ARCHIVAL DATA STORAGE SYSTEM

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims priority from and the benefit of U.S. Patent Application No. 13/569,591, entitled "DATA STORAGE POWER MANAGEMENT" filed August 8, 2013 (Attorney Docket No. 90204-841816 (054900US)) ; U.S. Patent Application No. 13/570,151, entitled "DATA STORAGE INTEGRITY VALIDATION" filed August 8, 2013 (Attorney Docket No. 90204-841810 (054600US)) and U.S. Application No. 13/570,088, entitled "ARCHIVAL DATA STORAGE SYSTEM" filed August 8, 2013 (Attorney Docket No. 90204-841806 (054000US)). This application incorporates by reference for all purposes the full disclosure of co-pending U.S. Patent Application No. 13/569,984, filed concurrently herewith, entitled "LOG-BASED DATA STORAGE ON SEQUENTIALLY WRITTEN MEDIA" (Attorney Docket No. 90204-841804 (054800US)), co-pending U.S. Patent Application No. 13/570,057, filed concurrently herewith, entitled "DATA STORAGE MANAGEMENT FOR SEQUENTIALLY WRITTEN MEDIA" (Attorney Docket No. 90204-841817 (055300US)), co-pending U.S. Patent Application No. 13/570,005, filed concurrently herewith, entitled "DATA WRITE CACHING FOR SEQUENTIALLY WRITTEN MEDIA" (Attorney Docket No. 90204-841812 (055000US)), co-pending U.S. Patent Application No. 13/570,030, filed concurrently herewith, entitled "PROGRAMMABLE CHECKSUM CALCULATIONS ON DATA STORAGE DEVICES" (Attorney Docket No. 90204-841813 (055200US)), co-pending U.S. Patent Application No. 13/569,994, filed concurrently herewith, entitled "ARCHIVAL DATA IDENTIFICATION" (Attorney Docket No. 90204-841807 (054300US)), co-pending U.S. Patent Application No. 13/570,029, filed concurrently herewith, entitled "ARCHIVAL DATA ORGANIZATION AND MANAGEMENT" (Attorney Docket No. 90204-841808 (054400US)), co-pending U.S. Patent Application No. 13/570,092, filed concurrently herewith, entitled "ARCHIVAL DATA FLOW MANAGEMENT" (Attorney Docket No. 90204-841809 (054500US)), co-pending U.S. Patent Application No. 13/569,665, filed concurrently herewith,

entitled "DATA STORAGE INVENTORY INDEXING" (Attorney Docket No. 90204-841811 (054700US)), co-pending U.S. Patent Application No. 13/569,714, filed concurrently herewith, entitled "DATA STORAGE SPACE MANAGEMENT" (Attorney Docket No. 90204-846202 (056100US)), co-pending U.S. Patent Application No. 13/570,074, filed concurrently herewith, entitled "DATA STORAGE APPLICATION PROGRAMMING INTERFACE" (Attorney Docket No. 90204-846378 (056200US)).

BACKGROUND

[0002] With increasing digitalization of information, the demand for durable and reliable archival data storage services is also increasing. Archival data may include archive records, backup files, media files and the like maintained by governments, businesses, libraries and the like. The archival storage of data has presented some challenges. For example, the potentially massive amount of data to be stored can cause costs to be prohibitive using many conventional technologies. Also, it is often desired that the durability and reliability of storage for archival data be relatively high, which further increases the amount of resources needed to store data, thereby increasing the expense. Conventional technologies such as magnetic tapes have traditionally been used in data backup systems because of the low cost. However, tape-based and other storage systems often fail to fully exploit advances in storage technology, such as data compression, error correction and the like, that enhance the security, reliability and scalability of data storage systems.

BRIEF DESCRIPTION OF THE DRAWINGS

[0003] Various embodiments in accordance with the present disclosure will be described with reference to the drawings, in which:

[0004] FIG. 1 illustrates an example series of communications between a customer and an archival data storage service, in accordance with at least one embodiment.

[0005] FIG. 2 illustrates an example environment in which archival data storage services may be implemented, in accordance with at least one embodiment.

[0006] FIG. 3 illustrates an interconnection network in which components of an archival data storage system may be connected, in accordance with at least one embodiment.

[0007] FIG. 4 illustrates an interconnection network in which components of an archival data storage system may be connected, in accordance with at least one embodiment.

[0008] FIG. 5 illustrates an example process for storing data, in accordance with at least one embodiment.

[0009] FIG. 6 illustrates an example process for retrieving data, in accordance with at least one embodiment.

[0010] FIG. 7 illustrates an example process for deleting data, in accordance with at least one embodiment.

[0011] FIG. 8 illustrates an example flow for describing an implementation of the data integrity validation described herein, according to at least one example.

[0012] FIGS. 9-10 illustrate block diagrams for describing at least some features of the data integrity validation described here, according to at least some examples.

[0013] FIGS. 11-13 illustrate example flow diagrams of one or more processes for implementing at least some features of the data integrity validation described herein, according to at least some examples.

[0014] FIG. 14 illustrates an example architecture for implementing data storage power management that includes an archival data storage system configured to manage operation of hard drives.

[0015] FIG. 15 illustrates an additional example architecture for implementing data storage power management that includes an archival data storage system configured to manage operation of hard drives.

[0016] FIGS. 16-18 illustrate example flow diagrams of processes for implementing at least some features of the data storage drive power management described herein, according to at least a few examples.

[0017] FIG. 19 illustrates an environment in which various embodiments can be implemented.

DETAILED DESCRIPTION

[0018] In the following description, various embodiments will be described. For purposes of explanation, specific configurations and details are set forth in order to provide a thorough understanding of the embodiments. However, it will also be apparent to one skilled in the art that the embodiments may be practiced without the specific details. Furthermore, well-known features may be omitted or simplified in order not to obscure the embodiment being described.

[0019] Techniques described and suggested herein include methods and system for providing cost-effective, reliable and scalable archival data storage services. In an embodiment, an archival data storage system allows a customer to store, retrieve and delete archival data objects as well as perform metadata and configuration operations using a programmatic user-interface.

[0020] In an embodiment, an archival data storage service operates such that, from the perspective of users of the service (either human users or automated users) data storage and deletion operations appear to be handled in a synchronous fashion while data retrieval and certain metadata operations appear to take longer time to complete. In an embodiment, when a customer requests that a data object be stored in an archival data storage system, the system provides the customer with a data object identifier in response to the customer request. The data object identifier may be used in subsequent requests to retrieve, delete or otherwise reference the stored data object. In some embodiments, the data object is stored immediately in an archival data storage described herein. In other embodiments, the data object is stored in a transient data store. As used herein, transient data store is used interchangeably with temporary or staging data store to refer to a data store that is used to store data objects before they are stored in an archival data storage described herein or to store data objects that are retrieved from the archival data storage. A transient data store may provide volatile or non-volatile (durable) storage. In most embodiments, while potentially usable for persistently storing data, a transient data store is intended to store data for a shorter period of time than an archival data storage system and may be less cost-effective than the data archival storage system described herein. Where data is stored in a transient data store, a data storage job may be created to eventually store the data object to the archival data storage system. At least some of the data storage job may then be processed after the data object identifier is provided to the customer. As used herein, a "job"

refers to a data-related activity corresponding to a customer request that may be performed after the request is handled or a response is provided.

[0021] In an embodiment, when a customer requests that a data object be retrieved from an archival data storage system, the system creates a data retrieval job associated with the customer request and provides the customer with the job identifier. In an embodiment, the request specifies a data object identifier for the data object to be retrieved, such as issued by the system when the data object is previously stored, described above. Subsequent to providing the job identifier, the system may process the data retrieval job along with other jobs in the system in order to benefit from efficiencies and other advantages of batch processing techniques. The customer may learn of the job's status by receiving a notification from the system, querying the system with the job identifier and the like, and download the retrieved data from a designated transient data store once the job is completed. In an embodiment, certain metadata operations are handled in a similar fashion.

[0022] In an embodiment, when a customer requests that a data object be deleted from an archival data storage system, the system provides the customer with an acknowledgement of the deletion. In some embodiments, the data object is deleted immediately. In other embodiments, the system creates a data deletion job associated with the customer request and processes the data deletion job after providing the acknowledgement of deletion, possibly days or even longer after providing the acknowledgment. In some embodiments, the system allows a grace period for the customer to cancel, undo or otherwise recover the data object.

[0023] In an embodiment, the archival data storage system comprises a front end subsystem that handles customer requests and perform functionalities such as authentication, authorization, usage metering, accounting and billing and the like. Additionally, the front end subsystem may also handle bulk data transfers in and out of the archival data storage system.

[0024] In an embodiments, the archival data storage system comprises a control plane for direct I/O that provides transient durable storage or staging area for incoming (for data storage) and outgoing (for data retrieval) payload data. In addition, the control plane for direct I/O may provide services for creating, submitting and monitoring the execution of jobs associated with customer requests.

[0025] In an embodiments, the archival data storage system comprises a common control plane that provides a queue-based load leveling service to dampen peak to average load coming into the archival data storage system among others. In an embodiment, the common control plane also provides a durable and high-efficiency transient store for job execution that may be used by services in the data plane (described below) to perform job planning optimization, check pointing, recovery, and the like.

[0026] In an embodiments, the archival data storage system comprises a data plane that provides services related to long-term archival data storage, retrieval and deletion, data management and placement, anti-entropy operations and the like. In an embodiment, data plane comprises a plurality of storage entities including storage devices (such as hard disk drives), storage nodes, servers and the like. In various embodiments, storage entities are designed to provide high storage capacity and low operational costs. For example, data plane may include hard drives with Shingled Magnetic Recording (SMR) technologies to offer increased hard drive capacity. As another example, a power management schedule may be used to control which hard drives are active at a given time to conserve power and cooling costs associated with hardware devices. In addition, data plane may also provide anti-entropy operations to detect entropic effects (e.g., hardware failure, data loss) and to initiate anti-entropy routines.

[0027] In an embodiments, the archival data storage system comprises a metadata plane that provides information about data objects stored in the system for inventory and accounting purposes, customer metadata inquiries, anti-entropy processing and the like. In particular, metadata plane may generate a cold index (i.e., an index that is updated infrequently) based on job completion records. In various embodiments, metadata plane services are designed to impose minimal cost overhead using techniques such as batch processing.

[0028] In various embodiments, the archival data storage system described herein is implemented to be efficient and scalable. For example, in an embodiment, batch processing and request coalescing is used at various stages (e.g., front end request handling, control plane job request handling, data plane data request handling) to improve efficiency. For another example, in an embodiment, processing of metadata such as jobs, requests and the like are partitioned so as to facilitate parallel processing of the partitions by multiple instances of services.

[0029] In an embodiment, data elements stored in the archival data storage system (such as data components, volumes, described below) are self-describing so as to avoid the need for a global index data structure. For example, in an embodiment, data objects stored in the system may be addressable by data object identifiers that encode storage location information. For another example, in an embodiment, volumes may store information about which data objects are stored in the volume and storage nodes and devices storing such volumes may collectively report their inventory and hardware information to provide a global view of the data stored in the system. In such an embodiment, the global view is provided for efficiency only and not required to locate data stored in the system.

[0030] In various embodiments, the archival data storage system described herein is implemented to improve data reliability and durability. For example, in an embodiment, a data object is redundantly encoded into a plurality of data components and stored across different data storage entities to provide fault tolerance. For another example, in an embodiment, data elements have multiple levels of integrity checks. In an embodiment, parent/child relations always have additional information to ensure full referential integrity. For example, in an embodiment, bulk data transmission and storage paths are protected by having the initiator pre-calculate the digest on the data before transmission and subsequently supply the digest with the data to a receiver. The receiver of the data transmission is responsible for recalculation, comparing and then acknowledging to the sender that includes the recalculated the digest. Such data integrity checks may be implemented, for example, by front end services, transient data storage services, data plane storage entities and the like described below in connection with FIG. 2.

[0031] Embodiments of the present disclosure are directed to, among other things, validating or otherwise verifying the integrity of data payloads or portions of data payloads intended for storage. In some examples, an archival data storage service may be configured to receive requests to store data, in some cases large amounts of data, in logical data containers or other archival storage devices for varying, and often relatively long, periods of time. In some examples, the archival data storage service may operate or otherwise utilize many different storage devices. For example, the archival storage service may include one or more disk drives

that, when operating, utilize spinning magnetic media. Additionally, the archival data storage service may include one or more racks located in one or more geographic locations (e.g., potentially associated to different postal codes). Each rack may further include one or more, or even hundreds of, hard drives such as, but not limited to, disk drives or the like.

[0032] In some aspects, the archival data storage service may provide storage, access and/or placement of one or more computing resources through a service such as, but not limited to, a web service, a remote program execution service or other network-based data management service. For example, a user, client entity, computing resource or other computing device may access, via the archival data storage service, data storage and/or management such that access mechanisms may be implemented and/or provided to the user or computing device. In some examples, computing resource services, such as those provided by the archival data storage service, may include one or more computing resources accessible across one or more networks through user interfaces (UIs), application programming interfaces (APIs) and/or other interfaces where the one or more computing resources may be scalable and/or expandable as desired.

[0033] In some examples, the archival data storage service may enable users or client entities such as, but not limited to, third-party services that utilize the archival data storage service or other web services associated with the archival data storage service to upload data for potentially long and persistent storage. Unless otherwise contradicted explicitly or clearly by context, the term "user" is used herein to describe any entity utilizing the archival data storage service. The archival data storage service may also wish to ensure the integrity of the archived data and/or guarantee the integrity of the archived data. In order to accomplish these goals, in some examples, the archival data storage service may provide a data object identifier for identifying the data once uploaded. In some cases, the data object identifier may also include a top-level tree digest for validating the archived data even after some extended period of time. Additionally, the data object identifier may also be configured in such a way that its integrity may be validated as well.

[0034] The archival data storage service may be configured to perform operations on data to be stored, such that the data is broken into parts, portions or other demarcated groupings. Once separated into parts, the archival data storage service may perform one or more encryption functions and/or algorithms on the parts including, but not limited to, a hash function or other

cryptographic or other method to produce a digest (referred to also as a hash value, a hash code, a checksum, etc.). In some examples, the data may be validated such that the archival storage service can ensure that the data being stored matches the data received. In this way, data integrity may be validated. Additionally, the sender of the data may be able to independently determine a data chunk size (i.e., the size of the parts of the payload) and not be requested to maintain or persist this information. For example, a user or client entity may request to upload a 1 gigabyte (GB) file in 2 megabyte (MB) portions (chunks). And without saving or otherwise persisting the fact that 2MB chunks were used, the data payload may still be validated. That is, based at least in part on generating one or more digests, checksums, hash codes, etc., for the chunks and at least a digest for the payload (in some examples based at least in part on combinations of the digests), the data may later be partitioned into different sized chunks without losing the ability to be validated.

[0035] In some aspects, this may be accomplished by providing instructions or an algorithm to a user or client entity that indicates the way in which the data should be partitioned and/or hashed. Additionally, in some cases, the archival storage service may expose or otherwise provide one or more API method calls and/or a software development kit (SDK) to enable users to appropriately upload the data in chunks and to facilitate the requested order of operations and/or inclusion of appropriate checksum information. For example, the users may be requested to select a chunk size of at least 1MB, or some other predefined size, for uploading the data. As used herein, the data being uploaded by a user or client entity and/or stored by the archival data storage service may be referred to as a payload. Additionally, the user may select or otherwise instruct the archival data storage service that the payload is to be partitioned into sizes including powers of two of 1MB (e.g., 1MB, 2MB, 4MB, 8MB, 16MB, etc.). In other examples, the payload may be partitioned into sizes including other multiples of a predefined size (e.g., 1MB). The other multiples may be based at least in part on the degree of children of a tree used to represent the data. For example, if a binary tree is used, the integer multiple may include integer powers of two, if a trinary tree is used, the integer multiple may be integer powers of three, if a tree of degree four (i.e., each node may have four children) is used, the integer multiple may include integer powers of four, and so on. The user may then follow an algorithm for generating one or more hash trees (e.g., one hash tree per part when the payload is partitioned) and/or one or

more hash values for each partition. Additionally, the user may generate a hash value (or digest) for each 1MB chunk independent of the selected partition size. In some examples, these digests corresponding to the 1MB chunks of a partition may be included in a hash tree for the partition. Further, in some examples, a root node of each partition's hash tree may be provided to the archival data storage service along with the digests for each 1MB sub-part of the partition. In this way, each sub-part may be validated by the archival data storage service, based at least in part on the 1MB digests, and each partition may be validated by the archival data storage service, based at least in part on the root digest for each part. Additionally, a final root hash associated with a hash of each root hash for the parts may be used by the archival data storage service, based at least in part on comparing the received final root hash and a top-level hash value determined by the archival data storage service.

[0036] In at least one example, generating a hash tree or other hierarchical data structure (e.g., other types of binary trees such as, but not limited to, B-trees and/or other types of data structures such as, but not limited to, arrays, records, etc.) may include concatenating digests and running a hash function on the concatenated digest. For example, in a binary hash tree, a root node may have two children represented by one hash value each. In some cases, generating the root hash value may be based at least in part on concatenating the two children hash values to form a new piece of data and further running the hash function on the new piece of data. The resulting hash value may represent the root hash. As such, each partition of a payload may have its own root hash, although its root hash may be used in calculating the top-level hash for the payload. In some examples, it may also be possible to validate the payload and/or portions of the payload without recalling or otherwise persisting the partition size chosen by the user.

[0037] Embodiments of the present disclosure are directed to, among other things, determining and/or implementing one or more schedules for enabling, operating, providing power to or otherwise controlling one or more storage devices of a storage service. In some examples, the storage service may be an archival data storage service that operates or otherwise utilizes many different computing devices and/or storage devices. For example, the archival storage service may include one or more disk drives that, when operating, utilize spinning magnetic media. Additionally the archival data storage service may include one or more racks located in one or more geographic locations (e.g., potentially associated to different postal codes). Each rack may

further include one or more, or even hundreds of hard drives such as, but not limited to, disk drives or the like. Additionally, rotating media of the hard drives, operating drive controllers and/or processors, etc., generally consume relatively large amounts of power and can be costly.

[0038] In some aspects, an archival data storage service may be configured to minimize power and/or cooling costs associated with storage devices by changing and/or managing hard drive power consumption. By way of example only, the archival data storage service may be configured to control the number of hard drives that may have actively spinning media at a given time. Additionally, in some examples, the archival data storage service may be configured to change the rotation speeds instead of shutting the drive on or off to save power. For example, some drives could be slowed down, and a rotation schedule could rotate which drives to slow down and which drives to run at normal speed. Additionally, in some cases, a data storage node, which may be configured to control a data storage device such as a hard drive, may determine and/or control which hard drives are active and which hard drives are not. A hard drive may be considered active if it is spinning its media, powering up and/or powering down. For example, a spinning hard drive may include one in which the disk is rotating and a head may magnetize locations of the drive to write, read or delete data from the drive while rotating. However, in other examples, a drive may have been activated; however, it may not be fully rotating yet. In this example, the drive may be powering up. Alternatively, once a drive has been activated, the data storage node or other controller may deactivate the drive; however, it may take some time before the drive is no longer spinning. This may be considered powering down. As such, there may be times when the drive is operational but not active. Further, in some examples, activating a data storage device may include providing power to a device that is not currently receiving power or activating a device that is receiving power but is in a sleep mode. As such, a data storage device may be in many states including, but not limited off, on, sleep, power down or power up.

[0039] In some aspects, the data storage nodes may be configured to determine a rotation schedule, provide the rotation schedule and/or activate and deactivate storage devices. Additionally, the storage nodes may report this information to a storage node registrar, queue or other library so that one or more storage node managers may be informed regarding which write, read or delete jobs to execute. It may be possible to execute write jobs at any time, based at least

in part on an assumption that there will generally be enough capacity to handle incoming write requests and place them into whichever drives are active. However, in some examples, the storage node managers may request, or otherwise be instructed, to execute write jobs in particular drives based at least in part on when they are active. In this case, the storage node managers may first request the rotation schedule, or at least an indication of which drives are active, from the storage nodes prior to executing the jobs. It also may be possible to execute write jobs at any time, based at least in part on an assumption that there will generally be enough active hard drives to process write requests at a desired rate, even though some number of drives may not be active.

[0040] In at least one example, rotation schedules may be based at least in part on a percentage or a number of drives to be active at the same time. For example, the archival storage service may determine that only a predefined percentage of the drives should be active at any given time. As such, all other drives may be in a sleep or other low-power state mode, while a predefined percentage may be active. Additionally, the total number of drives may be broken into groups based at least in part on the determined percentage. For example, and without limitation, if the archival storage service determines that 20% of the drives should be active, the set of drives to be controlled may be broken into five groups. In this way, when each group of drives is active, the predefined 20% of drives will be active. The other 80% of drives may be in a sleep or other low-power mode. Alternatively, any other percentage may be used to calculate a corresponding number of groups of drives (e.g., 25% would make four groups, 10% would make ten groups, 50% would make two groups, etc.) as desired. Further, in some aspects, groups may not necessarily be equal in order to accommodate percentages that do not allow for equality among groups (e.g., 11% would make nine equal groups and one smaller group). Alternatively, percentages may be rounded up or down to accommodate equality among groups. Additionally, in some examples, an hour may be divided by the resulting number of groups to determine how long each group should be active. For example, using the 20% example for illustrative purposes only, each group may be scheduled to be active for 12 minutes to accommodate each group being active at least once per hour. Further, in order to avoid power spikes or burnouts, in some examples, drives within a group may be powered up sequentially or at least not all at the same

time. Alternatively, or in addition, each drive of a group that is active may be sequentially powered down while drives of the next group to be activated are sequentially powered up.

[0041] In other examples, rotation schedules may be based at least in part on a predefined number of drives to be active at a given time, as opposed to a percentage. Additionally, the rotation schedule may include a sliding window that moves at a speed or interval along a theoretical or real array of drives. Drives that are located within the window may be activated while drives outside the window may be in a low power state (which may include being off or deactivated). As desired, the drives may be sorted on a time line based at least in part on their actual location and/or based at least in part on some logical mapping. The window may be configured to traverse the timeline. Alternatively, other examples of activating drives within a sliding window may be envisioned. The rotation schedules may also be based at least in part on demand including, but not limited to, peak power consumption times (e.g., based at least in part on utility company fees and/or estimates), customer demand, pending jobs, etc. By way of example only, one scenario may include scheduling drives to be actively based at least in part on pending read requests from multiple storage node managers. In this example, the storage node registrar may be configured to select a number or percentage of drives that should come into a rotation in order to meet the demand. The storage node registrar may also send a message to the storage nodes, once the drives are selected to come into the rotation, that the drives have been scheduled for rotation. Alternatively, or in addition, the storage node managers may notify the storage nodes once the registrar has included the appropriate drives in the rotation.

[0042] FIG. 1 illustrates an example series of communications between a customer 102 and an archival data storage system 104, in accordance with at least one embodiment. The example communications illustrate a series of requests and responses that may be used, for example, to store and subsequently retrieve an archival data object to and from an archival data storage system. In various embodiments, the illustrated series of communications may occur in an environment 200 such as illustrated in FIG. 2, described below.

[0043] In various embodiments, archival data storage system 104 may provide a user interface for an entity such as a customer to communicate archival data storage system 104. In various embodiments, such a user interface may include graphical user interfaces (GUIs), Web-based

interfaces, programmatic interfaces such as application programming interfaces (APIs) and/or sets of remote procedure calls (RPCs) corresponding to interface elements, messaging interfaces in which the interface elements correspond to messages of a communication protocol, and/or suitable combinations thereof. For example, a customer may use such a user interface to store, retrieve or delete data as well as to obtain metadata information, configure various operational parameters and the like. In various embodiments, archival data storage system 104 may include a front end service such as described in FIG. 2 below, for handling customer requests.

[0044] In an embodiment, a customer device 102 (referred to also as simply a "customer") initiates a request 106 to store data, for example, using an API as describe above. The storage request 106 may include payload data such as an archival file and metadata such as a size and digest of the payload data, user identification information (e.g., customer account identifier), an identifier of a logical storage container (described in connection with FIG. 2) and the like. In some embodiments, multiple storage requests 106 may be used to request the upload of a large payload data, where each of the multiple storage requests may include a portion of the payload data. In other embodiments, a storage request 106 may include multiple data objects to be uploaded.

[0045] Upon receiving storage request 106, in an embodiment, archival data storage system 104 may store 108 the data in a staging store. In some embodiments, only payload data is stored in the staging storage. In other embodiments, additional data such as a data object identifier may be stored with payload data. In one embodiment, a job is created for moving the data stored in staging storage to archival data storage as described herein. As used herein, a "job" refers to a data-related activity corresponding to a customer request that may be performed temporally independently from the time the request is received. For example, a job may include retrieving, storing and deleting data, retrieving metadata and the like. In various embodiments, a job may be identified by a job identifier that may be unique, for example, among all the jobs for a particular customer, within a data center or the like. Such a job may be processed to store 114 the data object in archival data storage. In an embodiment, the job may be executed in connection with other jobs (such as other data storage jobs or retrieval jobs discussed below) to optimize costs, response time, efficiency, operational performance and other metrics of the

system. For example, jobs may be sorted by storage location, coalesced, batch-processed or otherwise optimized to improve throughput, reduce power consumption and the like. As another example, jobs may be partitioned (e.g., by customer, time, and the like) and each partition may be processed in parallel by a different instance of service.

[0046] In an embodiment, archival data storage system 104 sends a response 110 to customer 102 acknowledging the storage of data (regardless of whether the data is only stored in a staging store). Thus, from the customer's perspective, the storage request is handled in a synchronous fashion. In an embodiment, the response includes a data object identifier that may be used by subsequent customer requests to retrieve, delete or otherwise manage the data. In some embodiments, customer 102 may store 112 the data object identifier in a customer-side data store, a data storage service and the like. For example, customer 102 may maintain a map associating data object identifiers with corresponding user-friendly names or global unique identifiers (GUIDs). In other embodiments, storage requests may be perceived by a customer as being handled in an asynchronous fashion in a manner similar to that described below in connection with retrieval requests.

[0047] In some embodiments, data object identifiers may encode (e.g., via encryption) storage location information that may be used to locate the stored data object, payload validation information such as size, timestamp, digest and the like that may be used to validate the integrity of the payload data, metadata validation information such as error-detection codes that may be used to validate the integrity of metadata such as the data object identifier itself, policy information that may be used to validate the requested access and the like. In other embodiments, data object identifier may include the above information without any encoding such as encryption.

[0048] In an embodiment, a customer 102 initiates a request 116 to retrieve data stored in archival data storage system 104, for example, using an API described above. The retrieval request 116 may specify a data object identifier associated the data to be retrieved, such as provided in response 110, described above. Upon receiving retrieval request 116, in an embodiment, archival data storage system 104 creates 118 a retrieval job to retrieve data from an

archival data storage described herein. The system may create such a retrieval job in a manner similar to that described in connection of a storage job, above.

[0049] In an embodiment, archival data storage system 104 provides a response 120 with the job identifier for the retrieval job. Thus, from a customer's perspective, the retrieval request is handled in an asynchronous fashion. In some embodiments, customer 102 stores 122 the job identifier in a customer-side storage and may optionally use the job identifier query or poll archival data storage system for the status of the retrieval job.

[0050] In an embodiment, archival data storage system 104 processes 124 the retrieval job in connection with other jobs (e.g., storage jobs, retrieval jobs, deletion jobs and the like) using techniques such as batch processing to optimize costs. After the processing of the retrieval job, archival data storage system 104 stores 126 the retrieved data in a staging store similar to the staging store for storing data associated with storage requests, discussed above.

[0051] In an embodiment, archival data storage system 104 provides a notification 128 to customer 102 after the requested data is retrieved, for example, in a staging store. Such notifications may be configurable by customers and may include job identifier associated with the retrieval request, a data object identifier, a path to a download location, identity of the customer or any other information that is used, according to the embodiment being implemented, to download the retrieved data. In various embodiments, such notifications may be provided by a notification service that may or may not be part of the archival data storage system. In other embodiments, customer 102 may query or poll the archival data storage system with the retrieval job identifier regarding the status of the job.

[0052] In an embodiment, upon learning of the completion of the retrieval job, customer 102 sends a download request 130 to download the staged data. In various embodiments, download request 130 may provide information included in the notification such as job identifier, data object identifier and other identifying information such as customer account identifier and the like that would be used, in the embodiment being implemented, to download the staged data. In response to the download request 130, archival data storage system 104 may retrieve 132 the staged data from the staging store and provide the retrieved data to customer 102 in a response 134.

[0053] FIG. 2 illustrates an example environment 200 in which an archival data storage system may be implemented, in accordance with at least one embodiment. One or more customers 202 connect, via a network 204, to an archival data storage system 206. As implied above, unless otherwise clear from context, the term "customer" refers to the system(s) of a customer entity (such as an individual, company or other organization) that utilizes data storage services described herein. Such systems may include datacenters, mainframes, individual computing devices, distributed computing environments and customer-accessible instances thereof or any other system capable of communicating with the archival data storage system. In some embodiments, a customer may refer to a machine instance (e.g., with direct hardware access) or virtual instance of a distributed computing system provided by a computing resource provider that also provides the archival data storage system. In some embodiments, the archival data storage system is integral to the distributed computing system and may include or be implemented by an instance, virtual or machine, of the distributed computing system. In various embodiments, network 204 may include the Internet, a local area network ("LAN"), a wide area network ("WAN"), a cellular data network and/or other data network.

[0054] In an embodiment, archival data storage system 206 provides a multi-tenant or multi-customer environment where each tenant or customer may store, retrieve, delete or otherwise manage data in a data storage space allocated to the customer. In some embodiments, an archival data storage system 206 comprises multiple subsystems or "planes" that each provides a particular set of services or functionalities. For example, as illustrated in FIG. 2, archival data storage system 206 includes front end 208, control plane for direct I/O 210, common control plane 212, data plane 214 and metadata plane 216. Each subsystem or plane may comprise one or more components that collectively provide the particular set of functionalities. Each component may be implemented by one or more physical and/or logical computing devices, such as computers, data storage devices and the like. Components within each subsystem may communicate with components within the same subsystem, components in other subsystems or external entities such as customers. At least some of such interactions are indicated by arrows in FIG. 2. In particular, the main bulk data transfer paths in and out of archival data storage system 206 are denoted by bold arrows. It will be appreciated by those of ordinary skill in the art

that various embodiments may have fewer or a greater number of systems, subsystems and/or subcomponents than are illustrated in FIG. 2. Thus, the depiction of environment 200 in FIG. 2 should be taken as being illustrative in nature and not limiting to the scope of the disclosure.

[0055] In the illustrative embodiment, front end 208 implements a group of services that provides an interface between the archival data storage system 206 and external entities, such as one or more customers 202 described herein. In various embodiments, front end 208 provides an application programming interface ("API") to enable a user to programmatically interface with the various features, components and capabilities of the archival data storage system. Such APIs may be part of a user interface that may include graphical user interfaces (GUIs), Web-based interfaces, programmatic interfaces such as application programming interfaces (APIs) and/or sets of remote procedure calls (RPCs) corresponding to interface elements, messaging interfaces in which the interface elements correspond to messages of a communication protocol, and/or suitable combinations thereof.

[0056] Capabilities provided by archival data storage system 206 may include data storage, data retrieval, data deletion, metadata operations, configuration of various operational parameters and the like. Metadata operations may include requests to retrieve catalogs of data stored for a particular customer, data recovery requests, job inquires and the like. Configuration APIs may allow customers to configure account information, audit logs, policies, notifications settings and the like. A customer may request the performance of any of the above operations by sending API requests to the archival data storage system. Similarly, the archival data storage system may provide responses to customer requests. Such requests and responses may be submitted over any suitable communications protocol, such as Hypertext Transfer Protocol ("HTTP"), File Transfer Protocol ("FTP") and the like, in any suitable format, such as REpresentational State Transfer ("REST"), Simple Object Access Protocol ("SOAP") and the like. The requests and responses may be encoded, for example, using Base64 encoding, encrypted with a cryptographic key or the like.

[0057] In some embodiments, archival data storage system 206 allows customers to create one or more logical structures such as a logical data containers in which to store one or more archival data objects. As used herein, data object is used broadly and does not necessarily imply any

particular structure or relationship to other data. A data object may be, for instance, simply a sequence of bits. Typically, such logical data structures may be created to meeting certain business requirements of the customers and are independently of the physical organization of data stored in the archival data storage system. As used herein, the term "logical data container" refers to a grouping of data objects. For example, data objects created for a specific purpose or during a specific period of time may be stored in the same logical data container. Each logical data container may include nested data containers or data objects and may be associated with a set of policies such as size limit of the container, maximum number of data objects that may be stored in the container, expiration date, access control list and the like. In various embodiments, logical data containers may be created, deleted or otherwise modified by customers via API requests, by a system administrator or by the data storage system, for example, based on configurable information. For example, the following HTTP PUT request may be used, in an embodiment, to create a logical data container with name "logical-container-name" associated with a customer identified by an account identifier "accountId".

```
PUT /{accountId}/logical-container-name HTTP/1.1
```

[0058] In an embodiment, archival data storage system 206 provides the APIs for customers to store data objects into logical data containers. For example, the following HTTP POST request may be used, in an illustrative embodiment, to store a data object into a given logical container. In an embodiment, the request may specify the logical path of the storage location, data length, reference to the data payload, a digital digest of the data payload and other information. In one embodiment, the APIs may allow a customer to upload multiple data objects to one or more logical data containers in one request. In another embodiment where the data object is large, the APIs may allow a customer to upload the data object in multiple parts, each with a portion of the data object.

```
POST /{accountId}/logical-container-name/data HTTP/1.1
Content-Length: 1128192
x-ABC-data-description: "annual-result-2012.xls"
x-ABC-md5-tree-hash: 634d9a0688aff95c
```

[0059] In response to a data storage request, in an embodiment, archival data storage system 206 provides a data object identifier if the data object is stored successfully. Such data object identifier may be used to retrieve, delete or otherwise refer to the stored data object in subsequent requests. In some embodiments, such as data object identifier may be "self-describing" in that it includes (for example, with or without encryption) storage location information that may be used by the archival data storage system to locate the data object without the need for a additional data structures such as a global namespace key map. In addition, in some embodiments, data object identifiers may also encode other information such as payload digest, error-detection code, access control data and the other information that may be used to validate subsequent requests and data integrity. In some embodiments, the archival data storage system stores incoming data in a transient durable data store before moving it archival data storage. Thus, although customers may perceive that data is persisted durably at the moment when an upload request is completed, actual storage to a long-term persisted data store may not commence until sometime later (e.g., 12 hours later). In some embodiments, the timing of the actual storage may depend on the size of the data object, the system load during a diurnal cycle, configurable information such as a service-level agreement between a customer and a storage service provider and other factors.

[0060] In some embodiments, archival data storage system 206 provides the APIs for customers to retrieve data stored in the archival data storage system. In such embodiments, a customer may initiate a job to perform the data retrieval and may learn the completion of the job by a notification or by polling the system for the status of the job. As used herein, a "job" refers to a data-related activity corresponding to a customer request that may be performed temporally independently from the time the request is received. For example, a job may include retrieving, storing and deleting data, retrieving metadata and the like. A job may be identified by a job identifier that may be unique, for example, among all the jobs for a particular customer. For example, the following HTTP POST request may be used, in an illustrative embodiment, to initiate a job to retrieve a data object identified by a data object identifier "dataObjectId." In other embodiments, a data retrieval request may request the retrieval of multiple data objects, data objects associated with a logical data container and the like.

```
POST /{accountId}/logical-data-container-name/data/{dataObjectId}
HTTP/1.1
```

[0061] In response to the request, in an embodiment, archival data storage system 206 provides a job identifier job-id," that is assigned to the job in the following response. The response provides, in this example, a path to the storage location where the retrieved data will be stored.

```
HTTP/1.1 202 ACCEPTED
Location: /{accountId}/logical-data-container-name/jobs/{job-id}
```

[0062] At any given point in time, the archival data storage system may have many jobs pending for various data operations. In some embodiments, the archival data storage system may employ job planning and optimization techniques such as batch processing, load balancing, job coalescence and the like, to optimize system metrics such as cost, performance, scalability and the like. In some embodiments, the timing of the actual data retrieval depends on factors such as the size of the retrieved data, the system load and capacity, active status of storage devices and the like. For example, in some embodiments, at least some data storage devices in an archival data storage system may be activated or inactivated according to a power management schedule, for example, to reduce operational costs. Thus, retrieval of data stored in a currently active storage device (such as a rotating hard drive) may be faster than retrieval of data stored in a currently inactive storage device (such as a spinned-down hard drive).

[0063] In an embodiment, when a data retrieval job is completed, the retrieved data is stored in a staging data store and made available for customer download. In some embodiments, a customer is notified of the change in status of a job by a configurable notification service. In other embodiments, a customer may learn of the status of a job by polling the system using a job identifier. The following HTTP GET request may be used, in an embodiment, to download data that is retrieved by a job identified by "job-id," using a download path that has been previously provided.

```
GET /{accountId}/logical-data-container-name/jobs/{job-id}/output
HTTP/1.1
```

[0064] In response to the GET request, in an illustrative embodiment, archival data storage system 206 may provide the retrieved data in the following HTTP response, with a tree-hash of the data for verification purposes.

```
HTTP/1.1 200 OK
Content-Length: 1128192
x-ABC-archive-description: "retrieved stuff"
x-ABC-md5-tree-hash: 693d9a7838aff95c

[1112192 bytes of user data follows]
```

[0065] In an embodiment, a customer may request the deletion of a data object stored in an archival data storage system by specifying a data object identifier associated with the data object. For example, in an illustrative embodiment, a data object with data object identifier "dataObjectId" may be deleted using the following HTTP request. In another embodiment, a customer may request the deletion of multiple data objects such as those associated with a particular logical data container.

```
DELETE /{accountId}/logical-data-container-name/data/{dataObjectId}
HTTP/1.1
```

[0066] In various embodiments, data objects may be deleted in response to a customer request or may be deleted automatically according to a user-specified or default expiration date. In some embodiments, data objects may be rendered inaccessible to customers upon an expiration time but remain recoverable during a grace period beyond the expiration time. In various embodiments, the grace period may be based on configurable information such as customer configuration, service-level agreement terms and the like. In some embodiments, a customer may be provided the abilities to query or receive notifications for pending data deletions and/or cancel one or more of the pending data deletions. For example, in one embodiment, a customer may set up notification configurations associated with a logical data container such that the customer will receive notifications of certain events pertinent to the logical data container. Such events may include the completion of a data retrieval job request, the completion of metadata request, deletion of data objects or logical data containers and the like.

[0067] In an embodiment, archival data storage system 206 also provides metadata APIs for retrieving and managing metadata such as metadata associated with logical data containers. In various embodiments, such requests may be handled asynchronously (where results are returned later) or synchronously (where results are returned immediately).

[0068] Still referring to FIG. 2, in an embodiment, at least some of the API requests discussed above are handled by API request handler 218 as part of front end 208. For example, API request handler 218 may decode and/or parse an incoming API request to extract information, such as uniform resource identifier ("URI"), requested action and associated parameters, identity information, data object identifiers and the like. In addition, API request handler 218 invoke other services (described below), where necessary, to further process the API request.

[0069] In an embodiment, front end 208 includes an authentication service 220 that may be invoked, for example, by API handler 218, to authenticate an API request. For example, in some embodiments, authentication service 220 may verify identity information submitted with the API request such as username and password Internet Protocol ("IP) address, cookies, digital certificate, digital signature and the like. In other embodiments, authentication service 220 may require the customer to provide additional information or perform additional steps to authenticate the request, such as required in a multifactor authentication scheme, under a challenge-response authentication protocol and the like.

[0070] In an embodiment, front end 208 includes an authorization service 222 that may be invoked, for example, by API handler 218, to determine whether a requested access is permitted according to one or more policies determined to be relevant to the request. For example, in one embodiment, authorization service 222 verifies that a requested access is directed to data objects contained in the requestor's own logical data containers or which the requester is otherwise authorized to access. In some embodiments, authorization service 222 or other services of front end 208 may check the validity and integrity of a data request based at least in part on information encoded in the request, such as validation information encoded by a data object identifier.

[0071] In an embodiment, front end 208 includes a metering service 224 that monitors service usage information for each customer such as data storage space used, number of data objects

stored, data requests processed and the like. In an embodiment, front end 208 also includes accounting service 226 that performs accounting and billing-related functionalities based, for example, on the metering information collected by the metering service 224, customer account information and the like. For example, a customer may be charged a fee based on the storage space used by the customer, size and number of the data objects, types and number of requests submitted, customer account type, service level agreement the like.

[0072] In an embodiment, front end 208 batch processes some or all incoming requests. For example, front end 208 may wait until a certain number of requests has been received before processing (e.g., authentication, authorization, accounting and the like) the requests. Such a batch processing of incoming requests may be used to gain efficiency.

[0073] In some embodiments, front end 208 may invoke services provided by other subsystems of the archival data storage system to further process an API request. For example, front end 208 may invoke services in metadata plane 216 to fulfill metadata requests. For another example, front end 208 may stream data in and out of control plane for direct I/O 210 for data storage and retrieval requests, respectively.

[0074] Referring now to control plane for direct I/O 210 illustrated in FIG. 2, in various embodiments, control plane for direct I/O 210 provides services that create, track and manage jobs created as a result of customer requests. As discussed above, a job refers to a customer-initiated activity that may be performed asynchronously to the initiating request, such as data retrieval, storage, metadata queries or the like. In an embodiment, control plane for direct I/O 210 includes a job tracker 230 that is configured to create job records or entries corresponding to customer requests, such as those received from API request handler 218, and monitor the execution of the jobs. In various embodiments, a job record may include information related to the execution of a job such as a customer account identifier, job identifier, data object identifier, reference to payload data cache 228 (described below), job status, data validation information and the like. In some embodiments, job tracker 230 may collect information necessary to construct a job record from multiple requests. For example, when a large amount of data is requested to be stored, data upload may be broken into multiple requests, each uploading a portion of the data. In such a case, job tracker 230 may maintain information to

keep track of the upload status to ensure that all data parts have been received before a job record is created. In some embodiments, job tracker 230 also obtains a data object identifier associated with the data to be stored and provides the data object identifier, for example, to a front end service to be returned to a customer. In an embodiment, such data object identifier may be obtained from data plane 214 services such as storage node manager 244, storage node registrar 248, and the like, described below.

[0075] In some embodiments, control plane for direct I/O 210 includes a job tracker store 232 for storing job entries or records. In various embodiments, job tracker store 230 may be implemented by a NoSQL data management system, such as a key-value data store, a relational database management system ("RDBMS") or any other data storage system. In some embodiments, data stored in job tracker store 230 may be partitioned to enable fast enumeration of jobs that belong to a specific customer, facilitate efficient bulk record deletion, parallel processing by separate instances of a service and the like. For example, job tracker store 230 may implement tables that are partitioned according to customer account identifiers and that use job identifiers as range keys. In an embodiment, job tracker store 230 is further sub-partitioned based on time (such as job expiration time) to facilitate job expiration and cleanup operations. In an embodiment, transactions against job tracker store 232 may be aggregated to reduce the total number of transactions. For example, in some embodiments, a job tracker 230 may perform aggregate multiple jobs corresponding to multiple requests into one single aggregated job before inserting it into job tracker store 232.

[0076] In an embodiment, job tracker 230 is configured to submit the job for further job scheduling and planning, for example, by services in common control plane 212. Additionally, job tracker 230 may be configured to monitor the execution of jobs and update corresponding job records in job tracker store 232 as jobs are completed. In some embodiments, job tracker 230 may be further configured to handle customer queries such as job status queries. In some embodiments, job tracker 230 also provides notifications of job status changes to customers or other services of the archival data storage system. For example, when a data retrieval job is completed, job tracker 230 may cause a customer to be notified (for example, using a notification service) that data is available for download. As another example, when a data storage job is

completed, job tracker 230 may notify a cleanup agent 234 to remove payload data associated with the data storage job from a transient payload data cache 228, described below.

[0077] In an embodiment, control plane for direct I/O 210 includes a payload data cache 228 for providing transient data storage services for payload data transiting between data plane 214 and front end 208. Such data includes incoming data pending storage and outgoing data pending customer download. As used herein, transient data store is used interchangeably with temporary or staging data store to refer to a data store that is used to store data objects before they are stored in an archival data storage described herein or to store data objects that are retrieved from the archival data storage. A transient data store may provide volatile or non-volatile (durable) storage. In most embodiments, while potentially usable for persistently storing data, a transient data store is intended to store data for a shorter period of time than an archival data storage system and may be less cost-effective than the data archival storage system described herein. In one embodiment, transient data storage services provided for incoming and outgoing data may be differentiated. For example, data storage for the incoming data, which is not yet persisted in archival data storage, may provide higher reliability and durability than data storage for outgoing (retrieved) data, which is already persisted in archival data storage. In another embodiment, transient storage may be optional for incoming data, that is, incoming data may be stored directly in archival data storage without being stored in transient data storage such as payload data cache 228, for example, when there is the system has sufficient bandwidth and/or capacity to do so.

[0078] In an embodiment, control plane for direct I/O 210 also includes a cleanup agent 234 that monitors job tracker store 232 and/or payload data cache 228 and removes data that is no longer needed. For example, payload data associated with a data storage request may be safely removed from payload data cache 228 after the data is persisted in permanent storage (e.g., data plane 214). On the reverse path, data staged for customer download may be removed from payload data cache 228 after a configurable period of time (e.g., 30 days since the data is staged) or after a customer indicates that the staged data is no longer needed.

[0079] In some embodiments, cleanup agent 234 removes a job record from job tracker store 232 when the job status indicates that the job is complete or aborted. As discussed above,

in some embodiments, job tracker store 232 may be partitioned to enable to enable faster cleanup. In one embodiment where data is partitioned by customer account identifiers, cleanup agent 234 may remove an entire table that stores jobs for a particular customer account when the jobs are completed instead of deleting individual jobs one at a time. In another embodiment where data is further sub-partitioned based on job expiration time cleanup agent 234 may bulk-delete a whole partition or table of jobs after all the jobs in the partition expire. In other embodiments, cleanup agent 234 may receive instructions or control messages (such as indication that jobs are completed) from other services such as job tracker 230 that cause the cleanup agent 234 to remove job records from job tracker store 232 and/or payload data cache 228.

[0080] Referring now to common control plane 212 illustrated in FIG. 2. In various embodiments, common control plane 212 provides a queue-based load leveling service to dampen peak to average load levels (jobs) coming from control plane for I/O 210 and to deliver manageable workload to data plane 214. In an embodiment, common control plane 212 includes a job request queue 236 for receiving jobs created by job tracker 230 in control plane for direct I/O 210, described above, a storage node manager job store 240 from which services from data plane 214 (e.g., storage node managers 244) pick up work to execute and a request balancer 238 for transferring job items from job request queue 236 to storage node manager job store 240 in an intelligent manner.

[0081] In an embodiment, job request queue 236 provides a service for inserting items into and removing items from a queue (e.g., first-in-first-out (FIFO) or first-in-last-out (FILO)), a set or any other suitable data structure. Job entries in the job request queue 236 may be similar to or different from job records stored in job tracker store 232, described above.

[0082] In an embodiment, common control plane 212 also provides a durable high efficiency job store, storage node manager job store 240, that allows services from data plane 214 (e.g., storage node manager 244, anti-entropy watcher 252) to perform job planning optimization, check pointing and recovery. For example, in an embodiment, storage node manager job store 240 allows the job optimization such as batch processing, operation coalescing and the like by supporting scanning, querying, sorting or otherwise manipulating and managing job items

stored in storage node manager job store 240. In an embodiment, a storage node manager 244 scans incoming jobs and sort the jobs by the type of data operation (e.g., read, write or delete), storage locations (e.g., volume, disk), customer account identifier and the like. The storage node manager 244 may then reorder, coalesce, group in batches or otherwise manipulate and schedule the jobs for processing. For example, in one embodiment, the storage node manager 244 may batch process all the write operations before all the read and delete operations. In another embodiment, the storage node manager 224 may perform operation coalescing. For another example, the storage node manager 224 may coalesce multiple retrieval jobs for the same object into one job or cancel a storage job and a deletion job for the same data object where the deletion job comes after the storage job.

[0083] In an embodiment, storage node manager job store 240 is partitioned, for example, based on job identifiers, so as to allow independent processing of multiple storage node managers 244 and to provide even distribution of the incoming workload to all participating storage node managers 244. In various embodiments, storage node manager job store 240 may be implemented by a NoSQL data management system, such as a key-value data store, a RDBMS or any other data storage system.

[0084] In an embodiment, request balancer 238 provides a service for transferring job items from job request queue 236 to storage node manager job store 240 so as to smooth out variation in workload and to increase system availability. For example, request balancer 238 may transfer job items from job request queue 236 at a lower rate or at a smaller granularity when there is a surge in job requests coming into the job request queue 236 and vice versa when there is a lull in incoming job requests so as to maintain a relatively sustainable level of workload in the storage node manager store 240. In some embodiments, such sustainable level of workload is around the same or below the average workload of the system.

[0085] In an embodiment, job items that are completed are removed from storage node manager job store 240 and added to the job result queue 242. In an embodiment, data plane 214 services (e.g., storage node manager 244) are responsible for removing the job items from the storage node manager job store 240 and adding them to job result queue 242. In some

embodiments, job request queue 242 is implemented in a similar manner as job request queue 235, discussed above.

[0086] Referring now to data plane 214 illustrated in FIG. 2. In various embodiments, data plane 214 provides services related to long-term archival data storage, retrieval and deletion, data management and placement, anti-entropy operations and the like. In various embodiments, data plane 214 may include any number and type of storage entities such as data storage devices (such as tape drives, hard disk drives, solid state devices, and the like), storage nodes or servers, datacenters and the like. Such storage entities may be physical, virtual or any abstraction thereof (e.g., instances of distributed storage and/or computing systems) and may be organized into any topology, including hierarchical or tiered topologies. Similarly, the components of the data plane may be dispersed, local or any combination thereof. For example, various computing or storage components may be local or remote to any number of datacenters, servers or data storage devices, which in turn may be local or remote relative to one another. In various embodiments, physical storage entities may be designed for minimizing power and cooling costs by controlling the portions of physical hardware that are active (e.g., the number of hard drives that are actively rotating). In an embodiment, physical storage entities implement techniques, such as Shingled Magnetic Recording (SMR), to increase storage capacity.

[0087] In an environment illustrated by FIG. 2, one or more storage node managers 244 each controls one or more storage nodes 246 by sending and receiving data and control messages. Each storage node 246 in turn controls a (potentially large) collection of data storage devices such as hard disk drives. In various embodiments, a storage node manager 244 may communicate with one or more storage nodes 246 and a storage node 246 may communicate with one or more storage node managers 244. In an embodiment, storage node managers 244 are implemented by one or more computing devices that are capable of performing relatively complex computations such as digest computation, data encoding and decoding, job planning and optimization and the like. In some embodiments, storage nodes 244 are implemented by one or more computing devices with less powerful computation capabilities than storage node managers 244. Further, in some embodiments the storage node manager 244 may not be included in the data path. For example, data may be transmitted from the payload data cache 228

directly to the storage nodes 246 or from one or more storage nodes 246 to the payload data cache 228. In this way, the storage node manager 244 may transmit instructions to the payload data cache 228 and/or the storage nodes 246 without receiving the payloads directly from the payload data cache 228 and/or storage nodes 246. In various embodiments, a storage node manager 244 may send instructions or control messages to any other components of the archival data storage system 206 described herein to direct the flow of data.

[0088] In an embodiment, a storage node manager 244 serves as an entry point for jobs coming into and out of data plane 214 by picking job items from common control plane 212 (e.g., storage node manager job store 240), retrieving staged data from payload data cache 228 and performing necessary data encoding for data storage jobs and requesting appropriate storage nodes 246 to store, retrieve or delete data. Once the storage nodes 246 finish performing the requested data operations, the storage node manager 244 may perform additional processing, such as data decoding and storing retrieved data in payload data cache 228 for data retrieval jobs, and update job records in common control plane 212 (e.g., removing finished jobs from storage node manager job store 240 and adding them to job result queue 242).

[0089] In an embodiment, storage node manager 244 performs data encoding according to one or more data encoding schemes before data storage to provide data redundancy, security and the like. Such data encoding schemes may include encryption schemes, redundancy encoding schemes such as erasure encoding, redundant array of independent disks (RAID) encoding schemes, replication and the like. Likewise, in an embodiment, storage node managers 244 performs corresponding data decoding schemes, such as decryption, erasure-decoding and the like, after data retrieval to restore the original data.

[0090] As discussed above in connection with storage node manager job store 240, storage node managers 244 may implement job planning and optimizations such as batch processing, operation coalescing and the like to increase efficiency. In some embodiments, jobs are partitioned among storage node managers so that there is little or no overlap between the partitions. Such embodiments facilitate parallel processing by multiple storage node managers, for example, by reducing the probability of racing or locking.

[0091] In various embodiments, data plane 214 is implemented to facilitate data integrity. For example, storage entities handling bulk data flows such as storage nodes managers 244 and/or storage nodes 246 may validate the digest of data stored or retrieved, check the error-detection code to ensure integrity of metadata and the like.

[0092] In various embodiments, data plane 214 is implemented to facilitate scalability and reliability of the archival data storage system. For example, in one embodiment, storage node managers 244 maintain no or little internal state so that they can be added, removed or replaced with little adverse impact. In one embodiment, each storage device is a self-contained and self-describing storage unit capable of providing information about data stored thereon. Such information may be used to facilitate data recovery in case of data loss. Furthermore, in one embodiment, each storage node 246 is capable of collecting and reporting information about the storage node including the network location of the storage node and storage information of connected storage devices to one or more storage node registrars 248 and/or storage node registrar stores 250. In some embodiments, storage nodes 246 perform such self-reporting at system start up time and periodically provide updated information. In various embodiments, such a self-reporting approach provides dynamic and up-to-date directory information without the need to maintain a global namespace key map or index which can grow substantially as large amounts of data objects are stored in the archival data system.

[0093] In an embodiment, data plane 214 may also include one or more storage node registrars 248 that provide directory information for storage entities and data stored thereon, data placement services and the like. Storage node registrars 248 may communicate with and act as a front end service to one or more storage node registrar stores 250, which provide storage for the storage node registrars 248. In various embodiments, storage node registrar store 250 may be implemented by a NoSQL data management system, such as a key-value data store, a RDBMS or any other data storage system. In some embodiments, storage node registrar stores 250 may be partitioned to enable parallel processing by multiple instances of services. As discussed above, in an embodiment, information stored at storage node registrar store 250 is based at least partially on information reported by storage nodes 246 themselves.

[0094] In some embodiments, storage node registrars 248 provide directory service, for example, to storage node managers 244 that want to determine which storage nodes 246 to contact for data storage, retrieval and deletion operations. For example, given a volume identifier provided by a storage node manager 244, storage node registrars 248 may provide, based on a mapping maintained in a storage node registrar store 250, a list of storage nodes that host volume components corresponding to the volume identifier. Specifically, in one embodiment, storage node registrar store 250 stores a mapping between a list of identifiers of volumes or volume components and endpoints, such as Domain Name System (DNS) names, of storage nodes that host the volumes or volume components.

[0095] As used herein, a "volume" refers to a logical storage space within a data storage system in which data objects may be stored. A volume may be identified by a volume identifier. A volume may reside in one physical storage device (e.g., a hard disk) or span across multiple storage devices. In the latter case, a volume comprises a plurality of volume components each residing on a different storage device. As used herein, a "volume component" refers a portion of a volume that is physically stored in a storage entity such as a storage device. Volume components for the same volume may be stored on different storage entities. In one embodiment, when data is encoded by a redundancy encoding scheme (e.g., erasure coding scheme, RAID, replication), each encoded data component or "shard" may be stored in a different volume component to provide fault tolerance and isolation. In some embodiments, a volume component is identified by a volume component identifier that includes a volume identifier and a shard slot identifier. As used herein, a shard slot identifies a particular shard, row or stripe of data in a redundancy encoding scheme. For example, in one embodiment, a shard slot corresponds to an erasure coding matrix row. In some embodiments, storage node registrar store 250 also stores information about volumes or volume components such as total, used and free space, number of data objects stored and the like.

[0096] In some embodiments, data plane 214 also includes a storage allocator 256 for allocating storage space (e.g., volumes) on storage nodes to store new data objects, based at least in part on information maintained by storage node registrar store 250, to satisfy data isolation

and fault tolerance constraints. In some embodiments, storage allocator 256 requires manual intervention.

[0097] In some embodiments, data plane 214 also includes an anti-entropy watcher 252 for detecting entropic effects and initiating anti-entropy correction routines. For example, anti-entropy watcher 252 may be responsible for monitoring activities and status of all storage entities such as storage nodes, reconciling live or actual data with maintained data and the like. In various embodiments, entropic effects include, but are not limited to, performance degradation due to data fragmentation resulting from repeated write and rewrite cycles, hardware wear (e.g., of magnetic media), data unavailability and/or data loss due to hardware/software malfunction, environmental factors, physical destruction of hardware, random chance or other causes. Anti-entropy watcher 252 may detect such effects and in some embodiments may preemptively and/or reactively institute anti-entropy correction routines and/or policies.

[0098] In an embodiment, anti-entropy watcher 252 causes storage nodes 246 to perform periodic anti-entropy scans on storage devices connected to the storage nodes. Anti-entropy watcher 252 may also inject requests in job request queue 236 (and subsequently job result queue 242) to collect information, recover data and the like. In some embodiments, anti-entropy watcher 252 may perform scans, for example, on cold index store 262, described below, and storage nodes 246, to ensure referential integrity.

[0099] In an embodiment, information stored at storage node registrar store 250 is used by a variety of services such as storage node registrar 248, storage allocator 256, anti-entropy watcher 252 and the like. For example, storage node registrar 248 may provide data location and placement services (e.g., to storage node managers 244) during data storage, retrieval and deletion. For example, given the size of a data object to be stored and information maintained by storage node registrar store 250, a storage node registrar 248 may determine where (e.g., volume) to store the data object and provides an indication of the storage location of the data object which may be used to generate a data object identifier associated with the data object. As another example, in an embodiment, storage allocator 256 uses information stored in storage node registrar store 250 to create and place volume components for new volumes in specific storage nodes to satisfy isolation and fault tolerance constraints. As yet another example, in an

embodiment, anti-entropy watcher 252 uses information stored in storage node registrar store 250 to detect entropic effects such as data loss, hardware failure and the like.

[0100] In some embodiments, data plane 214 also includes an orphan cleanup data store 254, which is used to track orphans in the storage system. As used herein, an orphan is a stored data object that is not referenced by any external entity. In various embodiments, orphan cleanup data store 254 may be implemented by a NoSQL data management system, such as a key-value data store, an RDBMS or any other data storage system. In some embodiments, storage node registrars 248 stores object placement information in orphan cleanup data store 254. Subsequently, information stored in orphan cleanup data store 254 may be compared, for example, by an anti-entropy watcher 252, with information maintained in metadata plane 216. If an orphan is detected, in some embodiments, a request is inserted in the common control plane 212 to delete the orphan.

[0101] Referring now to metadata plane 216 illustrated in FIG. 2. In various embodiments, metadata plane 216 provides information about data objects stored in the system for inventory and accounting purposes, to satisfy customer metadata inquiries and the like. In the illustrated embodiment, metadata plane 216 includes a metadata manager job store 258 which stores information about executed transactions based on entries from job result queue 242 in common control plane 212. In various embodiments, metadata manager job store 258 may be implemented by a NoSQL data management system, such as a key-value data store, a RDBMS or any other data storage system. In some embodiments, metadata manager job store 258 is partitioned and sub-partitioned, for example, based on logical data containers, to facilitate parallel processing by multiple instances of services such as metadata manager 260.

[0102] In the illustrative embodiment, metadata plane 216 also includes one or more metadata managers 260 for generating a cold index of data objects (e.g., stored in cold index store 262) based on records in metadata manager job store 258. As used herein, a "cold" index refers to an index that is updated infrequently. In various embodiments, a cold index is maintained to reduce cost overhead. In some embodiments, multiple metadata managers 260 may periodically read and process records from different partitions in metadata manager job store 258 in parallel and store the result in a cold index store 262.

[0103] In some embodiments cold index store 262 may be implemented by a reliable and durable data storage service. In some embodiments, cold index store 262 is configured to handle metadata requests initiated by customers. For example, a customer may issue a request to list all data objects contained in a given logical data container. In response to such a request, cold index store 262 may provide a list of identifiers of all data objects contained in the logical data container based on information maintained by cold index 262. In some embodiments, an operation may take a relative long period of time and the customer may be provided a job identifier to retrieve the result when the job is done. In other embodiments, cold index store 262 is configured to handle inquiries from other services, for example, from front end 208 for inventory, accounting and billing purposes.

[0104] In some embodiments, metadata plane 216 may also include a container metadata store 264 that stores information about logical data containers such as container ownership, policies, usage and the like. Such information may be used, for example, by front end 208 services, to perform authorization, metering, accounting and the like. In various embodiments, container metadata store 264 may be implemented by a NoSQL data management system, such as a key-value data store, a RDBMS or any other data storage system.

[0105] As described herein, in various embodiments, the archival data storage system 206 described herein is implemented to be efficient and scalable. For example, in an embodiment, batch processing and request coalescing is used at various stages (e.g., front end request handling, control plane job request handling, data plane data request handling) to improve efficiency. For another example, in an embodiment, processing of metadata such as jobs, requests and the like are partitioned so as to facilitate parallel processing of the partitions by multiple instances of services.

[0106] In an embodiment, data elements stored in the archival data storage system (such as data components, volumes, described below) are self-describing so as to avoid the need for a global index data structure. For example, in an embodiment, data objects stored in the system may be addressable by data object identifiers that encode storage location information. For another example, in an embodiment, volumes may store information about which data objects are stored in the volume and storage nodes and devices storing such volumes may collectively

report their inventory and hardware information to provide a global view of the data stored in the system (such as evidenced by information stored in storage node registrar store 250). In such an embodiment, the global view is provided for efficiency only and not required to locate data stored in the system.

[0107] In various embodiments, the archival data storage system described herein is implemented to improve data reliability and durability. For example, in an embodiment, a data object is redundantly encoded into a plurality of data components and stored across different data storage entities to provide fault tolerance. For another example, in an embodiment, data elements have multiple levels of integrity checks. In an embodiment, parent/child relations always have additional information to ensure full referential integrity. For example, in an embodiment, bulk data transmission and storage paths are protected by having the initiator pre-calculate the digest on the data before transmission and subsequently supply the digest with the data to a receiver. The receiver of the data transmission is responsible for recalculation, comparing and then acknowledging to the sender that includes the recalculated the digest. Such data integrity checks may be implemented, for example, by front end services, transient data storage services, data plane storage entities and the like described above.

[0108] FIG. 3 illustrates an interconnection network 300 in which components of an archival data storage system may be connected, in accordance with at least one embodiment. In particular, the illustrated example shows how data plane components are connected to the interconnection network 300. In some embodiments, the interconnection network 300 may include a fat tree interconnection network where the link bandwidth grows higher or "fatter" towards the root of the tree. In the illustrated example, data plane includes one or more datacenters 301. Each datacenter 301 may include one or more storage node manager server racks 302 where each server rack hosts one or more servers that collectively provide the functionality of a storage node manager such as described in connection with FIG. 2. In other embodiments, each storage node manager server rack may host more than one storage node manager. Configuration parameters such as number of storage node managers per rack, number of storage node manager racks and the like may be determined based on factors such as cost,

scalability, redundancy and performance requirements, hardware and software resources and the like.

[0109] Each storage node manager server rack 302 may have a storage node manager rack connection 314 to an interconnect 308 used to connect to the interconnection network 300. In some embodiments, the connection 314 is implemented using a network switch 303 that may include a top-of-rack Ethernet switch or any other type of network switch. In various embodiments, interconnect 308 is used to enable high-bandwidth and low-latency bulk data transfers. For example, interconnect may include a Clos network, a fat tree interconnect, an Asynchronous Transfer Mode (ATM) network, a Fast or Gigabit Ethernet and the like.

[0110] In various embodiments, the bandwidth of storage node manager rack connection 314 may be configured to enable high-bandwidth and low-latency communications between storage node managers and storage nodes located within the same or different data centers. For example, in an embodiment, the storage node manager rack connection 314 has a bandwidth of 10 Gigabit per second (Gbps).

[0111] In some embodiments, each datacenter 301 may also include one or more storage node server racks 304 where each server rack hosts one or more servers that collectively provide the functionalities of a number of storage nodes such as described in connection with FIG. 2. Configuration parameters such as number of storage nodes per rack, number of storage node racks, ration between storage node managers and storage nodes and the like may be determined based on factors such as cost, scalability, redundancy and performance requirements, hardware and software resources and the like. For example, in one embodiment, there are 3 storage nodes per storage node server rack, 30-80 racks per data center and a storage nodes / storage node manager ratio of 10 to 1.

[0112] Each storage node server rack 304 may have a storage node rack connection 316 to an interconnection network switch 308 used to connect to the interconnection network 300. In some embodiments, the connection 316 is implemented using a network switch 305 that may include a top-of-rack Ethernet switch or any other type of network switch. In various embodiments, the bandwidth of storage node rack connection 316 may be configured to enable high-bandwidth and low-latency communications between storage node managers and storage

nodes located within the same or different data centers. In some embodiments, a storage node rack connection 316 has a higher bandwidth than a storage node manager rack connection 314. For example, in an embodiment, the storage node rack connection 316 has a bandwidth of 20 Gbps while a storage node manager rack connection 314 has a bandwidth of 10 Gbps.

[0113] In some embodiments, datacenters 301 (including storage node managers and storage nodes) communicate, via connection 310, with other computing resources services 306 such as payload data cache 228, storage node manager job store 240, storage node registrar 248, storage node registrar store 350, orphan cleanup data store 254, metadata manager job store 258 and the like as described in connection with FIG. 2.

[0114] In some embodiments, one or more datacenters 301 may be connected via inter-datacenter connection 312. In some embodiments, connections 310 and 312 may be configured to achieve effective operations and use of hardware resources. For example, in an embodiment, connection 310 has a bandwidth of 30-100 Gbps per datacenter and inter-datacenter connection 312 has a bandwidth of 100-250 Gbps.

[0115] FIG. 4 illustrates an interconnection network 400 in which components of an archival data storage system may be connected, in accordance with at least one embodiment. In particular, the illustrated example shows how non-data plane components are connected to the interconnection network 300. As illustrated, front end services, such as described in connection with FIG. 2, may be hosted by one or more front end server racks 402. For example, each front end server rack 402 may host one or more web servers. The front end server racks 402 may be connected to the interconnection network 400 via a network switch 408. In one embodiment, configuration parameters such as number of front end services, number of services per rack, bandwidth for front end server rack connection 314 and the like may roughly correspond to those for storage node managers as described in connection with FIG. 3.

[0116] In some embodiments, control plane services and metadata plane services as described in connection with FIG. 2 may be hosted by one or more server racks 404. Such services may include job tracker 230, metadata manager 260, cleanup agent 232, job request balancer 238 and other services. In some embodiments, such services include services that do not handle frequent bulk data transfers. Finally, components described herein may communicate via connection 410,

with other computing resources services 406 such as payload data cache 228, job tracker store 232, metadata manager job store 258 and the like as described in connection with FIG. 2.

[0117] FIG. 5 illustrates an example process 500 for storing data, in accordance with at least one embodiment. Some or all of process 500 (or any other processes described herein or variations and/or combinations thereof) may be performed under the control of one or more computer systems configured with executable instructions and may be implemented as code (e.g., executable instructions, one or more computer programs or one or more applications) executing collectively on one or more processors, by hardware or combinations thereof. The code may be stored on a computer-readable storage medium, for example, in the form of a computer program comprising a plurality of instructions executable by one or more processors. The computer-readable storage medium may be non-transitory. In an embodiment, one or more components of archival data storage system 206 as described in connection with FIG. 2 may perform process 500.

[0118] In an embodiment, process 500 includes receiving 502 a data storage request to store archival data such as a document, a video or audio file or the like. Such a data storage request may include payload data and metadata such as size and digest of the payload data, user identification information (e.g., user name, account identifier and the like), a logical data container identifier and the like. In some embodiments, process 500 may include receiving 502 multiple storage requests each including a portion of larger payload data. In other embodiments, a storage request may include multiple data objects to be uploaded. In an embodiment, step 502 of process 500 is implemented by a service such as API request handler 218 of front end 208 as described in connection with FIG. 2.

[0119] In an embodiment, process 500 includes processing 504 the storage request upon receiving 502 the request. Such processing may include, for example, verifying the integrity of data received, authenticating the customer, authorizing requested access against access control policies, performing meter- and accounting-related activities and the like. In an embodiment, such processing may be performed by services of front end 208 such as described in connection with FIG. 2. In an embodiment, such a request may be processed in connection with other requests, for example, in batch mode.

[0120] In an embodiment, process 500 includes storing 506 the data associated with the storage request in a staging data store. Such staging data store may include a transient data store such as provided by payload data cache 228 as described in connection with FIG. 2. In some embodiments, only payload data is stored in the staging store. In other embodiments, metadata related to the payload data may also be stored in the staging store. In an embodiment, data integrity is validated (e.g., based on a digest) before being stored at a staging data store.

[0121] In an embodiment, process 500 includes providing 508 a data object identifier associated with the data to be stored, for example, in a response to the storage request. As described above, a data object identifier may be used by subsequent requests to retrieve, delete or otherwise reference data stored. In an embodiment, a data object identifier may encode storage location information that may be used to locate the stored data object, payload validation information such as size, digest, timestamp and the like that may be used to validate the integrity of the payload data, metadata validation information such as error-detection codes that may be used to validate the integrity of metadata such as the data object identifier itself and information encoded in the data object identifier and the like. In an embodiment, a data object identifier may also encode information used to validate or authorize subsequent customer requests. For example, a data object identifier may encode the identifier of the logical data container that the data object is stored in. In a subsequent request to retrieve this data object, the logical data container identifier may be used to determine whether the requesting entity has access to the logical data container and hence the data objects contained therein. In some embodiments, the data object identifier may encode information based on information supplied by a customer (e.g., a global unique identifier, GUID, for the data object and the like) and/or information collected or calculated by the system performing process 500 (e.g., storage location information). In some embodiments, generating a data object identifier may include encrypting some or all of the information described above using a cryptographic private key. In some embodiments, the cryptographic private key may be periodically rotated. In some embodiments, a data object identifier may be generated and/or provided at a different time than described above. For example, a data object identifier may be generated and/or provided after a storage job (described below) is created and/or completed.

[0122] In an embodiment, providing 508 a data object identifier may include determining a storage location for the before the data is actually stored there. For example, such determination may be based at least in part on inventory information about existing data storage entities such as operational status (e.g., active or inactive), available storage space, data isolation requirement and the like. In an environment such as environment 200 illustrated by FIG. 2, such determination may be implemented by a service such as storage node registrar 248 as described above in connection with FIG. 2. In some embodiments, such determination may include allocating new storage space (e.g., volume) on one or more physical storage devices by a service such as storage allocator 256 as described in connection with FIG. 2.

[0123] In an embodiment, a storage location identifier may be generated to represent the storage location determined above. Such a storage location identifier may include, for example, a volume reference object which comprises a volume identifier component and data object identifier component. The volume reference component may identify the volume the data is stored on and the data object identifier component may identify where in the volume the data is stored. In general, the storage location identifier may comprise components that identify various levels within a logical or physical data storage topology (such as a hierarchy) in which data is organized. In some embodiments, the storage location identifier may point to where actual payload data is stored or a chain of reference to where the data is stored.

[0124] In an embodiments, a data object identifier encodes a digest (e.g., a hash) of at least a portion of the data to be stored, such as the payload data. In some embodiments, the digest may be based at least in part on a customer-provided digest. In other embodiments, the digest may be calculated from scratch based on the payload data.

[0125] In an embodiment, process 500 includes creating 510 a storage job for persisting data to a long-term data store and scheduling 512 the storage job for execution. In environment 200 as described in connection with FIG. 2, steps 508, 510 and 512 may be implemented at least in part by components of control plane for direct I/O 210 and common control plane 212 as described above. Specifically, in an embodiment, job tracker 230 creates a job record and stores the job record in job tracker store 232. As described above, job tracker 230 may perform batch processing to reduce the total number of transactions against job tracker store 232. Additionally,

job tracker store 232 may be partitioned or otherwise optimized to facilitate parallel processing, cleanup operations and the like. A job record, as described above, may include job-related information such as a customer account identifier, job identifier, storage location identifier, reference to data stored in payload data cache 228, job status, job creation and/or expiration time and the like. In some embodiments, a storage job may be created before a data object identifier is generated and/or provided. For example, a storage job identifier, instead of or in addition to a data object identifier, may be provided in response to a storage request at step 508 above.

[0126] In an embodiment, scheduling 512 the storage job for execution includes performing job planning and optimization, such as queue-based load leveling or balancing, job partitioning and the like, as described in connection with common control plane 212 of FIG. 2. For example, in an embodiment, job request balancer 238 transfers job items from job request queue 236 to storage node manager job store 240 according to a scheduling algorithm so as to dampen peak to average load levels (jobs) coming from control plane for I/O 210 and to deliver manageable workload to data plane 214. As another example, storage node manager job store 240 may be partitioned to facilitate parallel processing of the jobs by multiple workers such as storage node managers 244. As yet another example, storage node manager job store 240 may provide querying, sorting and other functionalities to facilitate batch processing and other job optimizations.

[0127] In an embodiment, process 500 includes selecting 514 the storage job for execution, for example, by a storage node manager 244 from storage node manager job stored 240 as described in connection with FIG. 2. The storage job may be selected 514 with other jobs for batch processing or otherwise selected as a result of job planning and optimization described above.

[0128] In an embodiment, process 500 includes obtaining 516 data from a staging store, such as payload data cache 228 described above in connection with FIG. 2. In some embodiments, the integrity of the data may be checked, for example, by verifying the size, digest, an error-detection code and the like.

[0129] In an embodiment, process 500 includes obtaining 518 one or more data encoding schemes such as an encryption scheme, a redundancy encoding scheme such as erasure encoding, redundant array of independent disks (RAID) encoding schemes, replication, and the

like. In some embodiments, such encoding schemes evolve to adapt to different requirements. For example, encryption keys may be rotated periodically and stretch factor of an erasure coding scheme may be adjusted over time to different hardware configurations, redundancy requirements and the like.

[0130] In an embodiment, process 500 includes encoding 520 with the obtained encoding schemes. For example, in an embodiment, data is encrypted and the encrypted data is erasure-encoded. In an embodiment, storage node managers 244 described in connection with FIG. 2 may be configured to perform the data encoding described herein. In an embodiment, application of such encoding schemes generates a plurality of encoded data components or shards, which may be stored across different storage entities such as storage devices, storage nodes, datacenters and the like to provide fault tolerance. In an embodiment where data may comprise multiple parts (such as in the case of a multi-part upload), each part may be encoded and stored as described herein.

[0131] In an embodiment, process 500 includes determining 522 the storage entities for such encoded data components. For example, in an environment 200 illustrated by FIG. 2, a storage node manager 244 may determine the plurality of storage nodes 246 to store the encoded data components by querying a storage node registrar 248 using a volume identifier. Such a volume identifier may be part of a storage location identifier associated with the data to be stored. In response to the query with a given volume identifier, in an embodiment, storage node registrar 248 returns a list of network locations (including endpoints, DNS names, IP addresses and the like) of storage nodes 246 to store the encoded data components. As described in connection with FIG. 2, storage node registrar 248 may determine such a list based on self-reported and dynamically provided and/or updated inventory information from storage nodes 246 themselves. In some embodiments, such determination is based on data isolation, fault tolerance, load balancing, power conservation, data locality and other considerations. In some embodiments, storage registrar 248 may cause new storage space to be allocated, for example, by invoking storage allocator 256 as described in connection with FIG. 2.

[0132] In an embodiment, process 500 includes causing 524 storage of the encoded data component(s) at the determined storage entities. For example, in an environment 200 illustrated

by FIG. 2, a storage node manager 244 may request each of the storage nodes 246 determined above to store a data component at a given storage location. Each of the storage nodes 246, upon receiving the storage request from storage node manager 244 to store a data component, may cause the data component to be stored in a connected storage device. In some embodiments, at least a portion of the data object identifier is stored with all or some of the data components in either encoded or unencoded form. For example, the data object identifier may be stored in the header of each data component and/or in a volume component index stored in a volume component. In some embodiments, a storage node 246 may perform batch processing or other optimizations to process requests from storage node managers 244.

[0133] In an embodiment, a storage node 246 sends an acknowledgement to the requesting storage node manager 244 indicating whether data is stored successfully. In some embodiments, a storage node 246 returns an error message, when for some reason, the request cannot be fulfilled. For example, if a storage node receives two requests to store to the same storage location, one or both requests may fail. In an embodiment, a storage node 246 performs validation checks prior to storing the data and returns an error if the validation checks fail. For example, data integrity may be verified by checking an error-detection code or a digest. As another example, storage node 246 may verify, for example, based on a volume index, that the volume identified by a storage request is stored by the storage node and/or that the volume has sufficient space to store the data component.

[0134] In some embodiments, data storage is considered successful when storage node manager 244 receives positive acknowledgement from at least a subset (a storage quorum) of requested storage nodes 246. In some embodiments, a storage node manager 244 may wait until the receipt of a quorum of acknowledgement before removing the state necessary to retry the job. Such state information may include encoded data components for which an acknowledgement has not been received. In other embodiments, to improve the throughput, a storage node manager 244 may remove the state necessary to retry the job before receiving a quorum of acknowledgement.

[0135] In an embodiment, process 500 includes updating 526 metadata information including, for example, metadata maintained by data plane 214 (such as index and storage space

information for a storage device, mapping information stored at storage node registrar store 250 and the like), metadata maintained by control planes 210 and 212 (such as job-related information), metadata maintained by metadata plane 216 (such as a cold index) and the like. In various embodiments, some of such metadata information may be updated via batch processing and/or on a periodic basis to reduce performance and cost impact. For example, in data plane 214, information maintained by storage node registrar store 250 may be updated to provide additional mapping of the volume identifier of the newly stored data and the storage nodes 246 on which the data components are stored, if such a mapping is not already there. For another example, volume index on storage devices may be updated to reflect newly added data components.

[0136] In common control plane 212, job entries for completed jobs may be removed from storage node manager job store 240 and added to job result queue 242 as described in connection with FIG. 2. In control plane for direct I/O 210, statuses of job records in job tracker store 232 may be updated, for example, by job tracker 230 which monitors the job result queue 242. In various embodiments, a job that fails to complete may be retried for a number of times. For example, in an embodiment, a new job may be created to store the data at a different location. As another example, an existing job record (e.g., in storage node manager job store 240, job tracker store 232 and the like) may be updated to facilitate retry of the same job.

[0137] In metadata plane 216, metadata may be updated to reflect the newly stored data. For example, completed jobs may be pulled from job result queue 242 into metadata manager job store 258 and batch-processed by metadata manager 260 to generate an updated index such as stored in cold index store 262. For another example, customer information may be updated to reflect changes for metering and accounting purposes.

[0138] Finally, in some embodiments, once a storage job is completed successfully, job records, payload data and other data associated with a storage job may be removed, for example, by a cleanup agent 234 as described in connection with FIG. 2. In some embodiments, such removal may be processed by batch processing, parallel processing or the like.

[0139] FIG. 6 illustrates an example process 500 for retrieving data, in accordance with at least one embodiment. In an embodiment, one or more components of archival data storage system 206 as described in connection with FIG. 2 collectively perform process 600.

[0140] In an embodiment, process 600 includes receiving 602 a data retrieval request to retrieve data such as stored by process 500, described above. Such a data retrieval request may include a data object identifier, such as provided by step 508 of process 500, described above, or any other information that may be used to identify the data to be retrieved.

[0141] In an embodiment, process 600 includes processing 604 the data retrieval request upon receiving 602 the request. Such processing may include, for example, authenticating the customer, authorizing requested access against access control policies, performing meter and accounting related activities and the like. In an embodiment, such processing may be performed by services of front end 208 such as described in connection with FIG. 2. In an embodiment, such request may be processed in connection with other requests, for example, in batch mode.

[0142] In an embodiment, processing 604 the retrieval request may be based at least in part on the data object identifier that is included in the retrieval request. As described above, data object identifier may encode storage location information, payload validation information such as size, creation timestamp, payload digest and the like, metadata validation information, policy information and the like. In an embodiment, processing 604 the retrieval request includes decoding the information encoded in the data object identifier, for example, using a private cryptographic key and using at least some of the decoded information to validate the retrieval request. For example, policy information may include access control information that may be used to validate that the requesting entity of the retrieval request has the required permission to perform the requested access. As another example, metadata validation information may include an error-detection code such as a cyclic redundancy check ("CRC") that may be used to verify the integrity of data object identifier or a component of it.

[0143] In an embodiment, process 600 includes creating 606 a data retrieval job corresponding to the data retrieval request and providing 608 a job identifier associated with the data retrieval job, for example, in a response to the data retrieval request. In some embodiments, creating 606 a data retrieval job is similar to creating a data storage job as described in connection with

step 510 of process 500 illustrated in FIG. 5. For example, in an embodiment, a job tracker 230 may create a job record that includes at least some information encoded in the data object identifier and/or additional information such as a job expiration time and the like and store the job record in job tracker store 232. As described above, job tracker 230 may perform batch processing to reduce the total number of transactions against job tracker store 232. Additionally, job tracker store 232 may be partitioned or otherwise optimized to facilitate parallel processing, cleanup operations and the like.

[0144] In an embodiment, process 600 includes scheduling 610 the data retrieval job created above. In some embodiments, scheduling 610 the data retrieval job for execution includes performing job planning and optimization such as described in connection with step 512 of process 500 of FIG. 5. For example, the data retrieval job may be submitted into a job queue and scheduled for batch processing with other jobs based at least in part on costs, power management schedules and the like. For another example, the data retrieval job may be coalesced with other retrieval jobs based on data locality and the like.

[0145] In an embodiment, process 600 includes selecting 612 the data retrieval job for execution, for example, by a storage node manager 244 from storage node manager job stored 240 as described in connection with FIG. 2. The retrieval job may be selected 612 with other jobs for batch processing or otherwise selected as a result of job planning and optimization described above.

[0146] In an embodiment, process 600 includes determining 614 the storage entities that store the encoded data components that are generated by a storage process such as process 500 described above. In an embodiment, a storage node manager 244 may determine a plurality of storage nodes 246 to retrieve the encoded data components in a manner similar to that discussed in connection with step 522 of process 500, above. For example, such determination may be based on load balancing, power conservation, efficiency and other considerations.

[0147] In an embodiment, process 600 includes determining 616 one or more data decoding schemes that may be used to decode retrieved data. Typically, such decoding schemes correspond to the encoding schemes applied to the original data when the original data is

previously stored. For example, such decoding schemes may include decryption with a cryptographic key, erasure-decoding and the like.

[0148] In an embodiment, process 600 includes causing 618 retrieval of at least some of the encoded data components from the storage entities determined in step 614 of process 600. For example, in an environment 200 illustrated by FIG. 2, a storage node manager 244 responsible for the data retrieval job may request a subset of storage nodes 246 determined above to retrieve their corresponding data components. In some embodiments, a minimum number of encoded data components is needed to reconstruct the original data where the number may be determined based at least in part on the data redundancy scheme used to encode the data (e.g., stretch factor of an erasure coding). In such embodiments, the subset of storage nodes may be selected such that no less than the minimum number of encoded data components is retrieved.

[0149] Each of the subset of storage nodes 246, upon receiving a request from storage node manager 244 to retrieve a data component, may validate the request, for example, by checking the integrity of a storage location identifier (that is part of the data object identifier), verifying that the storage node indeed holds the requested data component and the like. Upon a successful validation, the storage node may locate the data component based at least in part on the storage location identifier. For example, as described above, the storage location identifier may include a volume reference object which comprises a volume identifier component and a data object identifier component where the volume reference component to identify the volume the data is stored and a data object identifier component may identify where in the volume the data is stored. In an embodiment, the storage node reads the data component, for example, from a connected data storage device and sends the retrieved data component to the storage node manager that requested the retrieval. In some embodiments, the data integrity is checked, for example, by verifying the data component identifier or a portion thereof is identical to that indicated by the data component identifier associated with the retrieval job. In some embodiments, a storage node may perform batching or other job optimization in connection with retrieval of a data component.

[0150] In an embodiment, process 600 includes decoding 620, at least the minimum number of the retrieved encoded data components with the one or more data decoding schemes determined

at step 616 of process 600. For example, in one embodiment, the retrieved data components may be erasure decoded and then decrypted. In some embodiments, a data integrity check is performed on the reconstructed data, for example, using payload integrity validation information encoded in the data object identifier (e.g., size, timestamp, digest). In some cases, the retrieval job may fail due to a less-than-minimum number of retrieved data components, failure of data integrity check and the like. In such cases, the retrieval job may be retried in a fashion similar to that described in connection with FIG. 5. In some embodiments, the original data comprises multiple parts of data and each part is encoded and stored. In such embodiments, during retrieval, the encoded data components for each part of the data may be retrieved and decoded (e.g., erasure-decoded and decrypted) to form the original part and the decoded parts may be combined to form the original data.

[0151] In an embodiment, process 600 includes storing reconstructed data in a staging store such as payload data cache 228 described in connection with FIG. 2. In some embodiments, data stored 622 in the staging store may be available for download by a customer for a period of time or indefinitely. In an embodiment, data integrity may be checked (e.g., using a digest) before the data is stored in the staging store.

[0152] In an embodiment, process 600 includes providing 624 a notification of the completion of the retrieval job to the requestor of the retrieval request or another entity or entities otherwise configured to receive such a notification. Such notifications may be provided individually or in batches. In other embodiments, the status of the retrieval job may be provided upon a polling request, for example, from a customer.

[0153] FIG. 7 illustrates an example process 700 for deleting data, in accordance with at least one embodiment. In an embodiment, one or more components of archival data storage system 206 as described in connection with FIG. 2 collectively perform process 700.

[0154] In an embodiment, process 700 includes receiving 702 a data deletion request to delete data such as stored by process 500, described above. Such a data retrieval request may include a data object identifier, such as provided by step 508 of process 500, described above, or any other information that may be used to identify the data to be deleted.

[0155] In an embodiment, process 700 includes processing 704 the data deletion request upon receiving 702 the request. In some embodiments, the processing 704 is similar to that for step 504 of process 500 and step 604 of process 600, described above. For example, in an embodiment, the processing 704 is based at least in part on the data object identifier that is included in the data deletion request.

[0156] In an embodiment, process 700 includes creating 706 a data retrieval job corresponding to the data deletion request. Such a retrieval job may be created similar to the creation of storage job described in connection with step 510 of process 500 and the creation of the retrieval job described in connection with step 606 of process 600.

[0157] In an embodiment, process 700 includes providing 708 an acknowledgement that the data is deleted. In some embodiments, such acknowledgement may be provided in response to the data deletion request so as to provide a perception that the data deletion request is handled synchronously. In other embodiments, a job identifier associated with the data deletion job may be provided similar to the providing of job identifiers for data retrieval requests.

[0158] In an embodiment, process 700 includes scheduling 708 the data deletion job for execution. In some embodiments, scheduling 708 of data deletion jobs may be implemented similar to that described in connection with step 512 of process 500 and in connection with step 610 of process 600, described above. For example, data deletion jobs for closely-located data may be coalesced and/or batch processed. For another example, data deletion jobs may be assigned a lower priority than data retrieval jobs.

[0159] In some embodiments, data stored may have an associated expiration time that is specified by a customer or set by default. In such embodiments, a deletion job may be created 706 and schedule 710 automatically on or near the expiration time of the data. In some embodiments, the expiration time may be further associated with a grace period during which data is still available or recoverable. In some embodiments, a notification of the pending deletion may be provided before, on or after the expiration time.

[0160] In some embodiments, process 700 includes selecting 712 the data deletion job for execution, for example, by a storage node manager 244 from storage node manager job

stored 240 as described in connection with FIG. 2. The deletion job may be selected 712 with other jobs for batch processing or otherwise selected as a result of job planning and optimization described above.

[0161] In some embodiments, process 700 includes determining 714 the storage entities for data components that store the data components that are generated by a storage process such as process 500 described above. In an embodiment, a storage node manager 244 may determine a plurality of storage nodes 246 to retrieve the encoded data components in a manner similar to that discussed in connection with step 614 of process 600 described above.

[0162] In some embodiments, process 700 includes causing 716 the deletion of at least some of the data components. For example, in an environment 200 illustrated by FIG. 2, a storage node manager 244 responsible for the data deletion job may identify a set of storage nodes that store the data components for the data to be deleted and requests at least a subset of those storage nodes to delete their respective data components. Each of the subset of storage node 246, upon receiving a request from storage node manager 244 to delete a data component, may validate the request, for example, by checking the integrity of a storage location identifier (that is part of the data object identifier), verifying that the storage node indeed holds the requested data component and the like. Upon a successful validation, the storage node may delete the data component from a connected storage device and sends an acknowledgement to storage node manager 244 indicating whether the operation was successful. In an embodiment, multiple data deletion jobs may be executed in a batch such that data objects located close together may be deleted as a whole. In some embodiments, data deletion is considered successful when storage node manager 244 receives positive acknowledgement from at least a subset of storage nodes 246. The size of the subset may be configured to ensure that data cannot be reconstructed later on from undeleted data components. Failed or incomplete data deletion jobs may be retried in a manner similar to the retrying of data storage jobs and data retrieval jobs, described in connection with process 500 and process 600, respectively.

[0163] In an embodiment, process 700 includes updating 718 metadata information such as that described in connection with step 526 of process 500. For example, storage nodes executing the deletion operation may update storage information including index, free space information

and the like. In an embodiment, storage nodes may provide updates to storage node registrar or storage node registrar store. In various embodiments, some of such metadata information may be updated via batch processing and/or on a periodic basis to reduce performance and cost impact.

[0164] FIG. 8 depicts an illustrative flow 800 in which techniques for the validation of data integrity may be implemented. These techniques are described in more detail below in connection with FIGS. 9-13. Returning to FIG. 8, in illustrative flow 800, operations may be performed by one or more processors of an archival data storage service and/or instructions that perform the operations may be stored in one or more memories of the archival data storage service. As desired, the flow 800 may begin at 802, where the archival data storage service may receive one or more parts of a data payload 804. In some examples, the data payload 804 may include any number of parts; however, in this example two parts are shown, Part 1 and Part 2. Each of Part 1 and Part 2 may include data 806 and 808, respectively. In some cases, the size of Part 1 and Part 2 may be selected by the uploader and/or may be the same. However, in some examples, the last part of a data payload 804 may be a different size from all the other consistently sized parts (e.g., as shown here in FIG. 8, wherein Part 1 is bigger than Part 2). At 810, the flow 800 may generate sub-parts of the parts of the payload 804. In some examples, the size of the sub-parts may be predefined by the archival data storage service (e.g., 1MB).

[0165] In some examples, the flow 800 may calculate a digest for each sub-part at 812. The respective digests may be stored as nodes 814 of a data structure such as, but not limited to, the data structure 816 generated at 818. By way of example only, the data structure 816 may include one or more sub-part digests (e.g., at nodes 814) and/or one or more part digests (e.g., Part 1 digest 820 and Part 2 digest 822). Additionally, at 824, the flow 800 may determine a root digest 826 for the root of the data structure 816. In some examples, the root digest 826 may be determined or generated based at least in part on concatenating part digests and calculating a digest for the concatenated digests. The flow 800 may end at 828, where the archival data storage service may verify that the received payload 804 matches a stored payload 830. The stored payload may, in some examples, contain data 832 determined based at least in part on combining each of the parts 806, 808 and/or sub-parts, when received. In some examples, verifying the data

payload may be based at least in part on comparing the root digest 826 against a second root digest received from the uploader.

[0166] FIG. 9 depicts an illustrative data structure 900 in which additional techniques for the validation of data integrity may be implemented. Illustrative data structure 900 is but one of many different types of data structures that may be utilized to implement the techniques described herein. By way of example only, a user or client entity may wish to upload a data payload 902 to the archival data storage service. The archival data storage service may then be configured to receive the data payload 902 (in one or more parts) and allow the user to verify, at some point (e.g., immediately after upload or after some time, in some cases, after a relatively long time), that the data stored in the archival data storage service is, in fact, the same as the data payload 902 that was uploaded without requesting any size partitioning information from the user. In other words, the archival data storage service may provide a data object identifier that the user may return in order to retrieve stored data; however, the user may not need to store any information other than the data object identifier.

[0167] In some examples, in order to accept data from the user, the archival data storage service may request that the user provide a tree digest like the data structure 900 of FIG. 9. Providing the data structure 900 may be performed in multiple ways in accordance with various embodiments. For example, all of the data illustrated in in the data structure 900 may be provided. As an alternative, in embodiments where the data structure 900 is constructible solely from the data for the leaf nodes, data for the leaf nodes may be provided without providing information for other, higher-level nodes. Additionally, the archival data storage service may provide instructions in the form of an algorithm, API and/or SDK for generating the data structure 900. In some instances, limitations on the size of upload chunks and their respective offsets may be imposed. For example, the chunks or parts of the data payload 902 may be limited to powers of two of 1MB. Additionally, in some examples, the determined size of each chunk may not be changed within a particular upload. Further, for each part received, the archival data storage service may calculate its own digest, based at least in part on the same algorithm used by the user, and provide the digest for each part. Upon completion of the storage job, the archival data storage service may provide the top-level digest value in the form of a data object identifier.

Retrieval of the data may, in some examples, be implemented in a similar fashion, with restrictions on chunk sizes and offsets limited to powers of two by 1MB, messages prepended with the digest of the data that is in the message and the top-level digest available upon completion of the job. However, based at least in part on this implementation, the data payload 902 should be able to be verified or validated independent of the chunk size selected by the user. A digest may be calculated by applying a cryptographic hash function such as those associated with SHA-1, SHA-2, MD5, MD6 and the like, a checksum or error-detection code such as cyclic redundancy check and the like to at least a portion of the payload data.

[0168] The data structure 900 of FIG. 9 may illustrate an appropriate digest tree for a data payload 902 where the user has chosen to upload the data payload in a single part. As such, there is no part size for the user to select in this example. However, the resulting root digest 906 should be calculable using the techniques described herein even if the user had selected to upload the data payload 902 in multiple parts, and even if the user had selected a part size unknown to the archival data storage service and/or not recorded by the user. In this example, for the sake of simplicity, it will be assumed that the data payload 902 is 7MBs in size. As such, and since the user has requested to upload the entire payload 902 in one part, the data payload 902 may be partitioned into seven 1MB chunks, Sub 1 – Sub 7. In some examples, however, if the size of the payload 902 were not divisible by 1MB, the last chunk, Sub 7, may be smaller than 1MB. The archival data storage service may, based at least in part on the hash tree algorithm, generate a hash value (or digest) for each 1MB chunk (i.e., Sub 1 – Sub 7). Each of these hash values may be represented at the lowest child node level 908 of the data structure 900. In order to generate the nodes of the second child node level 910, the archival data storage service may concatenate each pair of second-level node children and run the hash function on the concatenated data. In other words, the lowest level 908 of the data structure may include a digest of payload data, while parent nodes may include digests of digests. Moving up the data structure, the described operations may be repeated until a root digest 906 is generated.

[0169] As described, in some cases, the archival data storage service may provide intermediary root digests for individual parts of the payload 902. However, in this example, since the payload was not broken into parts, the archival data storage service may only provide the root digest 906 to the user. In some cases, though, the archival data storage service may also provide each 1MB

digest generated. As such, either the user or the archival data storage service should be able to verify that the data was uploaded correctly (including at the 1MB sub-part level) based at least in part on comparing each other's generated root digest 906.

[0170] FIG. 10 depicts another illustrative data structure 1000 in which additional techniques for the validation of data integrity may be implemented. As noted with reference to FIG. 9, the illustrative data structure 1000 is but one of many different types of data structures that may be utilized to implement the techniques described herein. By way of example only, a user or client entity may wish to upload a data payload 1002 to the archival data storage service. The archival data storage service may then be configured to receive the data payload 1002 (in this example, in two parts) and allow the user to verify that the data stored in the archival data storage service is, in fact, the same as the data payload 1002 that was uploaded. This validation may be done without requesting any size partitioning information from the user. In other words, the archival data storage service may provide a data object identifier that the user may return in order to retrieve stored data; however, the user may not need to store any information other than the data object identifier in order to request and/or validate the stored data.

[0171] In generating the data structure 1000, the user or the archival data storage service may once again break the data into sub-parts; however, in this example, each part Part 1 or Part 2 may be broken up separately (e.g., Sub 1 – Sub 4 of Part 1 and Sub 1 – Sub 3). Again, a digest for each sub-part may be generated and included in the data structure at the child level 1004 and digests of concatenated digests may be generated and included in the data structure at a first parent level 1006. In this example, however, since the payload 1002 has been broken into two parts, a top-level digest may be generated for each part. As such, Part 1 digest 1008 and Part 2 digest 1010 may be generated and included in the data structure 1000. Additionally, as the payload 1002 is uploaded, each of the sub-part digests (e.g., those at 1004) and the part digests (e.g., those at 1008) may be included in the upload. Further, a root digest 1012 may be generated in the same fashion that the other parent nodes are generated. That is, based at least in part on concatenating the children digests, and running the hash function on the concatenated information. In this example, this process would entail concatenating Part 1 digest 1008 and Part 2 digest 1010 to generate a part-level digest. The archival data storage service may then run the hash function on the part-level digest to generate the root digest 1006. In some examples, the

root digest may be received at the beginning of upload and once the upload is completed. Additionally, the archival data storage service may generate its own version of the data structure 1000 and/or the root digest 1006 in order to validate the integrity of the data. Further, in some examples, the root digest 1006 generated by the archival data storage service may be provided to the user as part of a data object identifier that the user may utilize to make read, delete or index viewing requests.

[0172] In some examples, the archival data storage service may assume that data corruptions can occur anywhere in the system and/or may be caused by hardware bugs, bit flips and/or due to bugs in software code implemented by the archival data storage service or the user. For at least this reason, the archival data storage service may review all, or a subset, of the data paths and operations to ensure that data integrity is provided throughout the system and that corrupt data is detected. In some cases, this may apply to the data payload (e.g., that stored in the archival data storage service) and to the metadata. As such, data integrity validation may be performed on the data object identifiers as well to ensure that requests to delete data are not pointing at the wrong data.

[0173] In some aspects, the archival data storage service 206 may be configured to expect that a selected or otherwise determined digest function may be acceptable for the validation of data integrity. In some examples, the digest function may not be used for some cases related to data transformation. Otherwise, it may be selected and/or provided for use with validating some, al, or portions of the data and/or metadata of the archival data storage service. Additionally, as noted, in some examples, the initiator (i.e., the user) may pre-calculate the digest of the data payload 1002 before transmission and then later may supply the digest again with the data to the archival data storage service. The archival data storage service may then recalculate the digest (e.g., the top-level digest), compare with the digest received from the initiator and/or acknowledge that the integrity of the data was validated by providing the archival data storage service-generated digest to the user. Additionally, each data subdivision and/or aggregation (e.g., the sub-parts, the parts, the part-level digests and/or the root digests) may be re-validated by calculating the independent digest on the split of the aggregate data and comparing the digest or even by performing a bit by bit comparison. In other words, given any data payload, of any size, calculations may be

performed to generate any number or type of the split or aggregated digests and, thus, validate the data and/or the parts.

[0174] Additionally, in some aspects, data transformation such as, but not limited to, erasure coding or encryption can be re-validated by performing the reverse transformation. The results of the reverse transformation may then be cross checked by comparing the digest and/or by bit by bit comparison. As such, the transformed data may include two digests. One of the two may testify to the integrity of the transformed data and the other may testify to the integrity of the original data. In some examples, referential items such as, but not limited to, the data object identifier that may reference the content may include the digest of the data being referenced. Additionally, the archival data storage service may also include information about the parent node that is being referenced. In some cases, messages from the control plane that are persisted in the storage node registrar store 250, the data object identifier and/or other data structures may include digests that self-validate. These digests may be produced after the structures are created and/or verified upon retrieval or before the action. In some examples, this prevent things such as bugs in the code, memory corruption or bit rot from flipping data object retrieve commands into delete commands. Further, on the return path, the archival data storage service may be configured to re-validate that the data that is being returned to the customer is matched against the request and/or that no substitution during execution happens due to a bug in the code or other the like.

[0175] FIGS. 11-13 illustrate example flow diagrams showing respective processes 1100-1300 for providing validation of data integrity. These processes are illustrated as logical flow diagrams, each operation of which represents a sequence of operations that can be implemented in hardware, computer instructions, or a combination thereof. In the context of computer instructions, the operations represent computer-executable instructions stored on one or more computer-readable storage media that, when executed by one or more processors, perform the recited operations. Generally, computer-executable instructions include routines, programs, objects, components, data structures and the like that perform particular functions or implement particular data types. The order in which the operations are described is not intended to be construed as a limitation, and any number of the described operations can be combined in any order and/or in parallel to implement the processes.

[0176] Additionally, some, any, or all of the processes may be performed under the control of one or more computer systems configured with executable instructions and may be implemented as code (e.g., executable instructions, one or more computer programs, or one or more applications) executing collectively on one or more processors, by hardware, or combinations thereof. As noted above, the code may be stored on a computer-readable storage medium, for example, in the form of a computer program comprising a plurality of instructions executable by one or more processors. The computer-readable storage medium may be non-transitory.

[0177] In some aspects, the API request handler 218, the payload data cache 228, the storage node manager 244 and/or the storage nodes 246 of the one or more archival data storage service 206 shown in FIG. 2 may perform the process 1100 of FIG. 11. The process 1100 may begin by providing (e.g., to a user or client entity of the archival data storage service) a function call for requesting data storage at 1102. The function call may be part of an API or an SDK for interacting with and/or interfacing with the archival data storage service. At 1104, the process 1100 may include receiving a plurality of portions of a data payload from a remote computing device (i.e., the user). In some cases, the size of each portion may be consistent. In other cases, the size of each portion may be consistent except that the last portion may be different. Additionally, the size may be selected or otherwise determined by the user. At 1106, the process 1100 may include receiving an indication of the size of each portion. In some instances, the actions performed at 1104 and 1106 may be performed together as a single action. However, some restrictions may apply regarding portion size. Such as, in some examples, the portions may be limited to a consistent size (i.e., they may be required to be the same size); however, the last portion may be a remainder of the data payload (i.e., the payload minus each other consistently-sized portion). For example, the size selection may be limited to 1MB or an integer multiple of 1MB. In other examples, the size may be limited to 1MB or a power of two of 1MB.

[0178] The process 1100 may also include generating one or more sub-portions of a predefined size for at least some of the portions of the payload at 1108. As noted above, regarding the portion size, while sub-portion size may be predefined and constant, the last sub-portion may be a different size. The predefined size may be 1MB or any other size. At 1110, the process 1100 may include calculating one or more digests or hash values based at least in part on the sub-portions. The digests may be calculated based at least in part on a published and/or otherwise

provided algorithm. In some examples, at 1112, the process 1100 may include generating a root node of a data structure. The data structure may be based at least in part on the sub-portion digests and/or aggregated digests as described above. At 1114, the process 1100 may include determining a top-level digest of the data structure. The top-level digest may be based at least in part on the root node of the data structure and/or on a parent node associated with one of the portions of data. At 1116, the process 1100 may include providing instructions configured to enable the remote computing device to generate the data structure. In this way, the user may generate the data structure along with the archival data storage service. The process 1100 may then include receiving a top-level digest generated by the remote computing device at 1118. The process 1100 may end at 1120 by verifying that the stored data payload matches a received data payload. In other words, the process 1100 may validate or verify the integrity of the data.

[0179] FIG. 12 illustrates another example flow diagram showing process 1200 for validating the integrity of data. In some aspects, the API request handler 218, the payload data cache 228, the storage node manager 244 and/or the storage nodes 246 of the one or more archival data storage service 206 shown in FIG. 2 may perform the process 1200 of FIG. 12. The process 1200 may begin by receiving one or more parts of a data payload at 1202. As noted above, the parts may be any size. However, in some examples, the part size may be limited to 1MB or multiples of 1MB. In this way, a data structure may be composed independent of the chosen size. At 1204, the process 1200 may include generating a sub-part for the one or more parts. Again, these sub-parts may be any size or may be limited to 1MB or other size limitation such as, but not limited to, 2MB, 10MB, etc. the process 1200 may include calculating a value based on the sub-part at 1206. The value may, in some cases, be a hash value, a digest, or other result of encryption. In some examples, the process 1200 may include generating a root node of a data structure at 1208. At 1210, the process 1200 may include determining a top-level value of the data structure based at least in part on traversing the data structure to the root node.

[0180] In some examples, the process 1200 may also include storing the data payload at 1212. The payload may be stored based at least in part on combining each of the one or more parts received at 1202. As such, in some cases, the archival data storage service 206 may not be able to store the payload at 1212 until the data transmission of all the parts is complete. At 1214, the process 1200 may include validating that the stored data payload matches the received data

payload. This may be performed by comparing the received top-level value with a calculated top-level value. At 1216, the process 1200 may include providing a data object identifier including the top-level value. The identifier may later be utilized by the user to retrieve and/or delete the stored data payload. In some examples, the process 1200 may include receiving a request for the stored payload at 1218. The stored payload may be provided back to the user in a similar fashion (with the integrity of the data being validated each step of the way). However, in some cases, the process 1200 may end at 1220 by verifying that the stored data payload has not changed prior to providing the payload.

[0181] FIG. 13 illustrates another example flow diagram showing process 1300 for validating the integrity of data. In some aspects, the API request handler 218, the payload data cache 228, the storage node manager 244 and/or the storage nodes 246 of the one or more archival data storage service 206 shown in FIG. 2 may perform the process 1300 of FIG. 13. The process 1300 may begin by providing instructions for making method calls to perform operations on data at 1302. In some examples, these method calls may be exposed via one or more APIs or provided in one or more SDKs. At 1304, the process 1300 may include performing a first operation, using a verification algorithm, based on a first partitioning of a data object into first partitions. The first partitions, in some examples, may include 1MB or other consistent sized chunks that may be utilized to generate a data structure such as, but not limited to, a hash tree or other binary tree of digests. In some examples, the first operation may include receiving the data from a user over a network. At 1306, the process 1300 may include verifying the data object to generate a first verification value (e.g., a hash code, checksum, etc.) based on the first partitions. The process 1300 may also include performing a second operation on a data object, utilizing the same verification algorithm, based at least in part on a second partitioning of the data object into second partitions at 1308. The second partitions may be a different size from the first partitions. Based at least in part on the second partitions, the process 1300 may include verifying the data object to generate a second value at 1310. Here, the second operation may also include transmitting data to the archival data storage service. The second verification value, like the first may include, but is not limited to, a digest for a partition, a digest for digests formed by aggregating partition digests and/or a top-level digest of a data structure. At 1312, the process 1300 may end by determining whether the second verification value equals the first verification

value. This may be determined based at least in part on comparing the two values. In some examples, if the verification algorithm is properly performed, and the data has maintained its integrity, the two values are expected to be equal. That is, independent of the size of the two sets of partitions (i.e., the first partitioning and the second partitioning), the verification values should be equal.

[0182] FIG. 14 depicts an illustrative architecture 1400 in which techniques for storage drive rotation scheduling may be implemented. In illustrative architecture 1400, an archival data storage system 1402 may provide one or more archival services including, but not limited to, archival data storage, batch job processing, data retrieval, data management, storage device management and/or rotation scheduling. In some aspects, the archival data storage system 1402 may determine and/or store a rotation schedule 1404. As noted above, the rotation schedule 1404 may indicate which drives are to be activated and when. Additionally, the rotation schedule 1404 may be determined based at least in part on several factors and/or may be provided by a data storage node manager and/or a data storage node. In some examples, the rotation schedule 1404 may be provided to a controller 1406. In some examples, the controller 1406 may include the one or more storage nodes noted above. Alternatively, or in addition, the controller 1406 may include, but is not limited to, a dedicated memory controller, a Small Computer System Interface (SCSI) controller, an Integrated Drive Electronics (IDE) controller, a Fibre Channel controller, a Serial Attached SCSI controller or the like. Additionally, the controller 1406 could be an abstraction of such controllers, or some "supercontroller" that communicates with such drive controllers via the protocols listed. By way of example only, the controller 1406 may be configured to control one or more hard drive (or hard disk drive (HDD)) groups of a rack 1408 or other configuration based at least in part on one or more instructions 1410. For example, instruction 1412 may correspond to an action to be performed on HDD Group 1, instruction 1414 may correspond to HDD Group 2 and so on. In this way, the controller 1406 may be configured to perform different instructions on each HDD Group based at least in part on the schedule 1404.

[0183] In one non-limiting example, the controller 1406 may follow the schedule 1404 to activate HDD Group 4 based at least in part on instruction 1418, labeled "ON." That is, instruction 1418 may correspond to HDD Group 4 and may indicate that HDD Group 4 should be active for a predetermined amount of time. As noted above, the amount of time each group

may be active may be based at least in part on several factors including, but not limited to, a portion of an hour based at least in part on a division of groups and/or some on-demand requests of the storage node. After the predefined time period has expired, the controller 1406 may move the "ON" instruction 1418 down the chain of instructions 1410 such that HDD Group 5 may be active next. As such, instruction 1420 is labeled "Power Up," to indicate that HDD Group 5 may be scheduled to begin powering up at some point in the future. Similarly, instruction 1416 is labeled "Power Down," to indicate that HDD Group 3 may still be powering down after being active. In at least one example, after a predefined time (e.g., based at least in part on the schedule), the controller 1406 may shift the instructions down one such that instruction 1418 may be labeled "ON," and HDD Group 5 may become activated. Other scenarios and/or examples may be possible for scheduling rotation of the drives. For example, each HDD group may include only a single storage drive such that each HDD Group may include one or more drives and is not limited to a scenario where each group includes multiple drives. Additionally, as noted above, additional instructions may be processed in order to avoid power consumption spikes based at least in part on activating multiple drives at the same time. For example, the drives of a group that are scheduled to rotate during a certain period may be scheduled to activate one at a time or in sub-groups as desired. Additionally, the HDD Groups (and/or individual hard drives) may not be listed and/or may not be in any particular order, as shown. For example, the controller 106 may instead keep track in a table, chart, index, etc., which drives or Groups have been activated during a particular period and randomly (or on demand) cycle through activation of drives that have yet to be activated during the period. Thus, the controller 1406 may be free to determine which drives should be activated and when without being limited by the grouping or list shown in FIG. 14.

[0184] FIG. 15 depicts an illustrative architecture 1500 in which additional techniques for storage drive rotation scheduling may be implemented. In illustrative architecture 1500, an archival data storage system 1502 may provide one or more archival services including, but not limited to, archival data storage, batch job processing, data retrieval, data management, storage device management and/or rotation scheduling. As such, in some aspects, the archival data storage system 1502 may determine and/or store a rotation schedule 1504. As noted above, the rotation schedule 1504 may indicate which drives of a storage service (e.g., those of the archival

data storage system 1504) are to be activated and/or during what time periods. Additionally, the rotation schedule 1504 may be determined based at least in part on several factors and/or may be provided by the data storage node manager 224 and/or the data storage node, as described above.

[0185] In some aspects, a controller 1506 may be configured to control one or more hard drives 1508(1), …, 1508(N) (collectively, "hard drives 1508"). By way of example only, the hard drives 1508 are illustrated in an array or ordered list; however, any ordering and/or other organization may be envisioned for determining which particular hard drives 1508 or sets of hard drives come before or after other particular hard drives 1508 (or sets of hard drives). For example, the array may be utilized by the controller for pointing to a next hard drive to activate (e.g., based at least in part on some logical mapping) and not for ordering the hard drives 1508. However, in other examples, the array may be utilized to dynamically order the hard drives (e.g., when an on-demand rotation schedule is implemented).

[0186] In some examples, a window 1510 (or sliding scale) may be envisioned for activating the hard drives 1508. Based at least in part on a predefined period of time, the window 1510 may move along the array to become windows 1512, 1514 and 1516, for example, at each period. In other words, the window 1510 may represent the hard drives 1508 that are active or are to be activated by the controller 1506 during a first period. That is, by way of example and not limitation, during the first period, the hard drives 1508(1)-1508(5) may be active (i.e., provided power or taken out of sleep mode). In the example shown in FIG. 15, at the end of the first period, the window 1510 may move to become the window 1512, where hard drives 1508(2)-1508(6) become active. In some examples, when the transition occurs (i.e., the window 1510 slides), the hard drive 1508(1) may be powered down prior to the controller 1506 activating the hard drive 1508(6). In this way, the power draw associated with operating the array of hard drives 1508 may not exceed a certain level.

[0187] Additionally, in some aspects, the windows 1510, 1512, 1514, 1516 may be defined by a length 1518. The length may be determined based at least in part on demand, a percentage of the total hard drives 1508, a predetermined number of the total hard drives 1508, some predetermined level of power consumption, temperature, cost, etc. or the like. In addition, the window may be configured to minimize power consumption of the hard drives 1508, minimize operational costs associated with the hard drives 1508, and/or maximize the operational lifetime

of the hard drives 1508 (e.g., how long the hard drives 1508 will work before failing or otherwise being serviced, being repaired, or being replaced). As such, the length 1518 may be constant throughout the rotation schedule 1504 or it may be dynamic. In some examples, once the window 1510 reaches the end of the array of hard drives 1508, the window 1510 may cycle back to the beginning. Pictorially, this may be represented as a circular array, where the hard drive 1508(1) would follow the hard drive 1508(N). Further, in some examples, an array as shown in FIG. 15 may not be used at all. Instead, any other form or type of data structure may be utilized to organize the hard drives 1508 in such a way that the rotation schedule 1504 may determine and/or control which hard drives 1508 are to be activated at what times. Additionally, each of the hard drives 1508 may actually be a group of one or more hard drives instead of a single drive. In this scenario, the controller 1506 may be further configured to stagger the activation of each single hard drive within the group of hard drives such as to limit power spikes or other potential power consumption issues.

[0188] In some examples, the hard drives 1508 may be arranged in a logical order, such as shown with the array 1520. The hard drives 1508 may also be arranged in a server or on a rack in a physical order that may be different from the logical order, such as shown with illustrative physical rack 1522. That is, the hard drives 1508 may actually be located on the physical rack 1522 in the order shown as opposed to the order shown in the logical array 1520. Thus, as the schedule 1504 moves the window 1510 along the logical order 1520, the hard drives 1508 may be activated or otherwise powered up based at least in part on their logical order and/or their physical location. Additionally, a matrix 1524 may indicate which hard drives 1508 are activated at each unit of time, according to the schedule 1504. In some examples, each row or column of the matrix 1524 may represent physical racks in a server or cabinet such that adjacent rows may represent adjacent racks. Additionally, in some examples, one or more constraints may be provided and/or implemented such that an even or other distribution of hard drives 1508 on the matrix 1524 may be ensured. For example, in order to evenly distribute the enabled hard drives 1508, the conditions or constraints may indicate that hard drives 1508 of a particular temperate, age, location, amount of vibration or the like may not be activated that are within a particular distance of other hard drives 1508 of the matrix 1524. For example, the schedule 1504 may indicate that a particular hard drive 1508 is to be activated; however, the condition may instruct

the archival data storage service 1502 to modify the indication or provide an instruction to activate a different hard drive 1508 instead.

[0189] As described with reference to FIG. 15, in some aspects, the total number of hard drives 1508 may be divided into a predefined number of groups. Then, these groups may be rotated in and out of operation based at least in part on the rotation schedule 1504. Such an approach may allow the storage node registrars 248 to determine when and where to place new write operations or when it may be possible to execute read or delete operations. In some examples, the data storage node managers 224 may be isolated from this knowledge and may follow directions from the storage node registrars 248. However, in other examples, the rotation schedule 804 may be determined on-demand. For example, based at least in part on read requests from one or more data storage node managers 224, a storage node registrar 248 may select a number of the hard drives 808 that should come in to a rotation, and then send messages to the data storage nodes. Alternatively, the data storage node manager 224 may directly contact the data storage nodes. In some examples, particular constraints may influence the rotation schedule 1504 including, but not limited to, a goal not to starve, for too long, a request to a drive which has only one read request outstanding.

[0190] FIGS. 16-18 illustrate example flow diagrams showing respective processes 1600-1800 for providing data storage power management. These processes are illustrated as logical flow diagrams, each operation of which represents a sequence of operations that can be implemented in hardware, computer instructions, or a combination thereof. In the context of computer instructions, the operations represent computer-executable instructions stored on one or more computer-readable storage media that, when executed by one or more processors, perform the recited operations. Generally, computer-executable instructions include routines, programs, objects, components, data structures and the like that perform particular functions or implement particular data types. The order in which the operations are described is not intended to be construed as a limitation, and any number of the described operations can be combined in any order and/or in parallel to implement the processes.

[0191] Additionally, some, any or all of the processes may be performed under the control of one or more computer systems configured with executable instructions and may be implemented as code (e.g., executable instructions, one or more computer programs, or one or more

applications) executing collectively on one or more processors, by hardware, or combinations thereof. As noted above, the code may be stored on a computer-readable storage medium, for example, in the form of a computer program comprising a plurality of instructions executable by one or more processors. The computer-readable storage medium may be non-transitory.

[0192] In some aspects, the storage node manager 244, the storage nodes 246, the storage node registrar store 250 and/or other data storage devices or collections of the archival data storage service 206 shown in FIG. 2 may perform the process 1600 of FIG. 16. The process 1600 may begin with classification of one or more data storage devices of the archival data storage service 206 into groups at 1602. In some examples, each group may be a subset or part of a subset of the data storage devices. Additionally, classifying may involve determining an amount or percentage of storage devices to be grouped together based at least in part on characteristics of the storage devices such as, but not limited to, location, power consumption, age, speed, cost, etc. At 1604, the process may include modifying a power state of the groups according to a schedule. The schedule may be configured to ensure that, on average, an amount of the data storage devices are in a power down state. The power down state may be a low power state, a sleep mode, a powering down state, off or the like. Additionally, the amount of devices to be in a power down state may be based at least in part on a desired level of power consumption and/or cost associated with operating the archival data storage service 206. Further, in some examples, the schedule may be received or otherwise based at least in part on the storage node registrar store 250, the storage node registrar 248 and/or the data storage node.

[0193] In some examples, the process 1600 may also include obtaining a batch of data processing requests at 1606. The data processing requests may be part of a batch or finite grouping of requests or they may be part of a queue for more synchronous operation. Additionally, the batch of requests may include read requests, index list requests, write requests, delete requests or the like. At 1608, the process 1600 may include processing the batch requests. In some examples, processing the requests may include identifying a storage device in a power down state that, when powered on, would be usable to service the request at 1610. Additionally, processing the requests may also include servicing the request when the group containing the identified storage device (e.g., from 1610) is modified according to the schedule to be in the power up state at 1612. That is, once the identified storage device is powered up or otherwise

enabled, the archival data storage service 206 may service the request (e.g., read data from the storage device or delete data from the storage device). Additionally, in some cases, the actions of 1608, 1610 and/or 1612 may be processed as a single action, two actions or three separate actions. In some examples, the process 1600 may end at 1612. However, in other example, the process 1600 may end with powering down a group of storage devices in the power up state. Alternatively, or in addition, the process 1600 may include powering down the group of storage devices in the power up state prior to powering up the group including the identified storage device and/or prior to servicing the request at 1614.

[0194] FIG. 17 illustrates another example flow diagram showing process 1700 for data storage power management. In some aspects, the storage node manager 244, the storage nodes 246, the storage node registrar store 250 and/or other data storage devices or collections of the archival data storage service 206 shown in FIG. 2 may perform the process 1700 of FIG. 17. The process 1700 may begin with reception of a rotation schedule, agenda or other type of timetable, at 1702, from the data storage node 246, the storage node registrar 248 or the storage node registrar store 250. As noted, the schedule may be performed by the data storage node 246. As noted above, the schedule may include instructions for which storage devices (or hard drives) are to be enabled or otherwise powered on and when. In some examples, the schedule may also include instructions for device controllers that may be configured to managing the power consumption of each storage device by slowing, stopping or speeding up disks and/or by disabling the storage device by removing an electrical signal. At 1704, the process 1700 may include managing power of one or more storage devices based at least in part on the schedule.

[0195] The process 1700 may also include identifying one or more data processing requests of a batch of requests associated with the storage devices at 1706. As noted above, the requests may be received in a single batch. Additionally, the requests may be processed in a batch at 1708. However, in some examples, the requests may be received in a batch and processed sequentially, yet no new requests may be received until the archival data storage service 206 has completed some or all of the requests of the batch. At 1710, the process 1700 may include identifying a data storage device in a low power mode that is able to service the request when powered up. Further, the process 1700 may end with service of the request once the identified storage device is powered up based on the schedule. In this way, the schedule may control the power management

of the storage devices, while at the same time, the batch of requests may influence the schedule. In this way, the data storage power management described herein may be considered on-demand. Additionally, in some aspects, the processing of the data processing requests at 1708 may include the actions of 1710 and/or 1712. As such, they may all be performed in a single action.

[0196] FIG. 18 illustrates another example flow diagram showing process 1800 for validating the integrity of data. In some aspects, the storage node manager 244, the storage nodes 246, the storage node registrar store 250 and/or other data storage devices or collections of the archival data storage service 206 shown in FIG. 2 may perform the process 1800 of FIG. 18. The process 1800 may begin, at 1802, with reception of a global time signal configured to enable management of the power consumption of one or more storage devices to be synchronized with access of the storage devices. That is the global time signal may be utilized by the archival data storage service 206 to accurately time when to manage power of each storage device. Additionally, the archival data storage service 206 may utilize the global time signal to determine when to access the data storage devices. As such, the two actions may, in some examples, by synchronized such that a storage device is accessed after its power consumption has been managed such that it is on, powered up or otherwise accessible without significant delay. At 1804, the process 1800 may include obtaining a schedule associated with enabling a first storage device at a different time than a second storage device, where each of the two storage device are part of a set of storage devices. In other words, the schedule may include information that indicates which storage devices of the set should be powered up and when including, but not limited to, when the first and second devices should be powered up. The process 1800 may also include identifying a request of a batch of requests for accessing the set of storage devices at 1806. As noted above, the requests may include read requests, write requests, delete requests, index list requests, or the like. At 1808, the process 1800 may include determining which storage device to enable based at least in part on the schedule. The determination may include selecting the first storage device, the second storage device or any other storage device of the set of storage devices being controlled or managed by the archival data storage service 206. Further, at 1810, the process 1800 may end with the management of power consumption of the determined storage device based at least in part on the determination from 1808.

[0197] Illustrative methods and systems for validating the integrity of data are described above. Some or all of these systems and methods may, but need not, be implemented at least partially by architectures such as those shown above.

[0198] FIG. 19 illustrates aspects of an example environment 1900 for implementing aspects in accordance with various embodiments. As will be appreciated, although a Web-based environment is used for purposes of explanation, different environments may be used, as appropriate, to implement various embodiments. The environment includes an electronic client device 1902, which can include any appropriate device operable to send and receive requests, messages or information over an appropriate network 1904 and convey information back to a user of the device. Examples of such client devices include personal computers, cell phones, handheld messaging devices, laptop computers, set-top boxes, personal data assistants, electronic book readers and the like. The network can include any appropriate network, including an intranet, the Internet, a cellular network, a local area network or any other such network or combination thereof. Components used for such a system can depend at least in part upon the type of network and/or environment selected. Protocols and components for communicating via such a network are well known and will not be discussed herein in detail. Communication over the network can be enabled by wired or wireless connections and combinations thereof. In this example, the network includes the Internet, as the environment includes a Web server 1906 for receiving requests and serving content in response thereto, although for other networks an alternative device serving a similar purpose could be used as would be apparent to one of ordinary skill in the art.

[0199] The illustrative environment includes at least one application server 1908 and a data store 1910. It should be understood that there can be several application servers, layers, or other elements, processes or components, which may be chained or otherwise configured, which can interact to perform tasks such as obtaining data from an appropriate data store. As used herein the term "data store" refers to any device or combination of devices capable of storing, accessing and retrieving data, which may include any combination and number of data servers, databases, data storage devices and data storage media, in any standard, distributed or clustered environment. The application server can include any appropriate hardware and software for

integrating with the data store as needed to execute aspects of one or more applications for the client device, handling a majority of the data access and business logic for an application. The application server provides access control services in cooperation with the data store, and is able to generate content such as text, graphics, audio and/or video to be transferred to the user, which may be served to the user by the Web server in the form of HTML, XML or another appropriate structured language in this example. The handling of all requests and responses, as well as the delivery of content between the client device 1902 and the application server 1908, can be handled by the Web server. It should be understood that the Web and application servers are not required and are merely example components, as structured code discussed herein can be executed on any appropriate device or host machine as discussed elsewhere herein.

[0200] The data store 1910 can include several separate data tables, databases or other data storage mechanisms and media for storing data relating to a particular aspect. For example, the data store illustrated includes mechanisms for storing production data 1912 and user information 1916, which can be used to serve content for the production side. The data store also is shown to include a mechanism for storing log data 1914, which can be used for reporting, analysis or other such purposes. It should be understood that there can be many other aspects that may need to be stored in the data store, such as for page image information and to access right information, which can be stored in any of the above listed mechanisms as appropriate or in additional mechanisms in the data store 1910. The data store 1910 is operable, through logic associated therewith, to receive instructions from the application server 1908 and obtain, update or otherwise process data in response thereto. In one example, a user might submit a search request for a certain type of item. In this case, the data store might access the user information to verify the identity of the user, and can access the catalog detail information to obtain information about items of that type. The information then can be returned to the user, such as in a results listing on a Web page that the user is able to view via a browser on the user device 1902. Information for a particular item of interest can be viewed in a dedicated page or window of the browser.

[0201] Each server typically will include an operating system that provides executable program instructions for the general administration and operation of that server, and typically will include a computer-readable storage medium (e.g., a hard disk, random access memory, read

only memory, etc.) storing instructions that, when executed by a processor of the server, allow the server to perform its intended functions. Suitable implementations for the operating system and general functionality of the servers are known or commercially available, and are readily implemented by persons having ordinary skill in the art, particularly in light of the disclosure herein.

[0202] The environment in one embodiment is a distributed computing environment utilizing several computer systems and components that are interconnected via communication links, using one or more computer networks or direct connections. However, it will be appreciated by those of ordinary skill in the art that such a system could operate equally well in a system having fewer or a greater number of components than are illustrated in FIG. 19. Thus, the depiction of the system 1900 in FIG. 19 should be taken as being illustrative in nature, and not limiting to the scope of the disclosure.

[0203] Example embodiments of the disclosure can be described in view of the following clauses:

1. A computer-implemented method for providing cost-effective and durable archival data storage service, comprising:

under the control of one or more computer systems of an archival data storage system that are configured with executable instructions,

receiving, over a network from a requestor system, a storage request to store a data object into the archival data storage system;

causing storage of the data object in the archival data storage system;

providing a data object identifier associated with data object;

receiving, in connection with a retrieval request to retrieve the data object, the data object identifier;

creating a retrieval job corresponding to the retrieval request;

adding the retrieval job to a collection of pending jobs;

processing, in one or more batches, the collection of pending jobs; and

providing the retrieved data object.

2. The computer-implemented method of clause 1, further comprising creating a storage job corresponding to the storage request and adding the storage job the collection of pending jobs, and wherein causing storage of the data object comprises causing storage of the data object in a staging storage.

3. The computer-implemented method of clause 1, further comprising providing a retrieval job identifier associated with the retrieval job and wherein providing the retrieved data object includes transmitting the retrieved data object in one or more parts to a requestor system that specified the retrieval job identifier in a request for the data object.

4. The computer-implemented method of clause 1, further comprising providing a notification of completion of the retrieval job after the retrieval job is successfully completed.

5. The computer-implemented method of clause 2, wherein providing the retrieved the data object comprises retrieving the data object from the staging store.

6. The computer-implemented method of clause 1, further comprising validating integrity of the data object based at least in part on a digest of at least a portion of the data object.

7. A computer-implemented method comprising:

under the control of one or more computer systems configured with executable instructions,

receiving a data retrieval request to retrieve a data object, the data retrieval request specifying a data object identifier;

creating a data retrieval job corresponding to the data retrieval request;

providing a job identifier associated with the data retrieval job that is usable for obtaining information about the data retrieval job; and

after providing the job identifier, processing the data retrieval job using at least in part the data object identifier to provide access to the data object.

8. The computer-implemented method of clause 7, wherein the data object identifier is provided in response to a previous storage request to store the data object.

9. The computer-implemented method of clause 7, wherein processing the data retrieval job comprises:

selecting the data retrieval job for execution;

determining, based at least in part on the data object identifier, one or more storage entities for storing one or more encoded data components, the one or more encoded data components are generated based at least in part on the data object;

causing retrieval of at least some of the one or more encoded data components from the determined; and

decoding the retrieved encoded data components to obtain the retrieved data object.

10. The computer-implemented method of clause 9, wherein selecting the data retrieval job is based at least in part on a batch processing schedule.

11. The computer-implemented method of clause 10, wherein the batch processing schedule is used to gain efficiency.

12. The computer-implemented method of clause 7, further comprising providing a status of the data retrieval job in response to a job status request that specifies the job identifier.

13. A system for providing archival data storage services, comprising:

one or more archival data storage devices;

a transient data store;

one or more processors;

memory, including executable instructions that, when executed by the one or more processors, cause the one or more processors to collectively at least:

receive a data storage request to store a data object;

cause storage of the data object in the transient store;

provide a data object identifier associated with the data, the data object identifier encoding at least storage location information usable to locate the data object; and

after providing the data object identifier, cause storage of the data object in location specified by the storage location information.

14. The system of clause 13, wherein the executable instructions, when executed by the one or more processors, further cause the one or more processors to collectively create a data storage job corresponding to the data storage request and wherein causing storage of the data object in location specified by the storage location information comprises processing the data storage job based at least in part on the storage location information.

15. The system of clause 14, wherein the processing the data storage job comprises scheduling the job for execution based at least in part on a batch processing schedule.

16. The system of clause 13, wherein causing storage of the data object in location specified by the storage location information comprises:

obtaining the data object from the transient store;

obtaining one or more data encoding schemes;

encoding the data object with the one or more encoding schemes to obtain a plurality of encoded data components; and

causing storage of the plurality of encoded data components in at least some of the one or more archival data storage devices.

17. The system of clause 13, wherein the data object identifier encodes at least data validation information usable to validate integrity of the data object.

18. The system of clause 17, wherein the executable instructions, when executed by the one or more processors, further cause the one or more processors to collectively validate integrity of the data object based at least in part on the data validation information.

19. One or more non-transitory computer-readable storage media having collectively stored thereon executable instructions that, when executed by one or more processors of an archival data storage system, cause the system to at least:

receive a plurality of data retrieval requests, each of the plurality of data retrieval request specifying a data object identifier for a data object to be retrieved;

cause data retrieval jobs to be created, each corresponding to a received data retrieval request;

cause job identifiers to be provided, each job identifier corresponding to a data retrieval job and being usable to obtaining information about the data retrieval job;

cause aggregation of at least a subset of the data retrieval jobs to form a job batch; and

cause processing of the job batch corresponding to the subset of data retrieval jobs after causing the job identifiers to be provided.

20. The one or more computer-readable storage media of clause 19, wherein the instructions further cause the system to cause aggregation of at least a subset of the plurality of data retrieval requests to form a request batch and cause validation of the request batch based at least in part on data object identifiers corresponding to the subset of data retrieval requests.

21. The one or more computer-readable storage media of clause 19, wherein the instructions further cause the system to cause processing of at least some of the data retrieval jobs based at least in part on a power management schedule.

22. The one or more computer-readable storage media of clause 19, wherein the wherein the instructions further cause the system to cause partitioning of the data retrieval jobs to facilitate parallel processing.

23. The one or more computer-readable storage media of clause 19, wherein the wherein the instructions further cause the system to provide access to at least some of the data objects corresponding to the plurality of data retrieval requests.

24. The one or more computer-readable storage media of clause 23, wherein the wherein the instructions further cause the system to cause validation of data integrity for at least some of the data objects corresponding to the plurality of data retrieval requests.

25. A computer-implemented method for data integrity verification, comprising:

under control of one or more computer systems configured with executable instructions,

receiving, in connection with a request to an archival data storage service, a plurality of portions of a data payload from a remote computing device;

for each portion of at least a subset of the plurality of portions, generating one or more sub-portions of a predefined size;

calculating one or more digests based at least in part on the one or more sub-portions;

generating at least a root node of a hierarchical data structure with nodes representing at least the one or more digests;

determining a top-level digest of the hierarchical data structure associated with a root node of the hierarchical data structure;

receiving a top-level digest generated by the remote computing device corresponding to the data payload; and

verifying, based at least in part on comparing the received top-level digest and the determined top-level digest, that a stored data payload matches a received data payload corresponding to the received plurality of portions.

26. The computer-implemented method of clause 25, further comprising providing an instruction to the remote computing device, the instruction configured to enable the remote computing device to generate the hierarchical data structure and provide the received top-level digest.

27. The computer-implemented method of clause 25, wherein the plurality of portions of the data payload are received as part of an electronic request, from the remote computing device, to store the data payload.

28. The computer-implemented method of clause 27, further comprising providing a function call for the remote device to make the electronic request, and wherein the electronic request includes at least the provided function call.

29. The computer-implemented method of clause 25, wherein the hierarchical data structure includes at least a binary tree.

30. The computer-implemented method of clause 25, further comprising receiving an indication of a size for each of the plurality of portions.

31. A computer-implemented method for data integrity verification, comprising:

under control of one or more computer systems configured with executable instructions,

receiving one or more parts of a data payload;

for each part of at least a subset of the one or more parts, generating a sub-part;

calculating a value based at least in part on the generated sub-part;

generating at least a root node of a data structure based at least in part on the calculated value; and

determining a top-level value of the data structure associated with the root node of the data structure.

32. The computer-implemented method of clause 31, further comprising:

storing the data payload in archival data storage; and

validating, based at least in part on the determined top-level value and a received top-level value, that the stored data payload matches a received data payload corresponding to the received one or more parts.

33. The computer-implemented method of clause 31, wherein the sub-part is generated to be a first predefined size, and wherein the received one or more parts are a second size that is an integer multiple of the first predefined size.

34. The computer-implemented method of clause 33, wherein the integer is based at least in part on a degree of branches of the data structure.

35. The computer-implemented method of clause 31, further comprising providing a data object identifier for identifying the data payload, the data object identifier including the top-level value of the data structure.

36. The computer-implemented method of clause 35, further comprising:

receiving a request for the stored data payload; and

verifying, based at least in part on the top-level value, that the stored data payload has not changed prior to providing the data payload.

37. The computer-implemented method of clause 31, further comprising:

storing the data payload of a first storage device to second storage device; and

verifying, based at least in part on the top-level value, that the data payload stored in the second storage device matches the data payload of the first storage device.

38. The computer-implemented method of clause 31, further comprising:

repartitioning the data payload based at least in part on a second generated sub-part, the second generated sub-part being different form the generated sub-part; and

verifying, based at least in part on the top-level value, that the repartitioned data payload matches a stored data payload, the stored data payload based at least in part on the received data payload.

39. A system for verifying data integrity, comprising:

at least one memory that stores computer-executable instructions; and

at least one processor configured to access the at least one memory, wherein the at least one processor is configured to execute the computer-executable instructions to collectively at least:

store at least a chunk of data, the data comprising a plurality of chunks;

generate at least one sub-chunk of the at least one chunk of data;

store at least a digest corresponding to the sub-chunk;

generate a data structure based at least in part on the stored digest corresponding to the sub-chunk; and

provide a top-level digest of the generated data structure to a computing device.

40. The system of clause 39, wherein the at least one processor is further configured to execute the computer-executable instructions to receive the at least a chunk of data from the computing device based at least in part on a predefined size indicated by the computing device.

41. The system of clause 40, wherein the at least a chunk of data is received in an electronic request, from the computing device, to store the data based at least in part on an instruction format provided by the system.

42. The system of clause 39, wherein the top-level digest is prepended to an identifier configured to identify the data.

43. The system of clause 42, wherein the at least one processor is further configured to execute the computer-executable instructions to:

store the data comprising the plurality of chunks;

receive a request to provide the data to the computing device, the request including the identifier; and

validate the data, based at least in part on the top-level digest, prior to providing the data to the computing device.

44. The system of clause 42, wherein the at least one processor is further configured to execute the computer-executable instructions to:

generate a digest for the identifier; and

validate the identifier, based at least in part on the generated digest for the identifier, prior to referencing the identifier in an instruction.

45. One or more computer-readable media storing computer-executable instructions for verifying data integrity that, when executed by one or more processors, configure the one or more processors to perform operations comprising:

performing a first data operation in connection with a data object, the first data operation based at least in part on a first partitioning of the data object into first partitions;

verifying the data object using a data verification algorithm to generate a first verification value based at least in part on the first partitions;

performing a second data operation in connection with the data object, the second data operation based at least in part on a second partitioning of the data object into second partitions, the second partitions being different from the first partitions;

verifying the data object using the data verification algorithm to generate a second verification value; and

determining whether the second verification value matches the first verification value.

46. The one or more computer-readable media of clause 45, wherein using the data verification algorithm includes generating one or more partitions of the data in a predetermined size.

47. The one or more computer-readable media of clause 46, wherein at least one of the first partitions or at least one of the second partitions has a size that is an a multiple of the predetermined size, the multiple being a result of integer exponentiation of a degree of a data structure associated with the first data object.

48. The one or more computer-readable media of clause 45, wherein the instructions further configure the one or more processors to perform operations comprising providing instructions for making method calls to perform at least one of the first data operation or the second data operation.

49. The one or more computer-readable media of clause 45, wherein at least one of the first data operation or the second data operation is a transfer of the data object over a network.

50. The one or more computer-readable media of clause 49, wherein the object is transferred over the network in parts.

51. The one or more computer-readable media of clause 45, wherein the first operation includes at least receiving the data object and the second operation includes at least storing the data object in archival data storage.

52. The one or more computer-readable media of clause 51, wherein the instructions further configure the one or more processors to perform operations comprising ensuring integrity of the data object based at least in part on the second verification value.

53. The one or more computer-readable media of clause 51, wherein the instructions further configure the one or more processors to perform operations comprising ensuring integrity of the data object based at least in part on the second partitions.

54. A computer-implemented method for managing storage device power, comprising:

under control of one or more computer systems configured with executable instructions,

classifying data storage devices of an archive data storage system into groups, each group comprising a subset of the data storage devices;

modifying a power state of the groups according to a schedule, the schedule being configured to ensure that, on average, a predetermined amount of the data storage devices are in a power down state;

obtaining a batch of data processing requests; and

processing the batch of data processing requests by at least, for each request of at least a subset of the requests:

identifying a data storage device in the power down state that, when in a power up state, is usable to service the request; and

servicing the request when a group containing the identified data storage device is modified according to the schedule to be in the power up state.

55. The computer-implemented method of clause 54, wherein the groups indicate at least an amount of the data storage devices of the subset.

56. The computer-implemented method of clause 54, wherein modifying the power state of the groups includes at least providing an instruction to a storage device controller configured to at least one of power up or power down the data storage devices of the groups.

57. The computer-implemented method of clause 54, wherein the batch of data processing requests includes at least one of a request to store data, a request to delete data, or a request to read data.

58. The computer-implemented method of clause 54, wherein servicing the request includes at least reading or deleting data from the identified data storage device.

59. The computer-implemented method of clause 54, further comprising powering down a storage device in of a group in the power up state prior to the group containing the identified data storage device being modified to be in the power up state.

60. A computer-implemented method for managing storage device power, comprising:

under control of one or more computer systems configured with executable instructions,

managing power of one or more data storage devices based at least in part on a schedule;

identifying a data processing request of a batch, the request associated with the one or more data storage devices; and

processing the data processing request by at least:

identifying a data storage device in a lower power state that is able to service the request when in a higher power state; and

servicing the request when the identified data storage device is powered up based at least in part on the schedule.

61. The computer-implemented method of clause 60, wherein the schedule is configured to keep a predetermined amount of the data storage devices in the lower power state.

62. The computer-implemented method of clause 60, further comprising implementing the schedule by a data storage node configured to perform data processing jobs associated with the data storage devices.

63. The computer-implemented method of clause 60, further comprising receiving the schedule from a storage node registrar configured to manage data processing jobs associated with the data storage devices.

64. The computer-implemented method of clause 60, wherein the data processing requests are processed based at least in part on the batch.

65. The computer-implemented method of clause 60, wherein the data processing request is at least one of a read request, a write request, or a delete request.

66. A system for managing storage device power, comprising:

at least one memory that stores computer-executable instructions; and

at least one processor configured to access the at least one memory, wherein the at least one processor is configured to execute the computer-executable instructions to collectively at least:

store information associated with a schedule for managing power of one or more storage devices;

identify a request associated with the one or more storage devices, the request included in a batch of requests;

process the request by executing additional computer-executable instructions to at least:

determine a temporarily deactivated storage device capable of servicing the request; and

service the request when the determined storage device is activated based at least in part on the schedule.

67. The system of clause 66, wherein the schedule indicates at least an amount of the one or more storage devices to be temporarily deactivated concurrently.

68. The system of clause 66, wherein the request is processed with other requests of the batch.

69. The system of clause 66, wherein the schedule is configured to at least one of minimize a cost associated with the one or more storage devices, minimize power consumption of the one or more storage devices, or maximize an operational life of the one or more storage devices.

70. The system of clause 66, wherein the schedule is received from a component configured to manage requests associated with the one or more storage devices.

71. The system of clause 66, wherein servicing the request includes at least modifying the schedule based at least in part on a request to access the one or more storage devices.

72. One or more computer-readable media storing computer-executable instructions for managing storage device power that, when executed by one or more processors, configure the one or more processors to perform operations comprising:

obtaining a schedule associated with enabling at least a first storage device of one or more storage devices at a different time than at least a second storage device of the one or more storage devices;

identifying a request of a batch of requests for accessing the one or more storage devices;

determining, based at least in part on the schedule and the request, which of at least the first storage device or the second storage device to activate; and

managing power consumption of at least the first or second storage device of the one or more storage devices based at least in part on the determination.

73. The one or more computer-readable media of clause 72, wherein the instructions further configure the one or more processors to perform operations comprising receiving a global time signal configured to enable management of the power consumption of the one or more storage devices to be synchronized with at least access of the one or more storage devices.

74. The one or more computer-readable media of clause 73, wherein managing the power consumption includes at least powering off or powering on at least the first or second storage device based at least in part on the global time signal.

75. The one or more computer-readable media of clause 73, wherein accessing the one or more storage devices includes at least reading data from, writing data to, or deleting data of at least the first or second storage device based at least in part on the global time signal.

76. The one or more computer-readable media of clause 72, wherein the schedule indicates at least one of a number of storage devices to be operational at a same time or a duration of operation of each operational storage device.

77. The one or more computer-readable media of clause 72, wherein the rotation schedule ensures an equal distribution of temperature or vibration associated with the one or more storage device.

78. The one or more computer-readable media of clause 72, wherein determining which of the at least the first storage device or the second storage device to activate is based at least in part on at least one of heat avoidance or vibration avoidance.

79. The one or more computer-readable media of clause 72, wherein the rotation schedule, for a particular server, groups the one or more storage devices of the particular server to activate or deactivate the one or more storage devices of the particular server based at least in part on a particular distribution.

[0204] The various embodiments further can be implemented in a wide variety of operating environments, which in some cases can include one or more user computers, computing devices or processing devices which can be used to operate any of a number of applications. User or client devices can include any of a number of general purpose personal computers, such as desktop or laptop computers running a standard operating system, as well as cellular, wireless and handheld devices running mobile software and capable of supporting a number of networking and messaging protocols. Such a system also can include a number of workstations running any of a variety of commercially-available operating systems and other known applications for purposes such as development and database management. These devices also can include other electronic devices, such as dummy terminals, thin-clients, gaming systems and other devices capable of communicating via a network.

[0205] Most embodiments utilize at least one network that would be familiar to those skilled in the art for supporting communications using any of a variety of commercially-available protocols, such as TCP/IP, OSI, FTP, UPnP, NFS, CIFS and AppleTalk. The network can be, for example, a local area network, a wide-area network, a virtual private network, the Internet, an intranet, an extranet, a public switched telephone network, an infrared network, a wireless network and any combination thereof.

[0206] In embodiments utilizing a Web server, the Web server can run any of a variety of server or mid-tier applications, including HTTP servers, FTP servers, CGI servers, data servers, Java servers and business application servers. The server(s) also may be capable of executing programs or scripts in response requests from user devices, such as by executing one or more Web applications that may be implemented as one or more scripts or programs written in any programming language, such as Java®, C, C# or C++, or any scripting language, such as Perl, Python or TCL, as well as combinations thereof. The server(s) may also include database servers, including without limitation those commercially available from Oracle®, Microsoft®, Sybase® and IBM®.

[0207] The environment can include a variety of data stores and other memory and storage media as discussed above. These can reside in a variety of locations, such as on a storage medium local to (and/or resident in) one or more of the computers or remote from any or all of

the computers across the network. In a particular set of embodiments, the information may reside in a storage-area network ("SAN") familiar to those skilled in the art. Similarly, any necessary files for performing the functions attributed to the computers, servers or other network devices may be stored locally and/or remotely, as appropriate. Where a system includes computerized devices, each such device can include hardware elements that may be electrically coupled via a bus, the elements including, for example, at least one central processing unit (CPU), at least one input device (e.g., a mouse, keyboard, controller, touch screen or keypad), and at least one output device (e.g., a display device, printer or speaker). Such a system may also include one or more storage devices, such as disk drives, optical storage devices, and solid-state storage devices such as random access memory ("RAM") or read-only memory ("ROM"), as well as removable media devices, memory cards, flash cards, etc.

[0208] Such devices also can include a computer-readable storage media reader, a communications device (e.g., a modem, a network card (wireless or wired), an infrared communication device, etc.) and working memory as described above. The computer-readable storage media reader can be connected with, or configured to receive, a computer-readable storage medium, representing remote, local, fixed and/or removable storage devices as well as storage media for temporarily and/or more permanently containing, storing, transmitting and retrieving computer-readable information. The system and various devices also typically will include a number of software applications, modules, services or other elements located within at least one working memory device, including an operating system and application programs, such as a client application or Web browser. It should be appreciated that alternate embodiments may have numerous variations from that described above. For example, customized hardware might also be used and/or particular elements might be implemented in hardware, software (including portable software, such as applets) or both. Further, connection to other computing devices such as network input/output devices may be employed.

[0209] Storage media and computer readable media for containing code, or portions of code, can include any appropriate media known or used in the art, including storage media and communication media, such as but not limited to volatile and non-volatile, removable and non-removable media implemented in any method or technology for storage and/or transmission of

information such as computer readable instructions, data structures, program modules or other data, including RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disk (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices or any other medium which can be used to store the desired information and which can be accessed by the a system device. Based on the disclosure and teachings provided herein, a person of ordinary skill in the art will appreciate other ways and/or methods to implement the various embodiments.

[0210] The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense. It will, however, be evident that various modifications and changes may be made thereunto without departing from the broader spirit and scope of the invention as set forth in the claims.

[0211] Other variations are within the spirit of the present disclosure. Thus, while the disclosed techniques are susceptible to various modifications and alternative constructions, certain illustrated embodiments thereof are shown in the drawings and have been described above in detail. It should be understood, however, that there is no intention to limit the invention to the specific form or forms disclosed, but on the contrary, the intention is to cover all modifications, alternative constructions and equivalents falling within the spirit and scope of the invention, as defined in the appended claims.

[0212] The use of the terms "a" and "an" and "the" and similar referents in the context of describing the disclosed embodiments (especially in the context of the following claims) are to be construed to cover both the singular and the plural, unless otherwise indicated herein or clearly contradicted by context. The terms "comprising," "having," "including," and "containing" are to be construed as open-ended terms (i.e., meaning "including, but not limited to,") unless otherwise noted. The term "connected" is to be construed as partly or wholly contained within, attached to, or joined together, even if there is something intervening. Recitation of ranges of values herein are merely intended to serve as a shorthand method of referring individually to each separate value falling within the range, unless otherwise indicated herein, and each separate value is incorporated into the specification as if it were individually recited herein. All methods described herein can be performed in any suitable order unless

otherwise indicated herein or otherwise clearly contradicted by context. The use of any and all examples, or exemplary language (e.g., "such as") provided herein, is intended merely to better illuminate embodiments of the invention and does not pose a limitation on the scope of the invention unless otherwise claimed. No language in the specification should be construed as indicating any non-claimed element as essential to the practice of the invention.

[0213] Preferred embodiments of this disclosure are described herein, including the best mode known to the inventors for carrying out the invention. Variations of those preferred embodiments may become apparent to those of ordinary skill in the art upon reading the foregoing description. The inventors expect skilled artisans to employ such variations as appropriate, and the inventors intend for the invention to be practiced otherwise than as specifically described herein. Accordingly, this invention includes all modifications and equivalents of the subject matter recited in the claims appended hereto as permitted by applicable law. Moreover, any combination of the above-described elements in all possible variations thereof is encompassed by the invention unless otherwise indicated herein or otherwise clearly contradicted by context.

[0214] All references, including publications, patent applications and patents, cited herein are hereby incorporated by reference to the same extent as if each reference were individually and specifically indicated to be incorporated by reference and were set forth in its entirety herein.

CLAIMS

WHAT IS CLAIMED IS:

1. A computer-implemented method comprising:

under the control of one or more computer systems configured with executable instructions,

receiving a data retrieval request to retrieve a data object, the data retrieval request specifying a data object identifier;

creating a data retrieval job corresponding to the data retrieval request;

providing a job identifier associated with the data retrieval job that is usable for obtaining information about the data retrieval job; and

after providing the job identifier, processing the data retrieval job using at least in part the data object identifier to provide access to the data object.

2. The computer-implemented method of claim 1, wherein the data object identifier is provided in response to a previous storage request to store the data object.

3. The computer-implemented method of claim 1, wherein processing the data retrieval job comprises:

selecting the data retrieval job for execution;

determining, based at least in part on the data object identifier, one or more storage entities for storing one or more encoded data components, the one or more encoded data components are generated based at least in part on the data object;

causing retrieval of at least some of the one or more encoded data components from the determined; and

decoding the retrieved encoded data components to obtain the retrieved data object.

4. The computer-implemented method of claim 3, wherein selecting the data retrieval job is based at least in part on a batch processing schedule.

5. The computer-implemented method of claim 1, further comprising providing a status of the data retrieval job in response to a job status request that specifies the job identifier.

6. A system for providing archival data storage services, comprising:

one or more archival data storage devices;

a transient data store;

one or more processors;

memory, including executable instructions that, when executed by the one or more processors, cause the one or more processors to collectively at least:

receive a data storage request to store a data object;

cause storage of the data object in the transient store;

provide a data object identifier associated with the data, the data object identifier encoding at least storage location information usable to locate the data object; and

after providing the data object identifier, cause storage of the data object in location specified by the storage location information.

7. The system of claim 6, wherein the executable instructions, when executed by the one or more processors, further cause the one or more processors to collectively create a data storage job corresponding to the data storage request and wherein causing storage of the data object in location specified by the storage location information comprises processing the data storage job based at least in part on the storage location information.

8. The system of claim 6, wherein causing storage of the data object in location specified by the storage location information comprises:

obtaining the data object from the transient store;

obtaining one or more data encoding schemes;

encoding the data object with the one or more encoding schemes to obtain a plurality of encoded data components; and

causing storage of the plurality of encoded data components in at least some of the one or more archival data storage devices.

9. The system of claim 6, wherein the data object identifier encodes at least data validation information usable to validate integrity of the data object.

10. The system of claim 9, wherein the executable instructions, when executed by the one or more processors, further cause the one or more processors to collectively validate integrity of the data object based at least in part on the data validation information

11. The system of claim 6, wherein the executable instructions, when executed by the one or more processors, further cause the one or more processors to:

receive a data retrieval request for the data object cause aggregation the data retrieval request with other data retrieval requests to form a request batch; and

cause validation of the request batch based at least in part on data object identifiers corresponding to the request batch.

12. The system of claim 6, wherein the executable instructions, when executed by the one or more processors, further cause the one or more processors to receive a data retrieval request for the data object; and

cause processing of the data retrieval request based at least in part on a power management schedule.

13. The system of claim 6, wherein the executable instructions, when executed by the one or more processors, further cause the one or more processors to cause partitioning of pending data retrieval requests to facilitate parallel processing.

14. The system of claim 6, wherein the executable instructions, when executed by the one or more processors, further cause the one or more processors to provide status of a data retrieval request in response to a status request that specifies the data object identifier.

15. The system of claim 6, wherein the executable instructions, when executed by the one or more processors, further cause the one or more processors to cause validation of data integrity for the data object.