientific terms used herein have the same meaning as commonly understood by one of ordinary skill in the art to which this disclosure pertains. In the case of conflict, the present document, including definitions will control.

I/we claim:

1. A computer-implemented method comprising:

identifying, at a storage management computer node of a storage management system, a specified storage device, the specified storage device being one of multiple storage devices associated with the storage management system, the storage management system storing a data object of multiple data objects as a first set of encoded data fragments, the first set of encoded data fragments stored across the storage devices;

identifying, by the storage management computer node, one or more of the data objects to which multiple encoded data fragments stored at the specified storage device correspond, the identifying including identifying that a group of the encoded data fragments correspond to the data object, the group of the encoded data fragments being part of the first set of encoded data fragments; and

regenerating, by the storage management computer node, a subset of the encoded data fragments as a function of a second set of encoded fragments representing the data object, the second set of encoded fragments being a difference between the first set of encoded data fragments and the group of the encoded data fragments, the second set of encoded data fragments stored at a first set of the storage devices, the first set of the storage devices excluding the specified storage device.

2. The computer-implemented method of claim **1** further comprising:

storing, by the storage management computer node, the regenerated subset of the encoded data fragments at a second set of the storage devices, the second set of the storage devices excluding the specified storage device.

3. The computer-implemented method of claim **2**, wherein the first set of the storage devices from which the storage management computer node obtains the subset of the encoded data fragments is same as the second set of the storage devices.

4. The computer-implemented method of claim **2**, wherein the first set of the storage devices from which the storage management computer node obtains the subset of the encoded data fragments is different from the second set of the storage devices.

5. The computer-implemented method of claim **1**, wherein the first set of encoded fragments is generated by encoding the data object, the first set of encoded fragments including a first specified number of encoded data fragments out of which a second specified number of encoded data fragments is required for regenerating the data object.

6. The computer-implemented method of claim **5**, wherein regenerating the subset of the encoded data fragments includes:

obtaining, from the first set of the storage devices, at least the second specified number of encoded fragments from the second set of encoded data fragments,

decoding, by the storage management computer node, the at least the second specified number of encoded data fragments to regenerate the subset of the encoded data fragments, and

storing, by the storage management computer node, the subset of the encoded data fragments at a second set of the storage devices, the second set of the storage devices excluding the specified storage device.

7. The computer-implemented method of claim **6**, wherein the decoding is executed as a function of an erasure coding technique.

8. The computer-implemented method of claim **5**, wherein the regenerating includes:

determining a specified storage resiliency of the data object,

determining a current storage resiliency of the data object, and

generating the subset of the encoded data fragments corresponding to the data object if the current storage resiliency is below the specified storage resiliency by a specified value.

9. The computer-implemented method of claim **8**, wherein the specified storage resiliency is a function of the first specified number of encoded data fragments and the second specified number of encoded data fragments.

10. The computer-implemented method of claim **8**, wherein the current storage resiliency is a function of a number of encoded fragments in the second set of encoded data fragments and the second specified number of encoded data fragments.

11. The computer-implemented method of claim **1**, wherein regenerating the group of the encoded data fragments includes regenerating the subset of the encoded fragments as a background process in the storage management computer node.

12. The computer-implemented method of claim **1**, wherein regenerating the group of the encoded data fragments includes regenerating the subset of the encoded fragments before the specified storage device is replaced with a replacement storage device.

13. The computer-implemented method of claim **12**, wherein the replacement storage device is used for storing a collection of encoded data fragments other than the subset of the encoded data fragments.

14. The computer-implemented method of claim **1** further comprising:

detecting, by the storage management computer node, an addition of a replacement storage device, the replacement storage device replacing the specified storage device; and

using, by the storage management computer node, the first storage device to store a collection of encoded data fragments other than the subset of the encoded data fragments.

15. The computer-implemented method of claim **14**, wherein the replacement storage device has a different storage capacity from that of the specified storage device.

16. The computer-implemented method of claim **1**, wherein identifying the one or more of the data objects to which the encoded data fragments stored at the specified storage device correspond includes:

determining, by the storage management computer node, a storage layout of the encoded data fragments, the storage layout including an identification information of the storage devices at which each of the encoded data fragments is stored.

17. The computer-implemented method of claim **16**, wherein identifying that the group of the encoded data fragments correspond to the data object includes:

determining, by the storage management computer node, the data object based on a mapping in the storage layout, the mapping including a mapping of the data object to the subset of the encoded data fragments of the data object.

18. The computer-implemented method of claim **16** further comprising:

updating, by the storage management computer node, the storage layout to indicate that the subset of the encoded data fragments is stored at a second set of the storage devices, the second set of the storage devices excluding the specified storage device.

19. The computer-implemented method of claim **1**, wherein the data object is encoded to the first set of encoded data fragments as a function of an erasure coding technique.

20. The computer-implemented method of claim **1**, wherein identifying the specified storage device includes identifying at least one of the storage devices that has failed, inaccessible or determined to fail.

21. A computer-implemented method comprising:

identifying, at a storage management computer node of a storage management system, a specified storage device of a set of storage devices associated with a storage computer node, the storage management computer node encoding a data object of multiple data objects to generate multiple encoded data segments, the storage computer node storing an encoded data segment of the encoded data segments as a set of encoded data fragments in the set of storage devices, the set of encoded data fragments including a first specified number of encoded data fragments out of which a second specified number of encoded data fragments is required for regenerating the encoded data segment,

wherein the storage computer node is one of multiple storage computer nodes, each of the storage computer nodes encoding at least one of the encoded data segments to generate a corresponding set of encoded data fragments and storing the corresponding set of encoded data fragments in a corresponding set of storage devices;

determining, using the storage computer node, an encoded data fragment of a group of encoded data fragments stored at the specified storage device, the group of encoded data fragments corresponding to one or more encoded data segments of one or more of the data objects;

identifying, by the storage computer node, the encoded data segment to which the encoded data fragment corresponds; and

generating, by the storage computer node, a replacement encoded data fragment as a function of at least the second specified number of encoded data fragments stored at one or more of a remaining set of the set of storage devices.

22. The computer-implemented method of claim **21** further comprising:

storing, by the storage management computer node, the replacement encoded data fragment at one of the one or more of the remaining set of the set of storage devices.

23. The computer-implemented method of claim **21**, wherein generating the replacement encoded data fragment includes:

obtaining, the at least the second specified number of encoded data fragments from the one or more of the remaining set of the set of storage devices,

encoding, by the storage computer node, the at least the second specified number of the encoded data fragments to generate the replacement encoded data fragment, and

storing, by the storage computer node, the replacement encoded data fragment at one of the one or more of the remaining set of the set of storage devices.

24. The computer-implemented method of claim **23**, wherein the encoding is executed as a function of an erasure coding technique.

25. The computer-implemented method of claim **21**, wherein the generating includes:

determining a specified storage resiliency of the encoded data segment,

determining a current storage resiliency of the encoded data segment, and

generating the replacement encoded data fragment if the current storage resiliency is below the specified storage resiliency by a specified value.

26. The computer-implemented method of claim **21** further comprising:

detecting, by the storage management computer node, a failure of the storage computer node;

identifying, by the storage management computer node, the data object to which the encoded data segment stored by the storage computer node corresponds; and

generating, by the storage management computer node, a replacement encoded data segment for the data object as a function of at least a third specified number of encoded data segments of the data object stored at a remaining set of the storage computer nodes.

27. The computer-implemented method of claim **26**, wherein generating the replacement encoded data segment includes:

obtaining, from the remaining set of the storage computer nodes, at least the third specified number of encoded data segments, wherein the data object is encoded to generate a fourth specified number of encoded data segments, which includes the third specified number of encoded data segments required for generating the encoded data segment,

encoding, by the storage management computer node, the at least the third specified number of encoded data segments to generate the replacement encoded data segment, and

sending, by the storage management computer node, the replacement encoded data segment to one of the remaining set of the storage computer nodes for further storage at a first set of storage devices associated with the one of the remaining set of the storage computer nodes.

28. The computer-implemented method of claim **27**, wherein obtaining a first encoded data segment of the at least the third specified number of encoded data segments includes:

obtaining, from a first storage computer node of the remaining set of the storage computer nodes that stores the first encoded data segment, a first set of encoded data fragments corresponding to the first encoded data segment, and

decoding the first set of encoded data fragments to generate the first encoded data segment.

29. The computer-implemented method of claim **26** further comprising:

storing, by one of the remaining set of the storage computer nodes, the replacement encoded data segment as a first set of encoded data fragments.

30. A system comprising:

a processor;

a first module configured to identify a specified storage device, the specified storage device being one of multiple storage devices associated with the system, the system storing a data object of multiple data objects as a first set of encoded data fragments, the set of encoded data fragments stored across the storage devices;

a second module configured to identify one or more of the data objects to which multiple encoded data fragments stored at the specified storage device correspond, the identifying including identifying that a subset of the encoded data fragments correspond to the data object, the subset of the encoded data fragments being part of the first set of encoded data fragments; and

a third module configured to regenerate the subset of the encoded data fragments as a function of a second set of encoded fragments representing the data object, the second set of encoded fragments being a difference between the first set of encoded data fragments and the subset of the encoded data fragments, the second set of encoded data fragments stored at a first set of the storage devices, the first set of the storage devices excluding the specified storage device.

31. A system comprising:

a processor;

a first module configured to identify a specified storage device of a set of storage devices associated with a storage computer node, the system encoding a data object to generate multiple encoded data segments, the storage computer node storing an encoded data segment of the encoded data segments as a set of encoded data fragments in the set of storage devices, the set of encoded data fragments including a first specified number of encoded data fragments out of which a second specified number of encoded data fragments is required for regenerating the encoded data segment,

wherein the storage computer node is one of multiple storage computer nodes, each of the storage computer nodes encoding at least one of the encoded data segments to generate a corresponding set of encoded data fragments and storing the corresponding set of encoded data fragments in a corresponding set of storage devices;

a second module configured to cause the storage computer node to determine an encoded data fragment of a group of encoded data fragments stored at the specified storage device;

a third module configured to cause the storage computer node to identify the encoded data segment to which the encoded data fragment corresponds; and

a fourth module configured to cause the storage computer node to generate a replacement encoded data fragment as a function of at least the second specified number of encoded data fragments stored at one or more of a remaining set of the set of storage devices.

* * * * *