(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
9 August 2012 (09.08.2012)

WIPO | PCT

(10) International Publication Number
WO 2012/106362 A2

(51) International Patent Classification:
G06F 12/00 (2006.01) *G06F 12/08* (2006.01)

(21) International Application Number:
PCT/US2012/023373

(22) International Filing Date:
31 January 2012 (31.01.2012)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/438,182 31 January 2011 (31.01.2011) US
61/438,194 31 January 2011 (31.01.2011) US

(71) Applicant *(for all designated States except US)*: **FUSION-IO, INC.** [US/US]; 2855 E. Cottonwood Parkway Box 100, Salt Lake City, Utah 84121 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **NELLANS, David** [US/US]; 1120 S. 1300 E., Salt Lake City, Utah 84105 (US). **ATKISSON, David** [US/US]; 13011 South Benchview Cove, Draper, Utah 84030 (US). **PETERSON, Jim** [US/US]; P.O. Box 721026, San Jose, California 95172 (US). **GARFF, Jeremy** [US/US]; 2144 E. Oak Manor Drive, Sandy, Utah 84092 (US). **ZAPPE, Mike** [US/US]; 8365 Quay Drive, Arvada, Colorado 80003 (US).

(74) Agents: **HILTON, Scott** et al.; 8 East Broadway Suite 600, Salt Lake City, Utah 84111 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

[Continued on next page]

(54) Title: APPARATUS, SYSTEM, AND METHOD FOR MANAGING EVICTION OF DATA

(57) Abstract: An apparatus, system, and method are disclosed for managing eviction of data. A cache write module 712 stores data on a non-volatile storage device 102 sequentially using a log-based storage structure 122 having a head region 128 and a tail region 124. A direct cache module 1016 caches data on the non-volatile storage device 102 using the log-based storage structure 122. The data is associated with storage operations between a host 114 and a backing store storage device 118. An eviction module 1014 evicts data of at least one region in succession from the log-based storage structure 122 starting with the tail region 124 and progressing toward the head region 128.

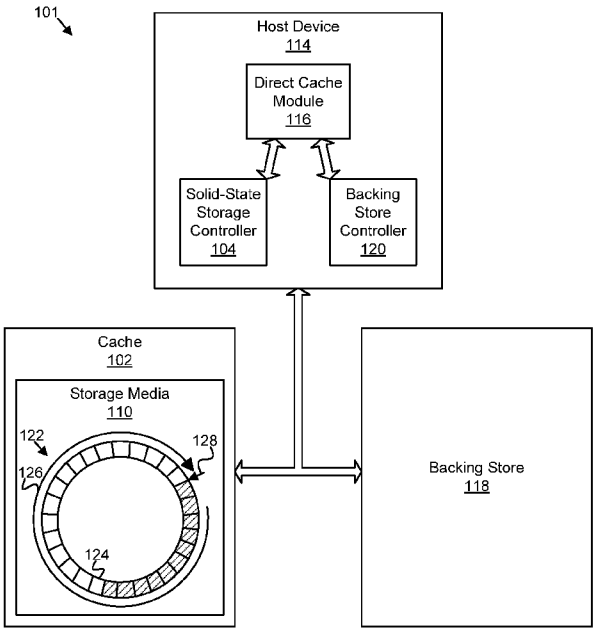


FIG. 1B

WO 2012/106362 A2

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *without international search report and to be republished upon receipt of that report (Rule 48.2(g))*

APPARATUS, SYSTEM, AND METHOD FOR MANAGING EVICTION OF DATA

FIELD OF THE INVENTION

The subject matter disclosed herein relates to caching data and more particularly relates to managing eviction of data from a cache.

DESCRIPTION OF THE RELATED ART

A cache device typically has a larger storage capacity than the backing store with which the cache device is associated. As a cache device fills with cached data, certain cached data may be evicted to free up room to cache additional data. If data is evicted from a cache device, a subsequent request for the evicted data yields a cache miss. Evicting the wrong data from a cache device can increase the number of cache misses and decrease the efficiency of the cache device.

SUMMARY OF THE INVENTION

The present invention has been developed in response to the present state of the art, and in particular, in response to the problems and needs in the art that have not yet been fully solved by currently available data caches. Accordingly, the present invention has been developed to provide an apparatus, system, and method for managing eviction of data that overcome many or all of the above-discussed shortcomings in the art.

A method of the present invention is presented for managing eviction of data. The method in the disclosed embodiments substantially includes the steps necessary to carry out the functions presented below with respect to the operation of the described apparatus and system. In one embodiment, the method includes storing data on non-volatile storage media sequentially using a log-based storage structure having a head region and a tail region.. In another embodiment, the method includes caching data on the non-volatile storage media using the log-based storage structure. The data, in a further embodiment, is associated with storage operations between a host and a backing store storage device. The method, in one embodiment, includes evicting data of at least one region in succession from the log-based storage structure starting with the tail region and progressing toward the head region.

In another embodiment, the method includes evicting regions from the log-based storage structure in a first-in-first-out order. The first-in-first-out order, in certain embodiments, comprises an order that regions of the log-based storage structure are added to the log-based storage structure. The method, in a further embodiment, includes designating a neighboring region to a present tail region of the log-based storage structure as tail region of the log-based storage structure in response to successfully evicting data from the present tail region.

The method, in one embodiment, includes evicting the data of the at least one region in response to a storage capacity recovery event. In another embodiment, the storage capacity recovery event comprises a number of regions in an available storage pool satisfying a threshold. In various other embodiments, the storage capacity recovery event may comprise one or more of an amount of available storage capacity for the non-volatile storage media falling below a threshold, an eviction request from a cache client, the non-volatile storage media satisfying an error condition, or the like.

In one embodiment, the method includes clearing data from a present tail region of the log-based storage structure during a garbage collection operation. The method, in another embodiment, includes designating a neighboring region to the present tail region of the log-based storage structure as tail region of the log-based storage structure in response to the present tail region comprising exclusively invalid data.

The method, in one embodiment, includes selectively copying valid, clean data out of the at least one region to retain the valid, clean data in response to the valid, clean data satisfying a data use metric. In a further embodiment, the method includes destaging dirty write data from the at least one storage region to the backing store prior to evicting data of the at least one region. In another embodiment, the method includes copying dirty write data from the at least one region forward on the log-based storage structure in response to evicting data of the at least one region.

In one embodiment, the method includes evicting a plurality of regions in order. The order, in certain embodiments, is defined by regions in the log-based storage structure, moving from the tail of the log towards the head of the log. The non-volatile storage media, in one embodiment, caches data according to a write through cache policy.

An apparatus and system for managing eviction of data is provided with a plurality of modules configured to functionally execute the necessary steps of evicting data from a non-volatile storage device as described above with regard to the disclosed method. These modules in the described embodiments include a cache write module, a direct cache module, an eviction module, a frequent data module, and a destage module.

The system may be embodied by a processor, a storage controller, and a cache controller. In particular, the system, in one embodiment, includes an eviction module of the cache controller. In another embodiment, the system includes a host computer system comprising the processor. The storage controller and the cache controller, in certain embodiments, each comprise a device driver executing on the processor of the host computer system

References throughout this specification to features, advantages, or similar language do not imply that all of the features and advantages may be realized in any single embodiment.

Rather, language referring to the features and advantages is understood to mean that a specific feature, advantage, or characteristic is included in at least one embodiment. Thus, discussion of the features and advantages, and similar language, throughout this specification may, but do not necessarily, refer to the same embodiment.

These features and advantages of the embodiments will become more fully apparent from the following description and appended claims, or may be learned by the practice of embodiments as set forth hereinafter.

BRIEF DESCRIPTION OF THE DRAWINGS

In order that the advantages of the invention will be readily understood, a more particular description of the invention briefly described above will be rendered by reference to specific embodiments that are illustrated in the appended drawings. Understanding that these drawings depict only typical embodiments of the invention and are not therefore to be considered to be limiting of its scope, the invention will be described and explained with additional specificity and detail through the use of the accompanying drawings, in which:

Figure 1A is a schematic block diagram illustrating one embodiment of a system for managing eviction of data in accordance with the present invention;

Figure 1B is a schematic block diagram illustrating another embodiment of a system for managing eviction of data in accordance with the present invention;

Figure 2 is a schematic block diagram illustrating one embodiment of a host device in accordance with the present invention;

Figure 3 is a schematic block diagram illustrating one embodiment of a direct cache module in accordance with the present invention;

Figure 4 is a schematic block diagram illustrating another embodiment of a direct cache module in accordance with the present invention;

Figure 5A is a schematic block diagram illustrating one embodiment of a circular grooming data structure and a grooming candidate set in accordance with the present invention;

Figure 5B is a schematic block diagram illustrating one embodiment of a region selection and a grooming candidate set in accordance with the present invention;

Figure 5C is a schematic block diagram illustrating another embodiment of a region selection and a grooming candidate set in accordance with the present invention;

Figure 5D is a schematic block diagram illustrating a further embodiment of a region selection and a grooming candidate set in accordance with the present invention;

Figure 5E is a schematic block diagram illustrating an additional embodiment of a region selection and a grooming candidate set in accordance with the present invention;

Figure 5F is a schematic block diagram illustrating another embodiment of a region selection and a grooming candidate set in accordance with the present invention;

Figure 5G is a schematic block diagram illustrating one more embodiment of a region selection and a grooming candidate set in accordance with the present invention;

Figure 6 is a schematic block diagram illustrating one embodiment of a mapping structure, a logical address space of a cache, a sequential, log-based, append-only writing structure, and an address space of a storage device in accordance with the present invention;

Figure 7 is a schematic block diagram illustrating one embodiment of a direct cache module in accordance with the present invention;

Figure 8 is a schematic flow chart diagram illustrating one embodiment of a method for managing eviction of data in accordance with the present invention; and

Figure 9 is a schematic flow chart diagram illustrating another method for managing eviction of data in accordance with the present invention.

DETAILED DESCRIPTION OF THE INVENTION

As will be appreciated by one skilled in the art, aspects of the present invention may be embodied as a system, method or computer program product. Accordingly, aspects of the present invention may take the form of an entirely hardware embodiment, an entirely software embodiment (including firmware, resident software, micro-code, etc.) or an embodiment combining software and hardware aspects that may all generally be referred to herein as a "circuit," "module" or "system." Furthermore, aspects of the present invention may take the form of a computer program product embodied in one or more computer readable medium(s) having computer readable program code embodied thereon.

CACHING SYSTEM

Figure 1A depicts one embodiment of a system 100 for managing eviction of data from a cache 102 in accordance with the present invention. The system 100, in the depicted embodiment, includes a cache 102 a host device 114, a direct cache module 116, and a backing store 118. The cache 102, in the depicted embodiment, includes a solid-state storage controller 104, a write data pipeline 106, a read data pipeline 108, and a solid-state storage media 110. In general, the system 100 caches data for the backing store 118 in the cache 102 and the direct cache module 116 grooms and selectively evicts cached data from the cache 102.

In the depicted embodiment, the system 100 includes a single cache 102. In another embodiment, the system 100 may include two or more caches 102. For example, in various embodiments, the system 100 may mirror cached data between several caches 102, may virtually stripe cached data across multiple caches 102, may include a hierarchy of multiple caches 102, or

otherwise cache data in more than one cache 102. In general, the cache 102 serves as a read and/or a write cache for the backing store 118 and the backing store 118 is a storage device that serves as a backing store for the cache 102. In one embodiment, the cache 102 operates in a write-back mode and the direct cache module 116 destages cached write data to the backing store 118 opportunistically after caching the write data in the cache 102. In certain embodiments, the cache 102 may operate, at least temporarily, in another mode, such as a write-through mode, a write-around mode, a read-only mode, a bypass mode, or the like, and the direct cache module 116 may write data to the backing store 118 substantially simultaneously with caching the data in the cache 102 or without caching the data in the cache 102.

In the depicted embodiment, the cache 102 is embodied by a non-volatile, solid-state storage device, with a solid-state storage controller 104 and non-volatile, solid-state storage media 110. The non-volatile, solid-state storage media 110 may include flash memory, nano random access memory ("nano RAM or NRAM"), magneto-resistive RAM ("MRAM"), dynamic RAM ("DRAM"), phase change RAM ("PRAM"), racetrack memory, memristor memory, nanocrystal wire-based memory, silicon-oxide based sub-10 nanometer process memory, graphene memory, silicon-oxide-nitride-oxide-silicon ("SONOS") memory, resistive random-access memory ("RRAM"), programmable metallization cell ("PMC"), conductive-bridging RAM ("CBRAM"), or the like. In further embodiments, the cache 102 may include other types of non-volatile and/or volatile data storage, such as dynamic RAM ("DRAM"), static RAM ("SRAM"), magnetic data storage, optical data storage, and/or other data storage technologies.

The cache 102, in one embodiment, stores or preserves data in a log. The log, in a further embodiment, comprises a sequential, append-only log-based structure, or the like. The cache 102 stores at least a portion of the log on the solid-state storage media 110. The cache 102, in certain embodiments, may store a portion of the log, metadata for the log, or the like in volatile memory, such as RAM, and may store at least enough data of the log in the solid-state storage media 110 to recreate the log structure after an improper shutdown or other failure. In one embodiment, the log includes a head at an append point and a tail at an end of the log with the oldest data (data written earliest in time). In certain embodiments, the log may include multiple append points, multiple sub-logs, or the like. In a further embodiment, the cache 102 may store or preserve data in multiple logs.

In general, the cache 102 caches data for the backing store 118. The backing store 118, in one embodiment, is a backing store associated with the cache 102 and/or with the direct cache module 116. The backing store 118 may include a hard disk drive, an optical drive with optical

media, a magnetic tape drive, or another type of storage device. In one embodiment, the backing store 118 may have a greater data storage capacity than the cache 102. In another embodiment, the backing store 118 may have a higher latency, a lower throughput, or the like, than the cache 102.

The backing store 118 may have a higher latency, a lower throughput, or the like due to properties of the backing store 118 itself, or due to properties of a connection to the backing store 118. For example, in one embodiment, the cache 102 and the backing store 118 may each include non-volatile, solid-state storage media 110 with similar properties, but the backing store 118 may be in communication with the host device 114 over a data network, while the cache 102 may be directly connected to the host device 114, causing the backing store 118 to have a higher latency relative to the host 114 than the cache 102.

In one embodiment, the cache 102 and/or the backing store 118 are in communication with a processor of the host device 114 over one or more communications buses. In the depicted embodiment, the cache 102 and the backing store 118 are in communication with the host device 114 through the direct cache module 116. The cache 102 and/or the backing store 118, in one embodiment, may be direct attached storage ("DAS") of the host device 114. DAS, as used herein, is data storage that is connected to a device, either internally or externally, without a storage network in between.

In one embodiment, the cache 102 and/or the backing store 118 are internal to the host device 114 and are connected using a system bus, such as a peripheral component interconnect express ("PCI-e") bus, a Serial Advanced Technology Attachment ("SATA") bus, or the like. In another embodiment, the cache 102 and/or the backing store 118 may be external to the host device 114 and may be connected using a universal serial bus ("USB") connection, an Institute of Electrical and Electronics Engineers ("IEEE") 1394 bus ("FireWire"), an external SATA ("eSATA") connection, or the like. In other embodiments, the cache 102 and/or the backing store 118 may be connected to the host device 114 using a peripheral component interconnect ("PCI") express bus using external electrical or optical bus extension or bus networking solution such as Infiniband or PCI Express Advanced Switching ("PCIe-AS"), or the like.

In various embodiments, the cache 102 and/or the backing store 118 may be in the form of a dual-inline memory module ("DIMM"), a daughter card, or a micro-module. In another embodiment, the cache 102 and/or the backing store 118 may be elements within a rack-mounted blade. In another embodiment, the cache 102 and/or the backing store 118 may be contained within packages that are integrated directly onto a higher level assembly (e.g. mother board, lap top, graphics processor). In another embodiment, individual components comprising the cache

102 and/or the backing store 118 are integrated directly onto a higher level assembly without intermediate packaging.

In the depicted embodiment, the cache 102 includes one or more solid-state storage controllers 104 with a write data pipeline 106 and a read data pipeline 108, and a solid-state storage media 110. The backing store 118, in the depicted embodiment, includes a backing store controller 120. The solid-state storage controller 104 and the backing store controller 120, in certain embodiments, may receive storage requests, perform management functions and the like for the cache 102 and the backing store 118, or perform other functions. The solid-state storage controller 104 and/or the backing store controller 120, in various embodiments, may comprise one or more device drivers installed on the host device 114, logic hardware or firmware of the cache 102 and/or the backing store 118, a combination of one or more device drivers and logic hardware or firmware, or the like.

In one embodiment, the storage controller 104 sequentially writes data on the solid-state storage media 110 in a log structured format and within one or more physical structures of the storage elements, the data is sequentially stored on the solid-state storage media 110. Sequentially writing data involves the storage controller 104 streaming data packets into storage write buffers for storage elements, such as a chip (a package of one or more dies) or a die on a circuit board. When the storage write buffers are full, the data packets are programmed to a designated virtual or logical page ("LP"). Data packets then refill the storage write buffers and, when full, the data packets are written to the next LP. The next virtual page may be in the same bank or another bank. This process continues, LP after LP, typically until a virtual or logical erase block ("LEB") is filled.

In another embodiment, the streaming may continue across LEB boundaries with the process continuing, LEB after LEB. Typically, the storage controller 104 sequentially stores data packets in an LEB by order of processing. In one embodiment, where a write data pipeline 106 is used, the storage controller 104 stores packets in the order that they come out of the write data pipeline 106. This order may be a result of data segments arriving from a requesting device mixed with packets of valid data that are being read from another storage location as valid data is being recovered from another LEB during a recovery operation.

The sequentially stored data, in one embodiment, can serve as a log to reconstruct data indexes and other metadata using information from data packet headers. For example, in one embodiment, the storage controller 104 may reconstruct a storage index by reading headers to determine the data structure to which each packet belongs and sequence information to determine where in the data structure the data or metadata belongs. The storage controller 104,

in one embodiment, uses physical address information for each packet and timestamp or sequence information to create a mapping between the physical locations of the packets and the data structure identifier and data segment sequence. Timestamp or sequence information is used by the storage controller 104 to replay the sequence of changes made to the index and thereby reestablish the most recent state.

In one embodiment, erase blocks are time stamped or given a sequence number as packets are written and the timestamp or sequence information of an erase block is used along with information gathered from container headers and packet headers to reconstruct the storage index. In another embodiment, timestamp or sequence information is written to an erase block when the erase block is recovered.

In a read, modify, write operation, data packets associated with the logical structure are located and read in a read operation. Data segments of the modified structure that have been modified are not written to the location from which they are read. Instead, the modified data segments are again converted to data packets and then written to the next available location in the virtual page currently being written. Index entries for the respective data packets are modified to point to the packets that contain the modified data segments. The entry or entries in the index for data packets associated with the same logical structure that have not been modified will include pointers to original location of the unmodified data packets. Thus, if the original logical structure is maintained, for example to maintain a previous version of the logical structure, the original logical structure will have pointers in the index to all data packets as originally written. The new logical structure will have pointers in the index to some of the original data packets and pointers to the modified data packets in the virtual page that is currently being written.

In a copy operation, the index includes an entry for the original logical structure mapped to a number of packets stored on the solid-state storage media 110. When a copy is made, a new logical structure is created and a new entry is created in the index mapping the new logical structure to the original packets. The new logical structure is also written to the solid-state storage media 110 with its location mapped to the new entry in the index. The new logical structure packets may be used to identify the packets within the original logical structure that are referenced in case changes have been made in the original logical structure that have not been propagated to the copy and the index is lost or corrupted. In another embodiment, the index includes a logical entry for a logical block.

Beneficially, sequentially writing packets facilitates a more even use of the solid-state storage media 110 and allows a solid-storage device controller to monitor storage hot spots and

level usage of the various virtual pages in the solid-state storage media 110. Sequentially writing packets also facilitates a powerful, efficient garbage collection system, which is described in detail below. One of skill in the art will recognize other benefits of sequential storage of data packets.

The system 100 may comprise a log-structured storage system or log-structured array similar to a log-structured file system and the order that data is stored may be used to recreate an index. Typically an index that includes a logical-to-physical mapping is stored in volatile memory. If the index is corrupted or lost, the index may be reconstructed by addressing the solid-state storage media 110 in the order that the data was written. Within a logical erase block ("LEB"), data is typically stored sequentially by filling a first logical page, then a second logical page, etc. until the LEB is filled. The solid-state storage controller 104 then chooses another LEB and the process repeats. By maintaining an order that the LEBs were written to and by knowing that each LEB is written sequentially, the index can be rebuilt by traversing the solid-state storage media 110 in order from beginning to end. In other embodiments, if part of the index is stored in non-volatile memory, such as on the solid-state storage media 110, the solid-state storage controller 104 may only need to replay a portion of the solid-state storage media 110 to rebuild a portion of the index that was not stored in non-volatile memory. One of skill in the art will recognize other benefits of sequential storage of data packets.

In a further embodiment, instead of being connected directly to the host device 114 as DAS, the cache 102 and/or the backing store 118 may be connected to the host device 114 over a data network. For example, the cache 102 and/or the backing store 118 may include a storage area network ("SAN") storage device, a network attached storage ("NAS") device, a network share, or the like. In one embodiment, the system 100 may include a data network, such as the Internet, a wide area network ("WAN"), a metropolitan area network ("MAN"), a local area network ("LAN"), a token ring, a wireless network, a fiber channel network, a SAN, a NAS, ESCON, or the like, or any combination of networks. A data network may also include a network from the IEEE 802 family of network technologies, such Ethernet, token ring, Wi-Fi, Wi-Max, and the like. A data network may include servers, switches, routers, cabling, radios, and other equipment used to facilitate networking between the host device 114 and the cache 102 and/or the backing store 118.

In one embodiment, at least the cache 102 is connected directly to the host device 114 as a DAS device. In a further embodiment, the cache 102 is directly connected to the host device 114 as a DAS device and the backing store 118 is directly connected to the cache 102. For example, the cache 102 may be connected directly to the host device 114, and the backing store

118 may be connected directly to the cache 102 using a direct, wire-line connection, such as a PCI express bus, an SATA bus, a USB connection, an IEEE 1394 connection, an eSATA connection, a proprietary direct connection, an external electrical or optical bus extension or bus networking solution such as Infiniband or PCIe-AS, or the like. One of skill in the art, in light of this disclosure, will recognize other arrangements and configurations of the host device 114, the cache 102, and the backing store 118 suitable for use in the system 100.

The system 100 includes the host device 114 in communication with the cache 102 and the backing store 118 through the direct cache module 116. A host device 114 may be a host, a server, a storage controller of a SAN, a workstation, a personal computer, a laptop computer, a handheld computer, a supercomputer, a computer cluster, a network switch, router, or appliance, a database or storage appliance, a data acquisition or data capture system, a diagnostic system, a test system, a robot, a portable electronic device, a wireless device, or the like.

In the depicted embodiment, the host device 114 is in communication with the direct cache module 116. The direct cache module 116, in general, receives or otherwise detects read and write requests from the host device 114 directed to the backing store 118 and manages the caching of data in the cache 102, the destaging of cached data to the backing store 118, the grooming of data in the cache 102, and/or the eviction of data from the cache 102. In one embodiment, the direct cache module 116 comprises a software application, file system filter driver, combination of filter drivers, or the like on the host device 114.

In another embodiment, the direct cache module 116 comprises one or more storage controllers, such as the solid-state storage controller 104 of the cache 102 and/or the backing store controller 120 of the backing store 118. Figure 1B depicts a system 101 that is substantially similar to the system 100 of Figure 1A, but depicting a sequential log-based writing structure 122 of the cache 102, and with the storage controller 104 and the backing store controller 120 in communication with the direct cache module 116 as device drivers and/or filter drivers on the host device 114. The storage controller 104 and the backing store controller 120 may be integrated with the direct cache module 116 as device drivers on the host device 114, as dedicated hardware logic circuits or firmware of the cache 102 and/or the backing store 118, as a combination of one or more device drivers and dedicated hardware, or the like. In a further embodiment, the direct cache module 116 comprises a combination of one or more software drivers of the host device 114 and one or more storage controllers, or the like. The direct cache module 116, in various software, hardware, and combined software and hardware embodiments, may generally be referred to as a cache controller.

In one embodiment, the host device 114 loads one or more device drivers for the cache 102 and/or the backing store 118 and the direct cache module 116 communicates with the one or more device drivers on the host device 114. As described above, in certain embodiments, the solid-state storage controller 104 of the cache 102 and/or the backing store controller 120 may comprise device drivers on the host device 114. In another embodiment, the direct cache module 116 may communicate directly with a hardware interface of the cache 102 and/or the backing store 118. In a further embodiment, the direct cache module 116 may be integrated with the cache 102 and/or the backing store 118.

In one embodiment, the cache 102 and/or the backing store 118 have block device interfaces that support block device commands. For example, the cache 102 and/or the backing store 118 may support the standard block device interface, the ATA interface standard, the ATA Packet Interface ("ATAPI") standard, the small computer system interface ("SCSI") standard, and/or the Fibre Channel standard which are maintained by the InterNational Committee for Information Technology Standards ("INCITS"). The direct cache module 116 may interact with the cache 102 and/or the backing store 118 using block device commands to read, write, and clear (or trim) data. In one embodiment, the solid-state storage controller 104 and/or the backing store controller 120 provide block device interfaces to the direct cache module 116.

In one embodiment, the direct cache module 116 serves as a proxy for the backing store 118, receiving read and write requests for the backing store 118 directly from the host device 114. The direct cache module 116 may represent itself to the host device 114 as a storage device having a capacity similar to and/or matching the capacity of the backing store 118. The direct cache module 116, upon receiving a read request or write request from the host device 114, in one embodiment, fulfills the request by caching write data in the cache 102 or by retrieving read data from one of the cache 102 and the backing store 118 and returning the read data to the host device 114.

Data caches are typically organized into cache lines which divide up the physical capacity of the cache, these cache lines may be divided into several sets. A cache line is typically larger than a block or sector of a backing store associated with a data cache, to provide for prefetching of additional blocks or sectors and to reduce cache misses and increase the cache hit rate. Data caches also typically evict an entire, fixed size, cache line at a time to make room for newly requested data in satisfying a cache miss. Data caches may be direct mapped, fully associative, N-way set associative, or the like.

In a direct mapped cache, each block or sector of a backing store has a one-to-one mapping to a cache line in the direct mapped cache. For example, if a direct mapped cache has T

number of cache lines, the backing store associated with the direct mapped cache may be divided into T sections, and the direct mapped cache caches data from a section exclusively in the cache line corresponding to the section. Because a direct mapped cache always caches a block or sector in the same location or cache line, the mapping between a block or sector address and a cache line can be a simple manipulation of an address of the block or sector.

In a fully associative cache, any cache line can store data from any block or sector of a backing store. A fully associative cache typically has lower cache miss rates than a direct mapped cache, but has longer hit times (i.e., it takes longer to locate data in the cache) than a direct mapped cache. To locate data in a fully associative cache, either cache tags of the entire cache can be searched, a separate cache index can be used, or the like.

In an N-way set associative cache, each sector or block of a backing store may be cached in any of a set of N different cache lines. For example, in a 2-way set associative cache, either of two different cache lines may cache data for a sector or block. In an N-way set associative cache, both the cache and the backing store are typically divided into sections or sets, with one or more sets of sectors or blocks of the backing store assigned to a set of N cache lines. To locate data in an N-way set associative cache, a block or sector address is typically mapped to a set of cache lines, and cache tags of the set of cache lines are searched, a separate cache index is searched, or the like to determine which cache line in the set is storing data for the block or sector. An N-way set associative cache typically has miss rates and hit rates between those of a direct mapped cache and those of a fully associative cache.

The cache 102, in one embodiment, may have characteristics of both a directly mapped cache and a fully associative cache. A logical address space of the cache 102, in one embodiment, is directly mapped to an address space of the backing store 118 while the physical storage media 110 of the cache 102 is fully associative with regard to the backing store 118. In other words, each block or sector of the backing store 118, in one embodiment, is directly mapped to a single logical address of the cache 102 while any portion of the physical storage media 110 of the cache 102 may store data for any block or sector of the backing store 118. In one embodiment, a logical address is an identifier of a block of data and is distinct from a physical address of the block of data, but may be mapped to the physical address of the block of data. Examples of logical addresses, in various embodiments, include logical block addresses ("LBAs"), logical identifiers, object identifiers, pointers, references, and the like.

Instead of traditional cache lines, in one embodiment, the cache 102 has logical or physical cache data blocks associated with logical addresses that are equal in size to a block or sector of the backing store 118. In a further embodiment, the cache 102 caches ranges and/or

sets of ranges of blocks or sectors for the backing store 118 at a time, providing dynamic or variable length cache line functionality. A range or set of ranges of blocks or sectors, in a further embodiment, may include a mixture of contiguous and/or noncontiguous blocks. For example, the cache 102, in one embodiment, supports block device requests that include a mixture of contiguous and/or noncontiguous blocks and that may include "holes" or intervening blocks that the cache 102 does not cache or otherwise store.

In one embodiment, one or more groups of logical addresses of the cache 102 are directly mapped to corresponding logical addresses of the backing store 118. Directly mapping logical addresses of the cache 102 to logical addresses of the backing store 118, in one embodiment, provides a one-to-one relationship between the logical addresses of the backing store 118 and the logical addresses of the cache 102. Directly mapping logical addresses of the cache 102 to the logical or physical address space of the backing store 118, in one embodiment, precludes the use of an extra translation layer in the direct cache module 116, such as the use of cache tags, a cache index, the maintenance of a translation data structure, or the like. In one embodiment, while the logical address space of the cache 102 may be larger than a logical address space of the backing store 118, both logical address spaces include at least logical addresses 0-N. In a further embodiment, at least a portion of the logical address space of the cache 102 represents or appears as the logical address space of the backing store 118 to a client, such as the host device 114.

Alternatively, in certain embodiments where physical blocks or sectors of the backing store 118 are directly accessible using physical addresses, at least a portion of logical addresses in a logical address space of the cache 102 may be mapped to physical addresses of the backing store 118. At least a portion of the logical address space of the cache 102, in one embodiment, may correspond to the physical address space of the backing store 118. At least a subset of the logical addresses of the cache 102, in this embodiment, is directly mapped to corresponding physical addresses of the backing store 118.

In one embodiment, the logical address space of the cache 102 is a sparse address space that is either as large as or is larger than the physical storage capacity of the cache 102. This allows the backing store 118 to have a larger storage capacity than the cache 102, while maintaining a direct mapping between the logical addresses of the cache 102 and logical or physical addresses of the backing store 118. The sparse logical address space may be thinly provisioned, in one embodiment. In a further embodiment, as the direct cache module 116 writes data to the cache 102 using logical addresses, the cache 102 directly maps the logical addresses to distinct physical addresses or locations on the solid-state storage media 110 of the cache 102,

such that the physical addresses or locations of data on the solid-state storage media 110 are fully associative with the backing store 118.

In one embodiment, the direct cache module 116 and/or the cache 102 use the same mapping structure to map addresses (either logical or physical) of the backing store 118 to logical addresses of the cache 102 and to map logical addresses of the cache 102 to locations/physical addresses of a block or sector (or range of blocks or sectors) on the physical solid-state storage media 110. In one embodiment, using a single mapping structure for both functions eliminates the need for a separate cache map, cache index, cache tags, or the like, decreasing access times of the cache 102.

Once the direct cache module 116 has destaged dirty data from the cache 102, the data is clean and the direct cache module 116 may clear, trim, replace, expire, and/or evict the data from the cache 102 and the physical addresses and associated physical storage media 110 may be freed to store data for other logical addresses. In one embodiment, as described above, the solid state storage controller 104 stores data at physical addresses using a log-based, append-only writing structure such that data evicted from the cache 102 or overwritten by a subsequent write request invalidates other data in the log. Consequently, a grooming or garbage collection process recovers the physical capacity of the invalid data in the log. One embodiment of the log-based, append only writing structure is logically ring-like data structure, as new data is appended to the log-based writing structure, previously used physical capacity is reused in a circular, theoretically infinite manner.

In one embodiment, the direct cache module 116 selects data of the cache 102 for grooming and/or evicting based on a grooming cost for the data. For example, the direct cache module 116 may select a region of the cache 102 periodically and examine a grooming cost for the selected region. In another embodiment, the direct cache module 116 may maintain a grooming candidate set of regions with low grooming costs and defines, identifies, or otherwise designates a low cost region from the grooming candidate set for grooming. Selecting data of the cache 102 for grooming and/or evicting based on periodically determined grooming costs and a grooming candidate set, such as an ordered set of the N lowest grooming costs, in certain embodiments, may lower processing overhead and determination times compared to determining a cost for each block or region of data at each grooming decision point. The direct cache module 116, in various embodiments, may recover storage capacity of the low cost region by invalidating, evicting, trimming, or otherwise clearing certain data from the low cost region and/or by copying certain data forward on the log-based writing structure to retain the data, such as dirty write data, frequently accessed data, or the like.

Alternatively, instead of selecting data for eviction based on grooming cost, the direct cache module 116 may evict data exclusively from a tail region 124 of the log-based writing structure 122 (e.g. the log described above) in the order 126 that regions were written to the log 122 or a first-in-first-out ("FIFO") order 126, to optimize efficiency of the cache 102 and management of the log-based writing structure 122. For example, the direct cache module 116 may evict data region by region, progressing successively from a tail region 124 of the log 122 toward a head region 128 or append point 128 of the log 122, in the order 126 that data was written to the log 122.

A region of the cache 102 is a physical or logical grouping or segment of the non-volatile storage media 110 of the cache, such as a block, a sector, a page, a logical block, a logical page, a physical erase block, a logical erase block, a packet, an error-correcting code ("ECC") chunk, or another logical or physical division of the cache 102. In one embodiment, the direct cache module 116 evicts data of a region comprising a logical erase block made up of a plurality of logical pages, each including a set of sectors, packets, or the like. Each sector/packet may include a header. In one embodiment, a plurality of sectors/packets are protected by error correction bits. The error correction bits and the data associated with the error correction bits comprise an ECC codeword. The ECC codeword may include multiple sectors/packets that may or may not be boundary aligned with the ECC codeword. The smallest read unit, in certain embodiments, is an ECC code word and the smallest write unit may be a logical page.

In certain embodiments, due to the log-based storage architecture of the cache 102, the cache 102 eventually reaches a point where the amount of data stored approaches the physical capacity limits of the non-volatile storage media 110. Generally, previously used storage capacity can be recovered to provide new space for the log-based writing structure 122 of the cache 102 to use. The recovered space generally comes from re-using space that has been invalidated due to a subsequent change to the data of a sector or block. This process is known as garbage collection, grooming, or compaction. Generally, the direct cache module 116 examines a logical erase block 124 and preserves valid data from the logical erase block 124 by writing the data at one or more append points 128 of the log-based writing structure 122 (generally at the head 128 of the log 122) so that the whole logical erase block 124 can be erased and re-used by the log-based writing structure 122.

To cache data as efficiently as possible, the direct cache module 116 maximizes the number of times that the direct cache module 116 services a data request from the cache 102 versus the number of times the associated data must be retrieved from the backing store 118, referred to as the cache hit rate or cache efficiency. Cache management for maximum cache

efficiency may compete with log-based storage management for the limited resource of available storage capacity on the non-volatile storage media 110. In certain embodiments, the direct cache module 116 maximizes cache efficiency and maximizes log-based storage management while minimizing the latency in servicing storage requests by evicting data from the cache 102 exclusively in FIFO order 126, from the tail 124 of the log-based writing structure 122 toward the head 128. Evicting in FIFO order 126 may simplify handling of TRIM (invalidation) commands, utilize the existing log-based writing structure 122 of the cache 102 to manage eviction from the cache 102, provide wear-leveling for the non-volatile storage media 110 by grooming and/or evicting from the oldest region 124, and/or select optimal data for eviction by evicting the oldest data.

In one embodiment, the cache 102 may store a plurality of independent cache units, each of which cache data for different cache clients, different backing stores 118, or the like. For example, the solid-state storage controller 104 may manage multiple virtual storage units ("VSUs"), one or more of which may be configured as cache units, to cache data for clients executing on the host device 114, over a data network, or the like. Each VSU is a data structure maintained by the solid-state storage controller 104 to logically divide the solid-state storage device of the cache 102 into independent storage units or containers, so that the solid-state storage device of the cache 102 may be shared between multiple clients. VSUs may serve as cache units, object stores, general-purpose storage units, swap/memory extension units, sparse address space units, or the like.

Each VSU may have different properties and attributes, such as different use cases, different quality-of-service ("QoS") levels, different priority levels, different logical address space types (e.g. sparse logical address space, contiguous logical address space), different replication attributes, different logical and/or physical storage capacities, or the like. VSUs, in certain embodiments, may be independently created, deleted, and managed. The solid-state storage controller 104 may store metadata defining attributes of the VSUs in volatile and/or nonvolatile storage of the host device 114, the solid-state storage device 102, the backing store 118, or the like.

While each VSU may be logically independent, in one embodiment, data stored in different VSUs is intermingled in the solid-state storage media 110. For example, the solid-state storage media 110 may store data using a sequential, append-only, log-based writing structure 122, and the solid-state storage controller 104 may write data of several VSUs sequentially to an append point 128 of the log-based writing structure 122 as the direct cache module 116 receives data to cache. Because data from each VSU or VSU cache unit, in certain embodiments, is

written to the same append point 128, the data from different VSUs may be dispersed throughout the log-based writing structure 122 on the solid-state storage media 110 of the cache 102.

By logically separating the data from different VSUs and/or VSU cache units but intermingling the data physically, data from each VSU receives the same data protection characteristics. For example, the solid-state storage controller 104, the write data pipeline 106, and the read data pipeline 108 provide certain data protection characteristics for data, such as error correction, garbage collection or storage capacity recovery, power cut or power loss protection, or the like to protect the integrity of data on the solid-state storage media 110. The solid-state storage controller 104 applies these data protection characteristics to data regardless of which VSU logically corresponds to the data.

In embodiments where the cache 102 comprises multiple VSUs or other cache units, regions of the solid-state storage media 110 may store data for a plurality of cache units, intermingled in a log-based writing structure 122 of the cache 102 or the like. The direct cache module 116, in certain embodiments, accounts for one or more attributes of the cache units in the grooming cost for a region, and/or selectively retains data from a tail region 124 based on cache unit attributes. Attributes of a cache unit may include a relative priority of cache units, allocated storage capacities for cache units, quality-of-service levels for cache units, or the like.

For example, by accounting for cache unit attributes in grooming costs or by selecting data from a tail region 124 to retain instead of evict based on cache unit attributes, the direct cache module 116 may select regions for grooming, garbage collection, and/or eviction that have more data from cache units with lower priorities than other cache units, that have reached or exceeded an allocated storage capacity, that have a lower quality-of-service level than other cache units, or the like. In this manner the direct cache module 116 may determine grooming costs to maintain a relative priority of cache units, allocated storage capacities for cache units, quality-of-service levels for cache units, or the like by selectively evicting data from storage regions based on the determined grooming costs.

DATA CACHING

Figure 2 depicts one embodiment of a host device 114. The host device 114 may be similar, in certain embodiments, to the host device 114 depicted in Figures 1A and 1B. The depicted embodiment includes a user application 502 in communication with a storage client 504. The storage client 504 is in communication with a direct cache module 116, which, in one embodiment, is substantially similar to the direct cache module 116 of Figures 1A and 1B, described above. The direct cache module 116, in the depicted embodiment, is in

communication with the cache 102 and the backing store 118 through the storage controller 104 and the backing store controller 120.

In one embodiment, the user application 502 is a software application operating on or in conjunction with the storage client 504. The storage client 504 manages file systems, files, data, and the like and utilizes the functions and features of the direct cache module 116, the cache 102, and the backing store 118. Representative examples of storage clients include, but are not limited to, a server, a file system, an operating system, a database management system ("DBMS"), a volume manager, and the like.

In the depicted embodiment, the storage client 504 is in communication with the direct cache module 116. In a further embodiment, the storage client 504 may also be in communication with the cache 102 and/or the backing store 118 directly. The storage client 504, in one embodiment, reads data from and writes data to the backing store 118 through the direct cache module 116, which uses the cache 102 to cache read data and/or write data for the backing store 118. In a further embodiment, the direct cache module 116 caches data in a manner that is substantially transparent to the storage client 504, with the storage client 504 sending read requests and write requests directly to the direct cache module 116.

In one embodiment, the direct cache module 116 has exclusive access to and/or control over the cache 102 and the backing store 118. The direct cache module 116 may represent itself to the storage client 504 as a storage device. For example, the direct cache module 116 may represent itself as a conventional block storage device, or the like. In a particular embodiment, the direct cache module 116 may represent itself to the storage client 504 as a storage device having the same number of logical blocks (0 to N) as the backing store 118. In another embodiment, the direct cache module 116 may represent itself to the storage client 504 as a storage device have the more logical blocks (0 to N+X) as the backing store 118, where X = the number of logical blocks addressable by the direct cache module 116 beyond N. In certain embodiments, $X = 2^{64} - N$.

As described above with regard to the direct cache module 116 depicted in the embodiments of Figures 1A and 1B, in various embodiments, the direct cache module 116 may be embodied by one or more of a storage controller 104 of the cache 102 and/or a backing store controller 120 of the backing store 118; a separate hardware controller device that interfaces with the cache 102 and the backing store 118; a device driver loaded on the host device 114; and the like.

In one embodiment, the host device 114 loads a device driver for the direct cache module 116. In a further embodiment, the host device 114 loads device drivers for the cache 102 and/or

the backing store 118, such as one or more device drivers of the storage controller 104 and/or the backing store controller 120. The direct cache module 116 may communicate with the cache 102 and/or the backing store 118 through device drivers loaded on the host device 114, through the storage controller 104 of the cache 102 and/or through the backing store controller 120 of the backing store 118, or the like.

In one embodiment, the storage client 504 communicates with the direct cache module 116 through an Input/Output ("I/O") interface represented by a block I/O emulation layer 506. In certain embodiments, the fact that the direct cache module 116 is providing caching services in front of one or more caches 102, and/or one or more backing stores, such as the backing store 118, may be transparent to the storage client 504. In such an embodiment, the direct cache module 116 may present (i.e., identify itself as) a conventional block device to the storage client 504.

In a further embodiment, the cache 102 and/or the backing store 118 either include a distinct block I/O emulation layer 506 or may be conventional block storage devices. Certain conventional block storage devices divide the storage media into volumes or partitions. Each volume or partition may include a plurality of sectors. One or more sectors are organized into a logical block. In certain storage systems, such as those interfacing with the Windows® operating systems, the logical blocks are referred to as clusters. In other storage systems, such as those interfacing with UNIX, Linux, or similar operating systems, the logical blocks are referred to simply as blocks. A logical block or cluster represents a smallest physical amount of storage space on the storage media that is addressable by the storage client 504. A block storage device may associate n logical blocks available for user data storage across the storage media with a logical block address, numbered from 0 to n. In certain block storage devices, the logical block addresses may range from 0 to n per volume or partition. In conventional block storage devices, a logical block address maps directly to a particular logical block. In conventional block storage devices, each logical block maps to a particular set of physical sectors on the storage media.

However, the direct cache module 116, the cache 102 and/or the backing store 118, in certain embodiments, may not directly or necessarily associate logical block addresses with particular physical blocks. The direct cache module 116, the cache 102, and/or the backing store 118 may emulate a conventional block storage interface to maintain compatibility with block storage clients 504 and with conventional block storage commands and protocols.

When the storage client 504 communicates through the block I/O emulation layer 506, the direct cache module 116 appears to the storage client 504 as a conventional block storage device. In one embodiment, the direct cache module 116 provides the block I/O emulation layer

506 which serves as a block device interface, or API. In this embodiment, the storage client 504 communicates with the direct cache module 116 through this block device interface. In one embodiment, the block I/O emulation layer 506 receives commands and logical block addresses from the storage client 504 in accordance with this block device interface. As a result, the block I/O emulation layer 506 provides the direct cache module 116 compatibility with block storage clients 504. In a further embodiment, the direct cache module 116 may communicate with the cache 102 and/or the backing store 118 using corresponding block device interfaces.

In one embodiment, a storage client 504 communicates with the direct cache module 116 through a direct interface layer 508. In this embodiment, the direct cache module 116 directly exchanges information specific to the cache 102 and/or the backing store 118 with the storage client 504. Similarly, the direct cache module 116, in one embodiment, may communicate with the cache 102 and/or the backing store 118 through direct interface layers 508.

A direct cache module 116 using the direct interface 508 may store data on the cache 102 and/or the backing store 118 as blocks, sectors, pages, logical blocks, logical pages, erase blocks, logical erase blocks, packets, ECC chunks or in any other format or structure advantageous to the technical characteristics of the cache 102 and/or the backing store 118. For example, in one embodiment, the backing store 118 comprises a hard disk drive and the direct cache module 116 stores data on the backing store 118 as contiguous sectors of 512 bytes, or the like, using physical cylinder-head-sector addresses for each sector, logical block addresses for each sector, or the like. The direct cache module 116 may receive a logical address and a command from the storage client 504 and perform the corresponding operation in relation to the cache 102, and/or the backing store 118. The direct cache module 116, the cache 102, and/or the backing store 118 may support a block I/O emulation layer 506, a direct interface 508, or both a block I/O emulation layer 506 and a direct interface 508.

As described above, certain storage devices, while appearing to a storage client 504 to be a block storage device, do not directly associate particular logical block addresses with particular physical blocks, also referred to in the art as sectors. Such storage devices may use a logical-to-physical translation layer 510. In the depicted embodiment, the cache 102 includes a logical-to-physical translation layer 510. In a further embodiment, the backing store 118 may also include a logical-to-physical translation layer 510. In another embodiment, the direct cache module 116 maintains a single logical-to-physical translation layer 510 for the cache 102 and the backing store 118. In another embodiment, the direct cache module 116 maintains a distinct logical-to-physical translation layer 510 for each of the cache 102 and the backing store 118.

The logical-to-physical translation layer 510 provides a level of abstraction between the logical block addresses used by the storage client 504 and the physical block addresses at which the cache 102 and/or the backing store 118 store the data. In the depicted embodiment, the logical-to-physical translation layer 510 maps logical block addresses to physical block addresses of data stored on the media of the cache 102. This mapping allows data to be referenced in a logical address space using logical identifiers, such as a logical block address. A logical identifier does not indicate the physical location of data in the cache 102, but is an abstract reference to the data. One example of a logical-to-physical translation layer 510 includes the direct mapping module 716 of Figure 4 discussed below.

In the depicted embodiment, the cache 102 and the backing store 118 separately manage physical block addresses in the distinct, separate physical address spaces of the cache 102 and the backing store 118. In one example, contiguous logical block addresses may in fact be stored in non-contiguous physical block addresses as the logical-to-physical translation layer 510 determines the location on the physical media 110 of the cache 102 at which to perform data operations.

Furthermore, in one embodiment, the logical address space of the cache 102 is substantially larger than the physical address space or storage capacity of the cache 102. This "thinly provisioned" or "sparse address space" embodiment, allows the number of logical addresses for data references to greatly exceed the number of possible physical addresses. A thinly provisioned and/or sparse address space also allows the cache 102 to cache data for a backing store 118 with a larger address space (i.e., a larger storage capacity) than the physical address space of the cache 102.

In one embodiment, the logical-to-physical translation layer 510 includes a map or index that maps logical block addresses to physical block addresses. The map or index may be in the form of a B-tree, a content addressable memory ("CAM"), a binary tree, and/or a hash table, and the like. In certain embodiments, the logical-to-physical translation layer 510 is a tree with nodes that represent logical block addresses and include references to corresponding physical block addresses.

As stated above, in conventional block storage devices, a logical block address maps directly to a particular physical block. When a storage client 504 communicating with the conventional block storage device deletes data for a particular logical block address, the storage client 504 may note that the particular logical block address is deleted and can re-use the physical block associated with that deleted logical block address without the need to perform any other action.

Conversely, when a storage client 504, communicating with a storage controller 104 or device driver with a logical-to-physical translation layer 510 (a storage controller 104 or device driver that does not map a logical block address directly to a particular physical block), deletes data of a logical block address, the corresponding physical block address may remain allocated because the storage client 504 may not communicate the change in used blocks to the storage controller 104 or device driver. The storage client 504 may not be configured to communicate changes in used blocks (also referred to herein as "data block usage information"). Because the storage client 504, in one embodiment, uses the block I/O emulation layer 506, the storage client 504 may erroneously believe that the direct cache module 116, the cache 102, and/or the backing store 118 is a conventional block storage device that would not utilize the data block usage information. Or, in certain embodiments, other software layers between the storage client 504 and the direct cache module 116, the cache 102, and/or the backing store 118 may fail to pass on data block usage information.

Consequently, the storage controller 104 or device driver may preserve the relationship between the logical block address and a physical address and the data on the cache 102 and/or the backing store 118 corresponding to the physical block. As the number of allocated blocks increases, the performance of the cache 102 and/or the backing store 118 may suffer depending on the configuration of the cache 102 and/or the backing store 118.

Specifically, in certain embodiments, the cache 102 and/or the backing store 118 are configured to store data sequentially, using an append-only writing process, and use a storage space recovery process that re-uses non-volatile storage media storing deallocated/unused logical blocks. Specifically, as described above, the cache 102 and/or the backing store 118 may sequentially write data on the solid-state storage media 110 in a log structured format and within one or more physical structures of the storage elements, the data is sequentially stored on the solid-state storage media 110. Those of skill in the art will recognize that other embodiments that include several caches 102 can use the same append-only writing process and storage space recovery process.

As a result of storing data sequentially and using an append-only writing process, the cache 102 and/or the backing store 118 achieve a high write throughput and a high number of I/O operations per second ("IOPS"). The cache 102 and/or the backing store 118 may include a storage space recovery, or garbage collection process that re-uses data storage cells to provide sufficient storage capacity. The storage space recovery process reuses storage cells for logical blocks marked as deallocated, invalid, unused, or otherwise designated as available for storage space recovery in the logical-physical translation layer 510. In one embodiment, the direct cache

module 116 marks logical blocks as deallocated or invalid based on a cache eviction policy, to recover storage capacity for caching additional data for the backing store 118. The direct cache module 116, in certain embodiments, selects data that is either cached read data or destaged, cleaned write data to clear, invalidate, or evict. The storage space recovery process is described in greater detail below with regard to the groomer module 702 of Figure 4.

As described above, the storage space recovery process determines that a particular section of storage may be recovered. Once a section of storage has been marked for recovery, the cache 102 and/or the backing store 118 may relocate valid blocks (e.g. packets, pages, sectors, etc.) in the section. The storage space recovery process, when relocating valid blocks, copies the packets and writes them to another location so that the particular section of storage may be reused as available storage space, typically after an erase operation on the particular section. The cache 102 and/or the backing store 118 may then use the available storage space to continue sequentially writing data in an append-only fashion. Consequently, the storage controller 104 expends resources and overhead in preserving data in valid blocks. Therefore, physical blocks corresponding to deleted logical blocks may be unnecessarily preserved by the storage controller 104, which expends unnecessary resources in relocating the physical blocks during storage space recovery.

Some storage devices are configured to receive messages or commands notifying the storage device of these unused logical blocks so that the storage device may deallocate the corresponding physical blocks (e.g. the physical storage media 110 storing the unused packets, pages, sectors, etc.). As used herein, to deallocate a physical block includes marking the physical block as invalid, unused, or otherwise designating the physical block as available for storage space recovery, its contents on storage media no longer needing to be preserved by the storage device. Data block usage information may also refer to information maintained by a storage device regarding which physical blocks are allocated and/or deallocated/unallocated and changes in the allocation of physical blocks and/or logical-to-physical block mapping information. Data block usage information may also refer to information maintained by a storage device regarding which blocks are in use and which blocks are not in use by a storage client 504. Use of a block may include storing of data in the block on behalf of the storage client 504, reserving the block for use by the storage client 504, and the like.

While physical blocks may be deallocated, in certain embodiments, the cache 102 and/or the backing store 118 may not immediately erase the data on the storage media. An erase operation may be performed later in time. In certain embodiments, the data in a deallocated physical block may be marked as unavailable by the cache 102 and/or the backing store 118 such

that subsequent requests for data in the physical block return a null result or an empty set of data. In certain embodiments, the direct cache module 116 evicts and/or invalidates data by deallocating physical blocks corresponding to the data in the cache 102.

One example of a command or message for such deallocation is the "TRIM" function is described in greater detail in U.S. Patent Application Number 12/711,113 entitled "APPARATUS, SYSTEM, AND METHOD FOR DATA BLOCK USAGE INFORMATION SYNCHRONIZATION FOR A NON-VOLATILE STORAGE VOLUME" and filed on February 23, 2010 and in U.S. Patent Application Number 11/952,113 entitled "APPARATUS, SYSTEM, AND METHOD FOR MANAGING DATA IN A STORAGE DEVICE WITH AN EMPTY DATA TOKEN DIRECTIVE" and filed on December 6, 2007, which are incorporated herein by reference. A storage device, upon receiving a TRIM command, may deallocate physical blocks for logical blocks whose data is no longer needed by the storage client 504. A storage device that deallocates physical blocks may achieve better performance and increased storage space, especially storage devices that write data using certain processes and/or use a similar data storage recovery process as that described above.

Consequently, the performance of the storage device is enhanced as physical blocks are deallocated when they are no longer needed such as through the TRIM command or other similar deallocation commands issued to the cache 102 and/or the backing store 118. In one embodiment, the direct cache module 116 clears, trims, and/or evicts cached data from the cache 102 based on a cache eviction policy, or the like. As used herein, clearing, trimming, or evicting data includes deallocating physical media associated with the data, marking the data as invalid or unused (using either a logical or physical address of the data), erasing physical media associated with the data, overwriting the data with different data, issuing a TRIM command or other deallocation command relative to the data, or otherwise recovering storage capacity of physical storage media corresponding to the data. Clearing cached data from the cache 102 based on a cache eviction policy frees storage capacity in the cache 102 to cache more data for the backing store 118.

The direct cache module 116, in various embodiments, may represent itself, the cache 102, and the backing store 118 to the storage client 504 in different configurations. In one embodiment, the direct cache module 116 may represent itself to the storage client 504 as a single storage device (e.g., as the backing store 118, as a storage device with a similar physical capacity as the backing store 118, or the like) and the cache 102 may be transparent or invisible to the storage client 504. In another embodiment, the direct cache module 116 may represent itself to the direct cache module 116 as a cache device (e.g., as the cache 102, as a cache device

with certain cache functions or APIs available, or the like) and the backing store 118 may be separately visible and/or available to the storage client 504 (with part of the physical capacity of the backing store 118 reserved for the cache 201). In a further embodiment, the direct cache module 116 may represent itself to the storage client 504 as a hybrid cache/storage device including both the cache 102 and the backing store 118.

Depending on the configuration, the direct cache module 116 may pass certain commands down to the cache 102 and/or to the backing store 118 and may not pass down other commands. In a further embodiment, the direct cache module 116 may support certain custom or new block I/O commands. In one embodiment, the direct cache module 116 supports a deallocation or trim command that clears corresponding data from both the cache 102 and the backing store 118, i.e., the direct cache module 116 passes the command to both the cache 102 and the backing store 118. In a further embodiment, the direct cache module 116 supports a flush type trim or deallocation command that ensures that corresponding data is stored in the backing store 118 (i.e., that the corresponding data in the cache 102 is clean) and clears the corresponding data from the cache 102, without clearing the corresponding data from the backing store 118. In another embodiment, the direct cache module 116 supports an evict type trim or deallocation command that evicts corresponding data from the cache 102, marks corresponding data for eviction in the cache 102, or the like, without clearing the corresponding data from the backing store 118.

In one embodiment, the direct cache module 116 operates according to a write through cache policy (described below) or another cache policy and manages TRIM (invalidation) commands from a storage client 504 in substantially the same manner as write commands. This means that TRIM commands are sent to both the storage controller 104 and the backing store controller 120. In one embodiment, the direct cache module 116 reviews write commands, TRIM commands, and read commands received from a storage client 504 to determine if the blocks/packets/sectors involved in those commands (indicated by the address ranges) overlap, or collide, with blocks/packets/sectors or address ranges presently in the process of being executed. In one embodiment, if the write command overlaps with another pending write command, pending read command, or pending TRIM command, the write command is acknowledged and the write command is sent to the backing store 118. In addition, in certain embodiments, the direct cache module 116 may cooperate with the storage controller 104 to invalidate blocks/packets/sectors identified in the write command, by for example, sending the storage controller 104 a TRIM command to the relevant address range. In one embodiment, if the TRIM command overlaps with another pending write command, or pending read command, no change

is made in the cache 102 in response to the TRIM command. In certain embodiments, the direct cache module 116 may send the TRIM command to the backing store controller 120.

In one embodiment, if a read command overlaps with another pending write command, or pending TRIM command, and any of the blocks/packets/sectors for the read command are present in the cache 102, the read command for those blocks/packets/sectors is serviced from the cache 102. The direct cache module 116 may cooperate with the backing store controller 120 to read blocks/packets/sectors not present in the cache 102. In one embodiment, if a read command overlaps with another pending write command, or pending TRIM command, and none of the blocks/packets/sectors for the read command are present in the cache 102, the read command is serviced from the backing store 118 but the data for servicing the read command is not stored in the cache, due to the collision.

In embodiments where the direct cache module 116 does not evict data exclusively from a tail 124 of the log-based writing structure 122 of the cache 102, but may select any region of the log-based writing structure for eviction, TRIM commands may be persisted to avoid the condition where a TRIM message may be lost if the cache 102 encounters a power failure or invalid shutdown during a grooming and/or eviction operation. Without persisting the TRIM command, in such an embodiment, the storage controller 104 may not know whether or not data is valid or invalid, which version of data is valid, or the like. However, by evicting data exclusively from a tail 124 of the log-based writing structure 122 of the cache 102, in certain embodiments, the direct cache module 116 avoids this problem of persisting TRIM commands, as the direct cache module 116 evicts the oldest data from the cache 102. If a TRIM message is lost during a grooming and/or eviction operation for a tail region 124, any data associated with the lost TRIM message that remains in the log-based writing structure 122 (e.g. to the right or clockwise from the tail region 124) is newer (later in time) than the lost TRIM message, and is therefore unaffected by its loss. In certain embodiments, even when the direct cache module 116 queues multiple regions 124 of the log-based storage structure 122 for eviction, the direct cache module 116 guarantees that regions 124 will be evicted exclusively in FIFO order 126, from the tail region 124 toward the head region 128 or append point 128, so that the storage controller 104 can properly determine data validity, even without persistent TRIM messages.

In a further embodiment, the direct cache module 116 may receive, detect, and/or intercept one or more predefined commands that a storage client 504 or another storage manager sent to the backing store 118, that a storage manager sends to a storage client 504, or the like. For example, in various embodiments, the direct cache module 116 or a portion of the direct cache module 116 may be part of a filter driver that receives or detects the predefined

commands, the direct cache module 116 may register with an event server to receive a notification of the predefined commands, or the like. The direct cache module 116, in one embodiment, performs one or more actions on the cache 102 in response to detecting the one or more predefined commands for the backing store 118, such as writing or flushing data related to a command from the cache 102 to the backing store 118, evicting data related to a command from the cache 102, switching from a write back policy to a write through policy for data related to a command, or the like.

One example of predefined commands that the direct cache module 116 may intercept or respond to, in one embodiment, includes a "freeze/thaw" commands. "Freeze/thaw" commands are used in SANs, storage arrays, and the like, to suspend storage access, such as access to the backing store 118 or the like, to take an snapshot or backup of the storage without interrupting operation of the applications using the storage. "Freeze/thaw" commands alert a storage client 504 that a snapshot is about to take place, the storage client 504 flushes pending operations, for example in-flight transactions, or data cached in volatile memory, the snapshot takes place while the storage client 504 use of the storage is in a "frozen" or ready state, and once the snapshot is complete the storage client 504 continues normal use of the storage in response to a thaw command.

The direct cache module 116, in one embodiment, flushes or cleans dirty data from the cache 102 to the backing store 118 in response to detecting a "freeze/thaw" command. In a further embodiment, the direct cache module 116 suspends access to the backing store 118 during a snapshot or other backup of a detected "freeze/thaw" command and resumes access in response to a completion of the snapshot or other backup. In another embodiment, the direct cache module 116 may cache data for the backing store 118 during a snapshot or other backup without interrupting the snapshot or other backup procedure. In other words, rather than the backup/snapshot software signaling the application to quiesce I/O operations, the direct cache module 116 receives and responds to the freeze/thaw commands.

In certain embodiments, the direct cache module 116 is configured to operate in one of five cache policies, cache modes. In one embodiment, the cache policies include: a write back cache policy; a write through cache policy; a write around cache policy; a read only cache policy; and a bypass cache policy. Each cache policy may dictate when, how, or if, acknowledgement for a data write operation is sent to a storage client 504. In certain embodiments, writing the data to the cache 102 will be sufficiently persisted to merit sending an acknowledgement once the data is written to the cache 102. In other embodiments, an acknowledgement may not be not sent until the data is written to the backing store 130. In other

embodiments, an acknowledgement may not be not sent until the data is written to both the backing store 118 and the cache 102.

Under a write back cache policy (also referred to as a copy back cache) policy, the direct cache 120 may direct write requests to the cache device 102 such that the data is written first to the cache 102. The data may be later migrated to the backing store 118. For example, the data may be moved to the backing store 118 when a cache member is evicted from the cache device 102 (an approach frequently called lazy write back). In such embodiments, the direct cache module 116 may also track whether data written to the cache 102 is clean or dirty to facilitate migrating data from the cache 102 to the backing store 118. Other approaches to moving data from the cache 102 to the backing store 118 (such as opportunistic write back) may also be used. For example, where the cache 102 is a solid state storage device (such as a Flash device), evictions may be triggered as part of a grooming operation. Such a cache 102 may store data in circumstances that would not be advisable for a volatile storage device since the risk of data loss is very low for a Flash memory device in good condition.

In certain embodiments, under a write back cache policy, the acknowledgement is sent once the data is written to the cache 102. The cache 120 may be configured to provide an additional notification when the data is moved from the cache 102 to the backing store 118. Reads may be executed by direct cache module 116 such that, when the cache 102 has the requested data, the read is performed from the cache 102 and not the backing store 118.

Under a write through cache policy, the direct cache module 116 writes data synchronously to the cache 102 and the backing store 118. In one embodiment, the direct cache module 116 ensures that the write request is executed against both the cache 102 and the backing store 118 in a manner that is transparent to the host 114 or storage clients 504. In certain embodiments, the acknowledgement is sent when the data is written on the backing store 118, when the data is written on both the backing store 118 and the cache 102, or the like. Thus, in certain embodiments, the direct cache module 116, when implementing a write through cache policy, may send write requests to the storage controller 104 and to the backing store controller 120, with a single acknowledgment being sent when the backing store 118 fulfills the request. The read policy associated with the write through cache policy may be identical to that described above for the write back cache policy.

In certain embodiments, under the write around cache policy, the direct cache module 116 directs writes only to the backing store 118. Acknowledgements may be sent when the data is stored in the backing store 118. In one embodiment, write requests coming from the storage clients 504 are not written to the cache 102. Read requests for data no yet present in the cache

102 are written to the cache 102. Thus, the direct cache module 116 may, when implementing a write around cache policy, send write requests exclusively to the backing store 118 such that the write requests bypass the cache 102, with acknowledgements sent when the backing store 118 fulfills the write request. Reads may be executed against the cache 102 as described above under the write around cache policy.

Under the read only cache policy, writes are directly only to the backing store 118. Acknowledgements may be sent when the data is stored in the backing store 118. Reads may be executed against the cache 102 as described above.

Under the bypass cache policy, writes are directly only to the backing store 118. Acknowledgements may be sent when the data is stored in the backing store 118. Reads may be directed to the backing store 118 instead of the cache 102. The direct cache module 116 may implement the bypass cache policy by sending write requests and read requests exclusively to the backing store controller 120.

Figure 3 depicts one embodiment of the direct cache module 116a. In the depicted embodiment, the direct cache module 116a includes a grooming cost module 602, a grooming candidate set module 604, and a low cost module 606. Another embodiment of a direct cache module is described below with regard to the direct cache module 116b of Figure 4, which includes several additional modules. In general, the direct cache module 116a grooms data of the cache 102. Grooming data, as used herein, includes processing a region of data to recover physical storage capacity of the region by copying data of the region to another location in the cache 102, such as to an append point 128 of a log 122, and/or evicting, invalidating, trimming, or otherwise clearing data from the region. Another term for grooming is garbage collection.

In one embodiment, the grooming cost module 602 examines a grooming cost for a selected region of the cache 102. As described below with regard to the grooming clock module 704 of Figure 4, in certain embodiments, regions of the cache 102 are selected periodically for the grooming cost module 602 to examine. The grooming cost module 602 may determine a grooming cost for a selected region itself, may receive a grooming cost for a selected region from the storage controller 104 or another module, or may otherwise access a grooming cost of a selected region. A region of the cache 102 is a physical or logical grouping or segment of the storage media 110 of the cache. For example, in various embodiments, a region of the cache 102 may include a block, a sector, a page, a logical block, a logical page, a physical erase block, a logical erase block, a packet, an ECC chunk, or another logical or physical division of the cache 102. In one embodiment, the grooming cost module 602 examines a grooming cost of the same size and type of region that a grooming or garbage collection process grooms, such as a physical

erase block, a logical erase block, or the like. One embodiment of a grooming or garbage collection process is described below with regard to the groomer module 702 of Figure 4.

A grooming cost for a region, in one embodiment, includes an estimate or other representation of a cost of evicting the region of the cache 102. A grooming cost for a selected region may be relative to other regions such that the grooming cost may be compared between different regions to select a low cost region for grooming. A grooming cost, in certain embodiments, represents several different costs associated with grooming data of the cache 102. In one embodiment, a grooming cost for a selected region represents a cache efficiency cost of evicting data from the selected region, such as an opportunity cost of evicting the data, a likelihood of a cache miss in response to evicting the data, or the like. In another embodiment, a grooming cost for a selected region represents a storage efficiency cost of copying data forward or otherwise retaining data from the selected region. One of skill in the art, in light of this disclosure, will recognize other costs associated with grooming data that the grooming cost module 602 may factor in to the grooming cost of a selected region.

A cache efficiency cost factor of a grooming cost, in one embodiment, may be based on the types of data that the region stores, such as recent read data, frequently accessed read data, dirty write data, clean write data, and/or other types of data. For example, the cache efficiency cost of evicting frequently accessed data may be higher than the cache efficiency cost of evicting recent read data, due to a greater likelihood of a cache miss if the frequently accessed data is evicted. Similarly, in certain embodiments, the cache efficiency cost of evicting dirty write data may be much larger than the cache efficiency cost of evicting clean write data or read data, because the backing store 118 does not yet store dirty write data, and the cache efficiency cost of evicting dirty write data (i.e. losing the data) may be very high or infinite.

As described above, in certain embodiments the cache 102 may comprise multiple VSUs or other cache units and regions of the solid-state storage media 110 may store data for a plurality of cache units, intermingled in a log-based writing structure 122 of the cache 102 or the like. In such embodiments, a grooming cost and/or a cache efficiency cost component of a grooming cost, may account for one or more attributes of the cache units, such as a relative priority of cache units, allocated storage capacities for cache units, quality-of-service levels for cache units, or the like.

The grooming cost module 602 and/or another entity determining a grooming cost for a region, may determine an amount of data in the region associated with different cache units, and scale or weight the grooming cost according to the cache unit attributes of the different cache units. For example, the grooming cost module 602 may determine a higher grooming cost for

regions that store data from cache units with higher priorities, with higher quality-of-service levels, or the like, so that data from those cache units will be less likely to be evicted than data from cache units with lower priorities, quality-of-service levels, or the like. By including cache unit attributes in grooming costs, the grooming cost module 602 may determine grooming costs to maintain a relative priority of cache units, to maintain allocated storage capacities for cache units, to maintain quality-of-service levels for cache units, or the like.

A storage efficiency cost factor of a grooming cost, in one embodiment, may be based on one or more effects that grooming a selected region may have on storage operations of the cache 102. For example, storage efficiency costs may include the cost and/or the write amplification incurred by copying data from the selected region forward, such as dirty write data, frequently accessed data, or the like, the performance cost on the cache 102 of grooming the selected region, an impact (positive or negative) on the storage media 110 of the cache 102 by grooming the selected region, and/or other storage efficiency costs. Write amplification is the rewriting or moving of data during a grooming or garbage collection process, causing the same data originally written in response to a storage request to be written more than once. Write amplification can increase the number of writes of a storage device, consume write bandwidth of a storage device, reduce a usable lifetime of a storage device, and otherwise reduce performance of a storage device.

In one embodiment, the storage efficiency cost factors may include wear leveling of the physical storage media 110. In another embodiment, the storage efficiency cost factors include a frequency of access of a selected region, i.e., how "hot" or "cold" the selected region is. In one embodiment, the storage efficiency cost factors include a position of a selected region of data in the physical storage media 110 relative to other "hot" data. In another embodiment, the storage efficiency cost factors may include a determined reliability of a selected region, such as an Uncorrectable Bit Error Rate ("UBER"), a Correctable Bit Error Rates ("BER"), a Program / Erase ("PE") cycle count, a read frequency, and/or other non-volatile solid state storage specific attributes of the selected region. High BER, UBER, or PEs may be used as factors to lower the grooming cost and to increase the likelihood that the direct cache module 116a will groom a selected region having those characteristics so that such regions may more rapidly be moved out of service. By including one or more factors relating to reliability of a selected region in a grooming cost, such as an UBER, a BER, a PE cycle count, a read frequency, or the like, in one embodiment, a grooming process such as the groomer module 702 described below optimizes storage efficiency and reliability of the cache 102, in addition to recovering storage capacity.

In one embodiment, the grooming cost of a selected region may be based at least partially on one or more counts or tallies of types of data in the selected region. A count or tally of data of a specific type may comprise a number or amount of blocks, packets, pages, or the like in the region of the specific type, data units (e.g., bytes, kilobytes, megabytes, etc.) representing the amount of data in the region of the specific type, or the like. The grooming cost, in various embodiments, may be based at least partially on a count of frequently accessed data, a count of recently accessed data, a count of dirty write data, a count of clean write data, a count of user data, a count of metadata, and/or other data type counts.

In one embodiment, the grooming cost for a selected region is based at least partially on and/or accounts for a frequency count for the selected region. A frequency count for a selected region, in one embodiment, is a count of data in the selected region that is identified as frequently accessed. Data in a region, in one embodiment, is identified as frequently accessed in response to at least a predetermined number of read requests for the data. For example, in one embodiment, data may be identified as recent data upon being stored in the cache 102 and may be identified as frequently accessed data after one or more subsequent read requests (i.e., a predefined number) for the data. A frequency count, in various embodiments, may be specific to cached read data, may be specific to cached write data, and/or may include both read and write data. In a further embodiment, the frequency count for a selected region is cleared periodically such that the frequency count represents data that has been accessed a predefined number of times during a predefined period of time. In one embodiment, the frequency count includes a map, bit array, bit field, or the like indicating which blocks, packets, pages, or other sub-regions of a selected region are identified as frequently accessed data.

In another embodiment, the grooming cost for a selected region is based at least partially on and/or accounts for an amount of cached dirty write data of the selected region. In one embodiment, the grooming cost module 602 identifies an amount of dirty write data in a selected region using a count, tally, map, bit array, bit field, or the like for dirty write data of the selected region. In further embodiments, the grooming cost for a selected region may be based at least partially on and/or account for an amount of clean write data of the selected region, an amount of recent data of the selected region, an amount of user data of the selected region, an amount of metadata of the selected region, and/or amounts of other types of data of the selected region.

In one embodiment, the grooming cost may account for amounts of several types of data stored in a selected region by scaling or weighing counts for the types of data and summing the results. For example, in certain embodiments, the grooming cost may include an amount of dirty write data multiplied by one or more scalars, an amount of clean write data multiplied by one or

more scalars, an amount of recent read data multiplied by one or more scalars, an amount of frequent read data multiplied by one or more scalars, and/or amounts of other types of data multiplied by one or more scalars. The scalars used in the grooming cost, in one embodiment, are selected to represent a cache efficiency cost, a storage efficiency cost, and/or another sub-cost of grooming data from a selected region. In one embodiment, grooming cost scalars may be predefined. In a further embodiment, a user may adjust one or more grooming cost scalars through an interface of the host device 114, a network interface, a configuration setting, a command line interface utility, a graphical user interface utility, or the like. In another embodiment, the grooming cost module 602 dynamically adjusts one or more grooming cost scalars during operation of the cache 102. For example, the grooming cost module 602 may adjust, update, or otherwise set values for grooming cost scalars based on a detected workload of the cache 102, storage requests received by the cache 102, or the like.

In one embodiment, the grooming candidate set module 604 adds a selected region to a grooming candidate set in response to the grooming cost for the selected region satisfying a grooming cost threshold. The grooming candidate set, in one embodiment, includes a plurality of regions of the cache 102. The regions of the cache 102 in the grooming candidate set, in one embodiment, may be ordered by grooming cost. In one embodiment, the grooming candidate set comprises a predetermined number ("N") of low cost regions of the cache 102. For example, the grooming candidate set module 604, in one embodiment, may track low cost regions in response to the grooming cost module 602 periodically examining selected regions of the cache, adding regions that satisfy a grooming cost threshold to the grooming candidate set.

In one embodiment, the grooming cost of a selected region satisfies the grooming cost threshold by being lower than the grooming cost of at least one region of the grooming candidate set. In certain embodiments, where the grooming candidate set is ordered by grooming cost, the grooming candidate set module 604 compares the grooming cost of a selected region to the grooming cost of a grooming candidate region in a highest grooming cost position in the grooming candidate set to determine whether the grooming cost of the selected region satisfies the grooming cost threshold.

In embodiments where the grooming candidate set is ordered by grooming cost, the low cost module 606 described below readily selects the grooming candidate region in the lowest grooming cost position. Advantageously, the grooming candidate set provides a predetermined low cost region for data recovery by the groomer module 702 described below with regard to Figure 4. The grooming candidate set module 604, in certain embodiments, inserts a selected

region into the grooming candidate set at a grooming cost ordered position so that the grooming candidate set remains ordered by grooming cost upon insertion of the selected region.

In further embodiments, the grooming cost may satisfy the grooming cost threshold by being lower than a predefined grooming cost threshold for the grooming candidate set, lower than or equal to the grooming cost of a grooming candidate of the grooming candidate set, lower than an average grooming cost for grooming candidates of the grooming candidate set, lower than a median grooming cost for the grooming candidate set, lower than an average of a predefined number of the most costly grooming costs, and/or by having another predefined relationship with one or more grooming costs of grooming candidate regions of the grooming candidate set.

In another embodiment, the grooming cost of a selected region satisfies the grooming cost threshold in response to an opening or empty position in the grooming candidate set. For example, in certain embodiments, if the direct cache module 116a has evicted a region from the grooming candidate set, if the grooming candidate set module 604 has not yet added a predefined number of regions to the grooming candidate set, or the like, the grooming candidate set module 604 may add a selected region to the grooming candidate set regardless of the selected region's grooming cost, or the like, to fill an opening in the grooming candidate set.

In one embodiment, the grooming candidate set module 604 maintains data for regions of the cache 102 in the grooming candidate set. The grooming candidate set module 604, in one embodiment, maintains an indicator, reference, pointer, and/or other representation of each region in the grooming candidate set in a data structure for the grooming candidate set, such as a linked-list, a table, an array, or the like. In a further embodiment, the grooming candidate set module 604 may maintain a frequency count, a grooming cost, other data counts, and/or other metadata associated with a region of the grooming candidate set. For example, in certain embodiments, the grooming candidate set module 604 may preserving a copy of a frequency count and/or another data count for a selected region after the count has been cleared from the selected region.

In one embodiment, the low cost module 606 defines a low cost region within the grooming candidate set that the grooming candidate set module 604 maintains. The low cost module 606 defines, identifies, or otherwise designates a low cost region that satisfies one or more low cost parameters. For example, in certain embodiments, the low cost module 606 selects a region with the lowest grooming cost of the members of the grooming candidate set as the low cost region by comparing the grooming costs of members of the grooming candidate set. In a further embodiment, the low cost module 606 may select a plurality of regions as low cost

regions, such as regions with grooming costs below a low cost threshold, or the like. In one embodiment, the low cost module 606 designates the low cost region for grooming, eviction, garbage collection, or the like to recover storage capacity of the low cost region. In certain embodiments, where the grooming candidate set is ordered by grooming cost, the low cost module 606 defines a low cost region by selecting a grooming candidate region in a lowest grooming cost position in the grooming candidate set as the low cost region.

In one embodiment, to select a low cost region, the low cost module 606 compares stored grooming costs for regions of the grooming candidate set, such as grooming costs that the grooming candidate set module 604 stores, without updating or re-determining the grooming costs. Comparing stored grooming costs instead of updating grooming costs with each selection, in certain embodiments, is more efficient than re-determining each grooming cost for each low cost region selection. In other embodiments, the low cost module 606 may update or re-determine one or more grooming costs, such as a grooming cost for a low cost region (to ensure that the low cost region still has the lowest grooming cost), grooming costs for members of the grooming candidate set, or the like to define a low cost region.

In one embodiment, the low cost module 606 defines a low cost region from the grooming candidate set in response to the grooming candidate set module 604 adding a selected region to the grooming candidate set. For example, in certain embodiments, in response to a request for a low cost region to groom or another storage capacity recovery event, the grooming cost module 602 may examine grooming costs for selected regions until the grooming candidate set module 604 adds a selected region to the grooming candidate set and the low cost module 606 may designate or otherwise define a low cost region to satisfy the request for a low cost region once the grooming candidate set module 604 adds the selected region to the grooming candidate set, or the like. In a further embodiment, the low cost module 606 may define a low cost region in anticipation of a storage capacity recovery event, or the like.

Figure 4 depicts another embodiment of the direct cache module 116b. In the depicted embodiment, the direct cache module 116b includes the block I/O emulation layer 506, the direct interface layer 508, the grooming cost module 602, the grooming candidate set module 604, and the low cost module 606 as described above with regard to Figures 2 and 3. The direct cache module 116b, in the depicted embodiment, further includes a groomer module 702, a grooming clock module 704, a frequency count module 706, a recovery event module 708, a write request module 710, a cache write module 712, a destage module 714, a direct mapping module 716, a dirty indicator module 718, a read request module 720, and a backing store interface module 722. The direct cache module 116b, in certain embodiments, may be substantially similar to the direct

cache module 116 of Figures 1A and 1B, the direct cache module 116 of Figure 2, and the direct cache module 116a of Figure 3.

In one embodiment, the groomer module 702 recovers storage capacity of a low cost region that the low cost module 606 defines. As described above with regard to Figure 3, in various embodiments, the grooming cost module 602, the grooming candidate set module 604, and the low cost module 606 cooperate to examine grooming costs of selected regions of the cache 102, maintain a grooming candidate set of the N lowest grooming cost regions, and defines a region from the grooming candidate set as a low cost region. Defining a low cost region based on periodic examination of grooming costs for each region of storage and maintaining a grooming candidate set, in one embodiment, allows the direct cache module 116b to select a low cost region for grooming with the lowest known grooming cost without the overhead and time of re-determining grooming costs for each region with every low cost region selection.

The groomer module 702, in certain embodiments, recovers storage capacity of a low cost region in response to a storage capacity recovery event, as described below with regard to the recovery event module 708. In the depicted embodiment, the groomer module 702 includes a dirty data module 724, a frequent data module 726, and an eviction module 728. In one embodiment, the groomer module 702 recovers storage capacity of a low cost region by copying forward or otherwise retaining certain types of data from the low cost region to optimize cache efficiency and by evicting or otherwise clearing other types of data from the low cost region to optimize storage efficiency of the cache 102.

In one embodiment, the groomer module 702 relocates or otherwise retains certain types of valid data that is in a low cost region to preserve the valid data, to service storage requests, or the like. The groomer module 702, in certain embodiments, uses the dirty data module 724 to relocate or copy forward dirty data that has not been destaged upon grooming the dirty data of a low cost region to preserve the dirty data. In another embodiment, the groomer module 702 may selectively relocate or copy forward clean data that has already been destaged, such as clean data identified as frequent data, or the like. In another embodiment, instead of relocating or copying forward dirty data of a low cost region, the destage module 714 described below destages the dirty data in response to the low cost module 606 selecting the low cost region for grooming, or the like.

In one embodiment, the groomer module 702 uses the frequent data module 726 to relocate or copy forward data identified as frequently accessed data from a low cost region, to retain the frequently accessed data in the cache 102. The frequent data module 726, in various embodiments, may retain cached read data identified as frequently accessed, cached clean write

data identified as frequently accessed, or both cached read data and clean write data identified as frequently accessed. The frequent data module 726, in a further embodiment, identifies frequently accessed data based on a frequency count, such as a map, bit field, bit array, frequent data flags, and/or other frequent data indicators. In one embodiment, the frequent data module 726 identifies frequently accessed data of a low cost region to copy forward or otherwise retain using a frequency count that the grooming candidate set 604 maintains for the low cost region. In certain embodiments, using a frequency count maintained by the grooming candidate set module 604 may reduce accesses to the physical storage media 110 of the low cost region and/or may otherwise increase storage efficiency of the cache 102.

In one embodiment, the frequent data module 726 may handle frequently accessed data of a low cost region differently based on a grooming mode of the groomer module 702. In certain embodiments, the groomer module 702 may operate in a plurality of modes, such as a low pressure groom mode, a high pressure groom mode, or the like. For example, the groomer module 702 may transition from a low pressure groom mode to a high pressure groom mode in response to a lack of available storage capacity in the cache 102, a percentage of data marked as invalid reaching a predefined threshold level, performance of the cache 102 crossing a threshold value, in response to a storage capacity recovery event described below with regard to the recovery event module 708, or the like.

The frequent data module 726, in one embodiment, retains cached data identified as frequently accessed data when the groomer module 702 is in a low pressure groom mode and the frequent data module 726 allows the eviction module 728 to evict cached data identified as frequently accessed data when the groomer module 702 is in a high pressure groom mode. By processing frequently accessed data differently in a high pressure groom mode than in a low pressure groom mode, in certain embodiments, the groomer module 702 optimizes cache efficiency by retaining frequently accessed data when there is low grooming pressure, while optimizing storage capacity recovery when there is high grooming pressure.

The eviction module 728, in one embodiment, evicts, trims, erases, or otherwise clears data from a low cost region to recover the storage capacity of the low cost region. Erasing data from a low cost region without relocating the data evicts the data from the cache 102. In one embodiment, the groomer module 702 and/or the eviction module 728 clears or erases all data in a low cost region of the physical storage media 110 in response to the dirty data module 724 and/or the frequent data module 726 retaining or copying forward dirty write data and/or frequent data from the low cost region, evicting data that is not retained or copied forward from the cache 102. In a further embodiment, the dirty data module 724 and/or the frequent data

module 726 may mark data to be retained as valid and/or the eviction module 728 may mark data to be evicted as invalid, and a separate garbage collection process of the groomer 702 may retain the valid data and erase or otherwise clear the invalid data.

In one embodiment, the groomer module 702 includes or is part of an autonomous garbage collector system that operates within the cache 102. This allows the cache 102 to manage data so that data is systematically spread throughout the solid-state storage media 110, or other physical storage media, to improve performance, data reliability and to avoid overuse and underuse of any one location or area of the solid-state storage media 110 and to lengthen the useful life of the solid-state storage media 110.

The groomer module 702, upon recovering a low cost region of the physical storage media 110, allows the cache 102 to re-use the region of the physical storage media 110 to store different data. In one embodiment, the groomer module 702 adds the recovered region of physical storage media 110 to an available storage pool for the cache 102, or the like. The groomer module 702 and/or the eviction module 728, in one embodiment, erase existing data in a low cost region. In a further embodiment, the groomer module 702 and/or the eviction module 728 allow the cache 102 to overwrite existing data in a low cost region. Whether or not the groomer module 702, in various embodiments, erases existing data in a low cost region may depend on the nature of the physical storage media 110. For example, Flash media requires that cells be erased prior to reuse where magnetic media such as hard drives does not have that requirement. In an embodiment where the groomer module 702 does not erase data in a low cost region, but allows the cache 102 to overwrite data in the low cost region, the groomer module 702, in certain embodiments, may mark the data in the low cost region as unavailable to service read requests so that subsequent requests for data in the low cost region return a null result or an empty set of data until the cache 102 overwrites the data.

In one embodiment, the groomer module 702 recovers storage capacity of the cache 102 one or more storage regions at a time, such as a series of low cost regions defined by the low cost module 606, or the like. A storage region, in one embodiment, is a logical or physical erase block or other predefined division. For flash memory, an erase operation on an erase block writes ones to every bit in the erase block. This is a lengthy process compared to a program operation which starts with a location being all ones, and as data is written, some bits are changed to zero. However, where the solid-state storage 110 is not flash memory or has flash memory where an erase cycle takes a similar amount of time as other operations, such as a read or a program, the eviction module 728 may erase the data of a storage division as it evicts data, instead of a separate garbage collection process of the groomer module 702.

In one embodiment, allowing the eviction module 728 to mark data as invalid rather than actually erasing the data and allowing the groomer module 702 to recover the physical media associated with invalid data, increases efficiency because, as mentioned above, for flash memory and other similar storage an erase operation takes a significant amount of time. Allowing the groomer module 702 to operate autonomously and opportunistically within the cache 102 provides a way to separate erase operations from reads, writes, and other faster operations so that the cache 102 operates very efficiently.

In one embodiment, the grooming clock module 704 selects regions of the cache 102 for the grooming cost module 602 to examine. The grooming clock module 704, in a further embodiment, selects next regions of the cache 102 for the grooming cost module 602 to examine successively until the grooming candidate set module 604 determines that a grooming cost for a selected region satisfies a grooming cost threshold and adds the selected region to the grooming candidate set. The grooming clock module 704, in certain embodiments, organizes regions of the cache 102 in a circular, repeating, and/or cyclical data structure so that the grooming clock module 704 periodically selects each region of the cache 102 in turn for the grooming cost module 602 to examine. In one embodiment, the circular arrangement of regions of the cache 102 may be referred to as a grooming clock.

The grooming clock module 704, in one embodiment, may arrange regions of the cache 102 using a circular or repeating data structure with identifiers, pointers, references, or the like for each region. In another embodiment, the grooming clock module 704 may arrange regions of the cache 102 using a known physical or logical order of the regions, with or without a dedicated data structure for the regions, or the like. For example, in one embodiment, the grooming clock module 704 may maintain a linked-list or other data structure of regions of the cache with pointers or other references between regions in the list forming a logical circle. In another example embodiment, the grooming clock module 704 may select regions of the cache 102 in order, for example based on an erase block identifier or the like, and may cycle through the regions, selecting a first region after selecting a last region to continue the cycle, and so on, providing a circular, repeating pattern of selections.

In one embodiment, the order of regions in the circular arrangement is substantially constant, so that the grooming clock module 704 selects each region once for each period around the circular data structure. For example, the grooming clock module 704 may order regions in address order, erase block identifier order, or the like so that the order of the regions is constant regardless of the relative positions of the regions in a log 122 of the cache 102. In another embodiment, the grooming clock module 704 may reorder the regions in the circular data

structure as the regions are groomed, removed from a log 122 of the cache 102, recycled, and used in a new position in the log 122, ordering the regions in log order, or the like.

In one embodiment, the grooming clock module 704 initially selects a region for the grooming cost module 602 to examine in response to a storage capacity recovery event as described below with regard to the recovery event module 708. The grooming clock module 704, in a further embodiment, continues to select successive next regions for the grooming cost module 602 to examine until a grooming cost for a selected region satisfies a grooming cost threshold, the grooming candidate set module 604 adds the selected region to the grooming candidate set, and the low cost module 606 defines a low cost region for the groomer 702 to groom. In another embodiment, the grooming clock module 704 may continue to select next regions for the grooming cost module 602 to examine until the groomer module 702 recovers a predefined amount of storage capacity, or the like. In certain embodiments, the grooming clock module 704 may maintain a pointer or other reference to a currently selected region to maintain a current position in the circular arrangement of regions between storage capacity recovery events.

In one embodiment, the frequency count module 706 provides a frequency count for a selected region to the grooming cost module 602 to use in determining a grooming cost for the selected region. As described above, in various embodiments, a frequency count may be relative to the life of cached data in the cache 102, relative to a predefined time period such as one or more periods of the circular grooming data structure, or the like. In one embodiment, the frequency count module 706 identifies data as frequently accessed and includes the data in a frequency count in response to a predefined number of accesses of the data in a predefined time period. For example, in one embodiment, the frequency count module 706 identifies data as recent data in response to a first access or read of the data and identifies the data as frequent data in response to one or more subsequent accesses or reads of the data, or the like.

The storage controller 104, in one embodiment, provides a frequency count for a selected region to the frequency count module 706 at request of the frequency count module 706. In another embodiment, the frequency count module 706 counts or tallies blocks of data identified as frequently accessed data for a selected region. The frequency count module 706 may count or tally frequently accessed data by traversing a map, a bit field, a bit array, frequent data flags, or the like, for a selected region and incrementing a counter for each block of data identified as frequently accessed, or the like. In one embodiment, the storage controller 104 provides a map, a bit field, a bit array, frequent data flags, or the like for a selected region to the frequency count module 706. In another embodiment, the frequency count module 706 maintains a map, a bit

field, a bit array, frequent data flags, or the like indicating which data in a selected region is frequently accessed data.

In one embodiment, the frequency count module 706 clears a frequency count for a region after the grooming clock module 704 selects the region and the grooming cost module 602 determines and/or examines a grooming cost for the region. The grooming candidate set module 604, in certain embodiments, retains a copy of the frequency count for a region that the grooming candidate set module 604 adds to the grooming candidate set. As described above with regard to the grooming candidate set module 604 of Figure 3, in certain embodiments, the frequency count for a region includes a map, a bit field, a bit array, frequent data flags, or the like indicating which blocks of data in the region are identified as frequently accessed.

By clearing the frequency counts for regions of the cache 102 each period of the circular grooming data structure, in certain embodiments, the frequency counts that the frequency count module 706 provides are relative to a number of blocks or other amount of data in a region that have been accessed at least a predetermined number of times during a predefined time period, or period of the circular grooming data structure. In a further embodiment, due to the frequency count module 706 periodically clearing frequency counts, each subsequent grooming cost that the grooming cost module 602 examines for a selected region may be based at least partially on a new frequency count for the selected region that is specific to the current period of the circular grooming data structure. In other embodiments, the frequency count module 706 may clear frequency counts every N periods of the circular grooming data structure, for example every other period, every third period, every fourth period, etc.

In one embodiment, the recovery event module 708 detects and/or initiates a storage capacity recovery event for the cache 102. A storage capacity recovery event is an event that triggers recovery of storage capacity of the cache 102. The recovery event module 708, in certain embodiments, monitors the storage controller 104 and/or the cache 102 for the occurrence of a storage capacity recovery event. In another embodiment, the recovery event module 708 may receive a storage capacity recovery event notification, such as a storage capacity recovery request, or the like, from another module, from the storage controller 104, from the host device 114, or the like.

In one embodiment, a storage capacity recovery event includes a grooming pressure for the cache exceeding a predefined grooming pressure threshold. In another embodiment, a storage capacity recovery event may include an available storage capacity of the cache 102 falling below a predefined available capacity threshold. A storage capacity recovery event, in a further embodiment, may include a percentage of data marked as invalid in the cache 102

reaching a predefined invalid data threshold level. In various other embodiments, a storage capacity recovery event may include a consolidation of valid data, an error detection rate reaching a threshold value, performance crossing a threshold value, a scheduled garbage collection or grooming cycle, or the like.

In one embodiment, the write request module 710 detects one or more write requests to store data on the backing store 118. The write request module 710 may detect a write request by receiving the write request directly, detecting a write request sent to a different module or entity (such as detecting a write request sent directly to the backing store 118), or the like. In one embodiment, the host device 114 sends the write request. The direct cache module 116b, in one embodiment, represents itself to the host device 114 as a storage device, and the host device 114 sends write requests directly to the write request module 710.

A write request, in one embodiment, includes data that is not stored on the backing store 118. Data that is not stored on the backing store 118, in various embodiments, includes new data not yet stored on the backing store 118, modifications to data that is stored on the backing store 118, and the like. The write request, in various embodiments, may directly include the data, may include a reference, a pointer, or an address for the data, or the like. For example, in one embodiment, the write request includes a range of addresses indicating data to be stored on the backing store 118 by way of a Direct Memory Access ("DMA") or Remote DMA ("RDMA") operation. In a further embodiment, a single write request may include several different contiguous and/or noncontiguous ranges of addresses or blocks. In a further embodiment, the write request includes one or more destination addresses for the data, such as logical and/or physical addresses for the data on the cache 102 and/or on the backing store 118. The write request module 710 and/or another cooperating module, in various embodiments, may retrieve the data of a write request directly from the write request itself, from a storage location referenced by a write request (i.e., from a location in system memory or other data storage referenced in a DMA or RDMA request), or the like.

The cache write module 712, in one embodiment, writes data of a write request to the cache 102 to cache the data in the cache 102. The cache write module 712, in another embodiment, caches the data of the write request to the cache 102 at one or more logical addresses of the cache 102 corresponding to one or more backing store addresses of the write request. In one embodiment, the cache write module 712 caches the data to the cache 102 by appending the data to a sequential, log-based writing structure 122 preserved in the physical storage media 110 of the cache 102 at an append point 128. The cache write module 712, in one embodiment, may return one or more physical addresses corresponding to a location of the

append point at which the data was appended to a direct mapping module such as the direct mapping module 716 described below, which maps the one or more logical addresses of the cache 102 to the one or more physical addresses corresponding to the append point 128.

The destage module 714, in one embodiment, destages cached data from the cache 102 to the backing store 118. The destage module 714 destages data to the backing store 118 by copying, writing, storing, or otherwise persisting the data in the backing store 118. The destage module 714 destages dirty write data that the backing store 118 does not yet store. Data that is stored in the cache 102 that is not yet stored in the backing store 118 is referred to as "dirty" data. Once the backing store 118 stores data, the data is referred to as "clean." The destage module 714 destages or cleans data in the cache 102 by writing the data to the backing store 118.

As discussed in greater detail below with regard to the dirty indicator module 718, in certain embodiments, the destage module 714 accesses one or more dirty data indicators to determine which data in the cache 102 is dirty and is a candidate for destaging. In various embodiments, a dirty data indicator may include one or more flags, one or more bit fields, one or more bit arrays, or the like. Dirty data indicators, in various embodiments, may be stored in a mapping structure, in a reverse map, in volatile memory of the cache 102 or the host device 114, in a region of data such as an erase block or a packet, and/or in other data storage accessible to the destage module 714. In a further embodiment, the destage module 714 may store dirty indicators on volatile memory and may also store at least enough information to reconstruct the dirty indicators in the storage media 110 of the cache 102. In one embodiment, the destage module 714 updates one or more dirty data indicators in response to successfully destaging data to the backing store 118 so that the one or more dirty data indicators indicate that the destaged data is clean.

The destage module 714, in one embodiment, may determine an address for selected destage data in the backing store 118 based on a write request corresponding to the data. In a further embodiment, the destage module 714 determines an address for destage data in the backing store 118 based on a logical address of the data in the cache 102, based on a cache index, a mapping structure, or the like. In another embodiment, the destage module 714 uses a reverse map or the like to determine an address for destage data in the backing store 118 based on a physical address of the data in the cache 102.

The destage module 714, in one embodiment, writes data to the backing store 118 based on a write policy. In one embodiment, the destage module 714 uses a write-back write policy, and does not immediately write data of a write request to the backing store 118 upon detecting the write request. Instead, the destage module 714, in one embodiment, performs an

opportunistic or "lazy" write, destaging data to the backing store 118 when the low cost module 606 defines a region associated with the data as a low cost region, when the cache 102 and/or the direct cache module 116b has a light load, when available storage capacity in the cache 102 falls below a threshold, to satisfy a destaging pressure or target destage rate, or the like. In certain write-back embodiments, the destage module 714 may read data from the cache 102 and write the data to the backing store 118.

In another embodiment, instead of cleaning data according to a write-back write policy, the destage module 714 uses a write-through policy, performing a synchronous write to the backing store 118 for each write request that the write request module 710 receives. The destage module 714, in one embodiment, transitions from a write-back to a write-through write policy in response to a predefined error condition, such as an error or failure of the cache 102, or the like.

In one embodiment, the destage module 714 does not invalidate or evict destaged data from the cache 102, but destaged data remains in the cache 102 to service read requests until the destaged data is evicted from the cache by a separate eviction process. In a further embodiment, the destage module 714 may invalidate, clear, or evict destaged data from the cache 102 once the backing store 118 stores the data. In certain embodiments, evicting data upon destaging may lead to an increase in cache misses, but may also increase a speed or efficiency of garbage collection/grooming of the cache 102 by the groomer module 702.

The direct mapping module 716, in one embodiment, directly maps logical or physical addresses of the backing store 118 ("backing store addresses") to logical addresses of the cache 102 and directly maps logical addresses of the cache 102 to the backing store addresses of the backing store 118. As used herein, direct mapping of addresses means that for a given address in a first address space there is exactly one corresponding address in a second address space with no translation or manipulation of the address to get from an address in the first address space to the corresponding address in the second address space. The direct mapping module 716, in a further embodiment, maps backing store addresses to logical addresses of the cache 102 such that each backing store 118 address has a one to one relationship with a logical address of the cache 102. In one embodiment, the logical addresses of the cache 102 are independent of the physical addresses of the physical storage media 110 for the cache 102 and the physical addresses of the physical storage media 110 of the cache 102 are fully associative with backing store addresses of the backing store 118.

In one embodiment, the direct mapping module 716 maps the backing store addresses directly to logical addresses of the cache 102 so that the backing store addresses of the backing store 118 and the logical addresses of the cache 102 are equal or equivalent. In one example of

this embodiment, the backing store addresses and the logical addresses of the cache 102 share a lower range of the logical address space of the cache 102, such as addresses between about 0-2^{32}, or the like.

In one embodiment, the direct mapping module 716 directly maps logical addresses of the cache 102 to physical addresses and/or locations on the physical storage media 110 of the cache 102. In a further embodiment, the direct mapping module 716 uses a single mapping structure to map backing store addresses to logical addresses of the cache 102 and to map logical addresses of the cache 102 to locations on the physical storage media 110 of the cache 102. The mapping structure, in various embodiments, may include a B-tree, B*-tree, B+-tree, a CAM, a binary tree, a hash table, an index, an array, a linked-list, a look-up table, or another mapping data structure.

Use of a B-tree as the mapping structure in certain embodiments, is particularly advantageous where the logical address space presented to the client is a very large address space (such as 2^{64} addressable blocks or the like – which may or may not be sparsely populated). Because B-trees maintain an ordered structure, searching such a large space remains very fast. For example, in one embodiment, the mapping structure includes a B-tree with multiple nodes and each node may store several entries. In the example embodiment, each entry may map a variable sized range of logical addresses of the cache 102 to a location (such as a starting location) on the physical storage media 110 of the cache 102. Furthermore, the number of nodes in the B-tree may vary as the B-tree grows wider and/or deeper.

In one embodiment, the mapping structure of the direct mapping module 716 only includes a node or entry for logical addresses of the cache 102 that are associated with currently cached data in the cache 102. In this embodiment, membership in the mapping structure represents membership in the cache 102. The direct mapping module 716, in one embodiment, adds entries, nodes, and the like to the mapping structure as data is stored in the cache and removes entries, nodes, and the like from the mapping structure in response to data being evicted, cleared, trimmed, or otherwise removed from the cache 102.

Similarly, membership in the mapping structure may represent valid allocated blocks on the solid-state storage media 110. The solid-state storage controller 104 (and/or the direct mapping module 716), in one embodiment, adds entries, nodes, and the like to the mapping structure as data is stored on the solid-state storage media 110 and removes entries, nodes, and the like from the mapping structure in response to data being invalidated cleared, trimmed, evicted, or otherwise removed from the solid-state storage media 110. In the case where the mapping structure is shared for both cache management and data storage management on the

solid-state storage media 110, the dirty indicator module 718 described below, in certain embodiments, may also track whether the data is dirty or not to determine whether the data is persisted on the backing store 118. The address order module 724, in one embodiment, may also traverse the mapping structure to locate ranges of data in backing store address order, may request ranges of data in backing store address order from the direct mapping module 716, or the like.

In a further embodiment, the mapping structure of the direct mapping module 716 may include one or more nodes or entries for logical addresses of the cache 102 that are not associated with currently stored data in the cache 102, but that are mapped to addresses of the backing store 118 that currently store data. The nodes or entries for logical addresses of the cache 102 that are not associated with currently stored data in the cache 102, in one embodiment, are not mapped to locations on the physical storage media 110 of the cache 102, but include an indicator that the cache 102 does not store data corresponding to the logical addresses. The nodes or entries, in a further embodiment, may include information that the data resides in the backing store 118. For example, in certain embodiments, the mapping structure of the direct mapping module 716 may include nodes or entries for read misses, data of which the backing store 118 stores but the cache 102 does not currently store.

Nodes, entries, records, or the like of the mapping structure, in one embodiment, may include information (such as physical addresses, offsets, indicators, etc.) directly, as part of the mapping structure, or may include pointers, references, or the like for locating information in memory, in a table, or in another data structure. The direct mapping module 716, in one embodiment, optimizes the mapping structure by monitoring the shape of the mapping structure, monitoring the size of the mapping structure, balancing the mapping structure, enforcing one or more predefined rules with regard to the mapping structure, ensuring that leaf nodes of the mapping structure are at the same depth, combining nodes, splitting nodes, and/or otherwise optimizing the mapping structure.

The direct mapping module 716, in one embodiment, stores the mapping structure on the solid-state storage media 110 of the cache 102. By storing the mapping structure on the cache 102, in a further embodiment, the mapping of addresses of the backing store 118 to the logical addresses of the cache 102 and/or the mapping of the logical addresses of the cache 102 to locations on the physical storage media 110 of the cache 102 are persistent, even if the cache 102 is subsequently paired with a different host device 114. In one embodiment, the backing store 118 is also subsequently paired with the different host device 114. In a further embodiment, the cache 102 rebuilds or restores at least a portion of data from the backing store 118 on a new

storage device associated with the different host device 114, based on the mapping structure and data stored on the cache 102.

In one embodiment, the direct mapping module 716 determines one or more factors of the grooming cost of a selected region for the grooming cost module 602 based on a history of access to the mapping structure. The direct mapping module 716, in a further embodiment, identifies areas of high frequency, "hot," use and/or low frequency, "cold," use by monitoring accesses of branches or nodes in the mapping structure. The direct mapping module 716, in a further embodiment, determines a count or frequency of access to a branch, directed edge, or node in the mapping structure. In one embodiment, a count associated with each node of a b-tree like mapping structure may be incremented for each I/O read operation and/or each I/O write operation that visits the node in a traversal of the mapping structure. Of course, in certain embodiments, separate read counts and write counts may be maintained for each node. Certain counts may be aggregated to different levels in the mapping structure in other embodiments.

The direct mapping module 716, the grooming cost module 602, and/or the groomer module 702, in one embodiment, share information to increase the efficiency of the cache 102, to reduce cache misses, to make intelligent eviction decisions, and the like. In one embodiment, the direct mapping module 716 tracks or monitors a frequency that I/O requests access logical addresses in the mapping structure. The direct mapping module 716, in a further embodiment, stores the access frequency information in the mapping structure, communicates the access frequency information to the grooming cost module 602, or the like. The direct mapping module 716, in another embodiment, may track, collect, or monitor other usage/access statistics relating to the logical to physical mapping of addresses for the cache 102 and/or relating to the mapping between the logical address space of the cache 102 and the address space of the backing store 118, and may share that data with the grooming cost module 602.

One example of a benefit of sharing information between the destage module 714, the direct mapping module 716, the grooming cost module 602, and the groomer module 702, in certain embodiments, is that write amplification can be reduced. As described above, in one embodiment, the groomer module 702 copies certain valid data in a low cost region forward to the current append point 128 of the log-based append-only writing structure 122 of the cache 102 before recovering the physical storage capacity of the low cost region. By cooperating with the destage module 714, the direct mapping module 716, and/or with the grooming cost module 602, in one embodiment, the groomer module 702 may clear certain valid data from a region without copying the data forward (for example because the grooming cost algorithm for the grooming cost module 602 indicates that the valid data is unlikely to be re-requested soon, giving the

region a low grooming cost), reducing write amplification, increasing available physical storage capacity and efficiency. The groomer module 702 can even clear valid user write data from an erase block, so long as the destage module 714 has destaged the data to the backing store 118.

For example, in one embodiment, the groomer module 702 preserves valid data with an access frequency in the mapping structure that is above a predefined threshold, and clears valid data from an erase block if the valid data has an access frequency below the predefined threshold, as described above with regard to the frequent data module 726. In a further embodiment, the eviction module 728 may mark certain data as conditionally evictable, conditionally invalid, or the like, and the groomer module 702 may evict the conditionally invalid data based on an access frequency or other data that the direct mapping module 716 provides. In another example, the destage module 714, the direct mapping module 716, the grooming cost module 602, and the groomer module 702 may cooperate such that valid data that is in the cache 102 and is dirty gets stored on the backing store 118 by the destage module 714 rather than copied to the front of the log 122, or the like.

Those of skill in the art will appreciate a variety of other examples and scenarios in which the modules responsible for managing the non-volatile storage media 110 that uses a log-based append-only writing structure 122 can leverage the information available in the direct cache module 116b. Furthermore, those of skill in the art will appreciate a variety of other examples and scenarios in which the modules responsible for managing the cache 102 (destage module 714, direct mapping module 716, groomer module 702, and/or grooming cost module 602) can leverage the information available in solid-state controller 104 regarding the condition of the non-volatile storage media 110.

In one embodiment, the dirty indicator module 718 sets an indicator that the destage module 714 has destaged data to the backing store 118 to track which data is clean and which data is dirty. The dirty indicator module 718, in one embodiment, sets the indicator that the backing store 118 stores the data once the destage module 714 has successfully written the data to the backing store 118. Setting the indicator (dirty/clean indicator) that the backing store 118 stores the data, in one embodiment, prevents the destage module 714 from destaging data a second time once the destage module 714 has already destaged the data. In a further embodiment, setting the indicator that the backing store 118 stores the data may affect a grooming cost that the grooming cost module 602 examines for a region associated with the data, may alert a garbage collection or grooming process, such as the groomer module 702, that the data may be cleared from the cache 102, or the like.

In one embodiment, the dirty indicator module 718 sets an indicator that the backing store 118 stores data by marking the data as clean in the cache 102. In a further embodiment, the dirty indicator module 718 may set an indicator that the backing store 118 stores data by communicating an address of the data to the direct mapping module 716 or by sending a request to the direct mapping module 716 to update an indicator in a logical to physical mapping or other mapping structure. In another embodiment, the dirty indicator module 718 may set an indicator that the backing store 118 stores data by updating one or more indicators for a region of data in the cache 102, or the like. For example, in certain embodiments, the dirty indicator module 718 may maintain a map, bit field, or bit array for one or more regions of the cache 102 representing which data is dirty and which data is clean within the one or more regions. In the map, bit fields, or bit arrays, in one embodiment, each bit represents a block such as a packet, a page, a sector, a range of data, or the like within a region, with one binary state indicating that the block is dirty and the other binary state representing that the block is clean.

In one embodiment, the read request module 720 services read requests for data stored in the cache 102 and/or the backing store 118. The read request module 720, in one embodiment, detects a read request to retrieve requested data from the backing store 118. In a further embodiment, the read request module 720 receives read requests from the host device 114. A read request is a read command with an indicator, such as a logical address or range of logical addresses, of the data being requested. In one embodiment, the read request module 720 supports read requests with several contiguous and/or noncontiguous ranges of logical addresses, as discussed above with regard to the write request module 710.

In the depicted embodiment, the read request module 720 includes a read miss module 730 and a read retrieve module 732. The read miss module 730, in one embodiment, determines whether or not requested data is stored in the cache 102. The read miss module 730 may query the cache 102 directly, query the direct mapping module 716, query the mapping structure of the direct mapping module 716, or the like to determine whether or not requested data is stored in the cache 102.

The read retrieve module 732, in one embodiment, returns requested data to the requesting entity, such as the host device 114. If the read miss module 730 determines that the cache 102 stores the requested data, in one embodiment, the read retrieve module 732 reads the requested data from the cache 102 and returns the data to the requesting entity. The direct mapping module 716, in one embodiment, provides the read retrieve module 732 with one or more physical addresses of the requested data in the cache 102 by mapping one or more logical addresses of the requested data to the one or more physical addresses of the requested data.

If the read miss module 730 determines that the cache 102 does not store the requested data, in one embodiment, the read retrieve module 732 reads the requested data from the backing store 118, writes the requested data to the cache 102, and returns the requested data to the requesting entity. In one embodiment, the read retrieve module 732 writes the requested data to the cache 102 by appending the requested data to an append point 128 of a log-based writing structure 122 of the cache 102. In one embodiment, the read retrieve module 732 provides one or more physical addresses corresponding to the append point to the direct mapping module 716 with the one or more logical addresses of the requested data and the direct mapping module 716 adds and/or updates the mapping structure with the mapping of logical and physical addresses for the requested data. The read retrieve module 732, in one embodiment, writes the requested data to the cache 102 using and/or in conjunction with the cache write module 712.

In one embodiment, the read miss module 730 detects a partial miss, where the cache 102 stores one portion of the requested data but does not store another. A partial miss, in various embodiments, may be the result of eviction of the unstored data, a block I/O request for noncontiguous data, or the like. The read miss module 730, in one embodiment, reads the missing data or "hole" data from the backing store 118 and returns both the portion of the requested data from the cache 102 and the portion of the requested data from the backing store 118 to the requesting entity. In one embodiment, the read miss module 730 stores the missing data retrieved from the backing store 118 in the cache 102.

In one embodiment, the backing store interface module 722 provides an interface between the direct cache module 116b, the cache 102, and/or the backing store 118. As described above with regard to Figure 2, in various embodiments, the direct cache module 116b may interact with the cache 102 and/or the backing store 118 through a block device interface, a direct interface, a device driver on the host device 114, a storage controller, or the like. In one embodiment, the backing store interface module 722 provides the direct cache module 116b with access to one or more of these interfaces. For example, the backing store interface module 722 may receive read commands, write commands, and clear (or TRIM) commands from one or more of the cache write module 712, the direct mapping module 716, the read request module 720, the destage module 714, the groomer module 702, and the like and relay the commands to the cache 102 and/or the backing store 118. In a further embodiment, the backing store interface module 722 may translate or format a command into a format compatible with an interface for the cache 102 and/or the backing store 118.

In one embodiment, the backing store interface module 722 has exclusive ownership over the backing store 118 and the direct cache module 116b is an exclusive gateway to accessing the

backing store 118. Providing the backing store interface module 722 with exclusive ownership over the backing store 118 and preventing access to the backing store 118 by other routes obviates stale data issues and cache coherency requirements, because all changes to data in the backing store 118 are processed by the direct cache module 116b.

In a further embodiment, the backing store interface module 722 does not have exclusive ownership of the backing store 118, and the backing store interface module 722 manages cache coherency for the cache 102. For example, in various embodiments, the backing store interface module 722 may access a common directory with other users of the backing store 118 to maintain coherency, may monitor write operations from other users of the backing store 118, may participate in a predefined coherency protocol with other users of the backing store 118, or the like.

Figure 5A depicts one embodiment of a circular grooming data structure 800 and a grooming candidate set 820a. In the depicted embodiment, the circular grooming data structure 800 includes a plurality of regions 802 including "EB0" 802a – "EBX" 802o. The regions 802a-o, in the depicted embodiment, are in an erase block order. In the depicted embodiment, the regions 802a-o are embodied by logical erase blocks. In other embodiments, the regions 802a-o may include another type of region, as described above. The grooming clock module 704, in the depicted embodiment, maintains a selected region pointer 804, or clock hand, identifying a currently selected region. The currently selected region in Figure 5A is "EB3" 802d. The grooming clock module 704, in the depicted embodiment, rotates the selected region pointer 804 in a circular pattern 806 around the circular grooming data structure 800, selecting regions 802a-o successively in turn for each period of the circular grooming data structure 800. In the depicted embodiment, each region 802a-o includes a frequency count 808 and the currently selected region "EB3" 802d includes a grooming cost 810.

The grooming candidate set 820a, in the depicted embodiment, includes N entries 822a-g identifying the N lowest cost regions 802a-o of the cache 102. In the depicted embodiment, N=7 and the grooming candidate set 820a includes 7 entries 822a-g. In other embodiments, different values may be selected for N. The entries 822a-g, in the depicted embodiment, are ordered by grooming cost 810 from lowest cost entry 822a to highest cost entry 822g. The grooming candidate set 820a, in the depicted embodiment, stores a grooming cost 810 and a frequency count 808 for each entry 822a-g. In a further embodiment, the grooming candidate set 820a may further preserve a copy of a map, bit field, bit array, frequent data flags, or the like for a frequency count 808 identifying frequently accessed data of a corresponding region 802a-o.

In one embodiment, because the entries 822a-g are ordered by grooming cost, the grooming cost module 602 compares the grooming cost 810 of a selected region to the grooming cost 810 of the highest cost entry 822g to determine whether or not to add the selected region to the grooming candidate set 820a, or the like. The low cost module 606, in one embodiment, defines the region of the lowest cost entry 822a as the low cost region. In certain embodiments, ordering the grooming candidate set 820a by grooming cost 810 reduces the overhead of traversing the grooming candidate set 820a, comparing multiple grooming costs 810, or the like for the grooming cost module 602 and/or the low cost module 606, simplifying the process of adding a selected region to the grooming candidate set 820a and defining a low cost region.

Figure 5B depicts one embodiment of a region selection 830 and a grooming candidate set 820b. In the depicted embodiment, the grooming cost 810 of the selected region "EB3" 802d is "31." The grooming candidate set module 604, in the depicted embodiment, determines that the grooming cost 810 of "31" satisfies the grooming cost threshold by being lower than at least one region of the grooming candidate set 820b, so the grooming candidate set module 604 adds the selected region "EB3" 802d to the grooming candidate set 820b as the fifth entry 822e, in grooming cost order. The low cost module 606, in the depicted embodiment, defines region "EB6" 802g (depicted as the first entry 822a of the grooming candidate set 820a of Figure 5A) as the low cost region, removes region "EB6" 802g from the grooming candidate set 820b, and the groomer module 702 grooms region "EB6" 802g.

Figure 5C depicts another embodiment of a region selection 840 and a grooming candidate set 820c. In the depicted embodiment, the frequency count module 706 clears the frequency count 808 of the previously selected region "EB3" 802d and the grooming clock module 704 moves the selected region pointer 804 to the next selected region "EB4" 802e. In one embodiment, the grooming clock module 704 selects the next region "EB4" 802e in response to a storage capacity recovery event, or the like. In the depicted embodiment, the grooming candidate set module 604 determines that the grooming cost 810 of the next selected region "EB4" 802e does not satisfy the grooming cost threshold, because the grooming cost 810 of "50" is greater than the grooming costs 810 of regions represented by entries 822a-g of the grooming candidate set 820c. In one embodiment, described below with regard to Figure 5D, the grooming clock module 704 selects a next region as the selected region in response to the grooming cost 810 failing to satisfy the grooming cost threshold.

Figure 5D depicts a further embodiment of a region selection 850 and a grooming candidate set 820d. In the depicted embodiment, the frequency count module 706 clears the frequency count 808 of the previously selected region "EB4" 802e and the grooming clock

module 704 moves the selected region pointer 804 to the next selected region "EB5" 802f. In one embodiment, the grooming clock module 704 selects the next region "EB5" 802f because the grooming cost 810 of the previously selected region "EB4" 802e failed to satisfy the grooming cost threshold. In the depicted embodiment, the grooming candidate set module 604 determines that the grooming cost 810 of the next selected region "EB5" 802f also does not satisfy the grooming cost threshold, because the grooming cost 810 of "55" is greater than the grooming costs 810 of regions represented by entries 822a-g of the grooming candidate set 820c. In one embodiment, described below with regard to Figure 5D, the grooming clock module 704 selects a next region as the selected region in response to the grooming cost 810 failing to satisfy the grooming cost threshold.

Figure 5E depicts an additional embodiment of a region selection 860 and a grooming candidate set 802e. In the depicted embodiment, the frequency count module 706 clears the frequency count 808 of the previously selected region "EB5" 802f and the grooming clock module 704 selects a next region "EB6" 802g as the selected region. Because the low cost module 606 previously defined the selected region "EB6" 802g as the low cost region during the same cycle of the circular data structure 800, as described above with regard to Figure 5B, this is the first selection of region "EB6" 802g since the groomer module 702 groomed the region 802g. In the depicted embodiment, the grooming clock module 704 skips the region "EB6" 802g and selects a next region as the selected region, as described below with regard to Figure 5F. Skipping the region "EB6" 802g permits the region "EB6" 802g to potentially receive more requests for data of the region 802g which would increase the frequency count 808.

Figure 5F depicts another embodiment of a region selection 870 and a grooming candidate set 820f. In the depicted embodiment, the frequency count module 706 clears the frequency count 808 of the previously selected region "EB6" 802g and the grooming clock module 704 selects a next region "EB7" 802h as the selected region. The selected region "EB7" 802h, in the depicted embodiment, is already a member of the grooming candidate set 820f, so the grooming candidate set module 604 updates the grooming cost 810 of the entry 822c for the selected region "EB7" 802h. The low cost module 606 defines region "EB2" 802c (referenced by the first entry 822a in the grooming candidate set 820e of Figure 5E) as the low cost region. The grooming candidate set module 604 removes the low cost region "EB2" 802c from the grooming candidate set 820f and the groomer module 702 grooms the low cost region "EB2" 802c, leaving an empty entry 822g in the grooming candidate set 820f as "EB10" becomes the new low cost region in the lowest cost entry 822a position and the other grooming candidate regions from the grooming candidate set 820f each shift toward the lowest cost entry 822a.

Figure 5G depicts one more embodiment of a region selection 880 and a grooming candidate set 820g. In the depicted embodiment, the frequency count module 706 clears the frequency count 808 of the previously selected region "EB7" 802h. The grooming clock module 704, in response to a storage capacity recovery event or the like, selects a next region "EB8" 802i as the selected region. The grooming candidate set module 604, in the depicted embodiment, adds the selected region "EB8" 802i to the grooming candidate set 820g even though the grooming cost 810 of the selected region "EB8" 802i is greater than the grooming costs 810 of members of the grooming candidate set 820g, to fill the empty entry 822g.

Figure 6 depicts one embodiment of a mapping structure 900, a logical address space 920 of the cache 102, a combined logical address space 919 that is accessible to a storage client, a sequential, log-based, append-only writing structure 940, and a storage device address space 970 of the backing store 118. The log-based writing structure 940, in certain embodiments, is substantially similar to the log-based writing structure 122 described above with regard to Figure 1B. While a variety of data structures may be used to implement the writing structure 940, in one embodiment the writing structure 940 comprises a linked list of regions. As noted above, a region of the cache 102 is a physical or logical grouping or segment of the storage media 110 of the cache. For example, in various embodiments, a region of the cache 102 may include a block, a sector, a page, a logical block, a logical page, a physical erase block, a logical erase block, a packet, an ECC chunk, or another logical or physical division of the cache 102. The mapping structure 900, in one embodiment, is maintained by the direct mapping module 716. The mapping structure 900, in the depicted embodiment, is a B-tree, with several additional entries. Further, the nodes of the mapping structure 900 include direct references to physical locations in the cache 102. The mapping structure 900, in various embodiments, may be used either with or without a reverse map. In certain embodiments, the references in the mapping structure 900 may include alpha-numerical characters, hexadecimal characters, pointers, links, and the like.

The mapping structure 900, in the depicted embodiment, includes a plurality of nodes. Each node, in the depicted embodiment, is capable of storing two entries. In other embodiments, each node may be capable of storing a greater number of entries, the number of entries at each level may change as the mapping structure 900 grows or shrinks through use, or the like. In a further embodiment, each entry may store one or more indicators of whether the data corresponding to the entry is clean or dirty, valid or invalid, read data or write data, or the like.

Each entry, in the depicted embodiment, maps a variable length range of logical addresses of the cache 102 to a physical location in the storage media 110 for the cache 102. Further, while variable length ranges of logical addresses, in the depicted embodiment, are

represented by a starting address and an ending address, in other embodiments, a variable length range of addresses may be represented by a starting address and a length or by another representation. In one embodiment, the capital letters 'A' through 'M' represent a logical or physical erase block in the physical storage media 110 of the cache 102 that stores the data of the corresponding range of logical addresses. In other embodiments, the capital letters may represent other physical addresses or locations of the cache 102. In the depicted embodiment, the capital letters 'A' through 'M' are also depicted in the writing structure 940 which represents the physical storage media 110 of the cache 102. Although each range of logical addresses maps simply to an entire erase block, in the depicted embodiment, for simplicity of description, in other embodiments, a single erase block may store a plurality of ranges of logical addresses, ranges of logical addresses may cross erase block boundaries, and the like.

In the depicted embodiment, membership in the mapping structure 900 denotes membership (or storage) in the cache 102. In another embodiment, an entry may further include an indicator of whether the cache 102 stores data corresponding to a logical block/sector within the range of logical addresses, and/or other data. For example, in one embodiment, the mapping structure 900 may also map logical addresses of the backing store 118 to physical addresses or locations within the backing store 118, and in certain embodiments an entry may include an indicator that the cache 102 does not store the data and a physical address or location for the data on the backing store 118.

In the depicted embodiment, the root node includes entries 902, 904 with noncontiguous ranges of logical addresses. A "hole" exists at logical address "208" between the two entries 902, 904 of the root node. In one embodiment, a "hole" indicates that the cache 102 does not store data corresponding to one or more logical addresses corresponding to the "hole." In one embodiment, a "hole" may exist because the eviction module 716 evicted data corresponding to the "hole" from the cache 102. If the eviction module 716 evicted data corresponding to a "hole," in one embodiment, the backing store 118 still stores data corresponding to the "hole." In another embodiment, the cache 102 and/or the backing store 118 supports block I/O requests (read, write, trim, etc.) with multiple contiguous and/or noncontiguous ranges of addresses (i.e., ranges that include one or more "holes" in them). A "hole," in one embodiment, may be the result of a single block I/O request with two or more noncontiguous ranges of addresses. In a further embodiment, a "hole" may be the result of several different block I/O requests with address ranges bordering the "hole."

The root node 903 includes a single entry with a logical address range of "205-212," without the hole at logical address "208." If the entry of the root node 903 were a fixed size

cache line of a traditional cache, the entire range of logical addresses "205-212" would be evicted together. Instead, in the embodiment depicted in Figure 6, the eviction module 728 evicts data of a single logical address "208" and splits the range of logical addresses into two separate entries 902, 904. In one embodiment, the direct mapping module 716 may rebalance the mapping structure 900, adjust the location of a directed edge, root node, or child node, or the like in response to splitting a range of logical addresses. Similarly, in one embodiment, each range of logical addresses may have a dynamic and/or variable length, allowing the cache 102 to store dynamically selected and/or variable lengths of logical block/sector ranges.

In the depicted embodiment, similar "holes" or noncontiguous ranges of logical addresses exist between the entries 906, 908 of the node 909, between the entries 910, 912 of the left child node of the node 909, between entries 914, 916 of the node 818, and between entries of the node 918. In one embodiment, similar "holes" may also exist between entries in parent nodes and child nodes. For example, in the depicted embodiment, a "hole" of logical addresses "060-071" exists between the left entry 906 of the node 909 and the right entry 912 of the left child node of the node 909.

The "hole" at logical address "003," in the depicted embodiment, can also be seen in the logical address space 920 of the cache 102 at logical address "003" 930. The hash marks at logical address "003" 930 represent an empty location, or a location for which the cache 102 does not store data. In the depicted embodiment, storage device address "003" 980 of the storage device address space 970 does store data (identified as 'b'), indicating that the eviction module 728 evicted data from logical address "003" 930 of the cache 102. The "hole" at logical address 934 in the logical address space 920, however, has no corresponding data in storage device address 984, indicating that the "hole" is due to one or more block I/O requests with noncontiguous ranges, a trim or other deallocation command to both the cache 102 and the backing store 118, or the like.

The "hole" at logical address "003" 930 of the logical address space 920, however, in one embodiment, is not viewable or detectable to a storage client. In the depicted embodiment, the combined logical address space 919 represents the data that is available to a storage client, with data that is stored in the cache 102 and data that is stored in the backing store 118 but not in the cache 102. As described above, the read miss module 730 of Figure 4 handles misses and returns requested data to a requesting entity. In the depicted embodiment, if a storage client requests data at logical address "003" 930, the read miss module 730 will retrieve the data from the backing store 118, as depicted at address "003" 980 of the storage device address space 970, and return the requested data to the storage client. The requested data at logical address "003"

930 may then also be placed back in the cache 102 and thus logical address 930 would indicate 'b' as present in the cache 102.

For a partial miss, the read miss module 730 may return a combination of data from both the cache 102 and the backing store 118. For this reason, the combined logical address space 919 includes data 'b' at logical address "003" 930 and the "hole" in the logical address space 920 of the cache 102 is transparent. In the depicted embodiment, the combined logical address space 919 is the size of the logical address space 920 of the cache 102 and is larger than the storage device address space 980. In another embodiment, the direct cache module 116 may size the combined logical address space 919 as the size of the storage device address space 980, or as another size.

The logical address space 920 of the cache 102, in the depicted embodiment, is larger than the physical storage capacity and corresponding storage device address space 970 of the backing store 118. In the depicted embodiment, the cache 102 has a 64 bit logical address space 920 beginning at logical address "0" 922 and extending to logical address "2^{64}-1" 926. The storage device address space 970 begins at storage device address "0" 972 and extends to storage device address "N" 974. Storage device address "N" 974, in the depicted embodiment, corresponds to logical address "N" 924 in the logical address space 920 of the cache 102. Because the storage device address space 970 corresponds to only a subset of the logical address space 920 of the cache 102, the rest of the logical address space 920 may be shared with an additional cache 102, may be mapped to a different backing store 118, may store data in the cache 102 (such as a Non-volatile memory cache) that is not stored in the storage device 970, or the like.

For example, in the depicted embodiment, the first range of logical addresses "000-002" 928 stores data corresponding to the first range of storage device addresses "000-002" 978. Data corresponding to logical address "003" 930, as described above, was evicted from the cache 102 forming a "hole" and a potential cache miss. The second range of logical addresses "004-059" 932 corresponds to the second range of storage device addresses "004-059" 982. However, the final range of logical addresses 936 extending from logical address "N" 924 extends beyond storage device address "N" 974. No storage device address in the storage device address space 970 corresponds to the final range of logical addresses 936. The cache 102 may store the data corresponding to the final range of logical addresses 936 until the data backing store 118 is replaced with larger storage or is expanded logically, until an additional data backing store 118 is added, simply use the non-volatile storage capability of the cache 102 to indefinitely provide storage capacity directly to a storage client 504 independent of a backing store 118, or the like.

In a further embodiment, the direct cache module 116 alerts a storage client 504, an operating system, a user application 502, or the like in response to detecting a write request with a range of addresses, such as the final range of logical addresses 936, that extends beyond the storage device address space 970. The user may then perform some maintenance or other remedial operation to address the situation. Depending on the nature of the data, no further action may be taken. For example, the data may represent temporary data which if lost would cause no ill effects.

The sequential, log-based, append-only writing structure 940, in the depicted embodiment, is a logical representation of the log preserved in the physical storage media 110 of the cache 102. In a further embodiment, the backing store 118 may use a substantially similar sequential, log-based, append-only writing structure 940. In certain embodiments, the cache 102 stores data sequentially, appending data to the writing structure 940 at an append point 944. In certain embodiments, the cache 102 also maintains a pointer to the last region of storage in the writing structure 940 at a tail pointer 964. The tail pointer 964 indicates which region of the writing structure 940 was last added to the log-based writing structure 940. The head pointer 966 points to the region that is the first one in the writing structure 940, at the head, or beginning, of the writing structure 940. The cache 102, in a further embodiment, uses a storage space recovery process, such as the garbage collection module 714 that re-uses non-volatile storage media 110 storing deallocated, unused, or evicted logical blocks/sectors. Non-volatile storage media 110 storing deallocated, unused, or evicted logical blocks/sectors, in the depicted embodiment, is added to an available storage pool 946 for the cache 102. By evicting and clearing certain data from the cache 102, as described above, and adding the physical storage capacity corresponding to the evicted and/or cleared data back to the available storage pool 946, in one embodiment, the writing structure 940 is ring-like and has a theoretically infinite capacity.

In the depicted embodiment, the append point 944 progresses around the log-based, append-only writing structure 940 in a circular pattern 942. In one embodiment, the circular pattern 942 wear balances the solid-state storage media 110, increasing a usable life of the solid-state storage media 110. In the depicted embodiment, the groomer module 702 selects the region indicated by the tail pointer 964 as the next region to be recovered. The groomer module 702, in one embodiment, will recover the physical storage capacity of the region pointed to by the tail pointer 964 and add the recovered capacity to the available storage pool 946. In the depicted embodiment, modified versions of data in the regions 948, 950, 952, 954 have been appended to the writing structure 940 in new regions 956, 958, 960, 962 supplied from the available storage pool 946 to satisfy data modification operations. In further embodiments, the groomer module

702 may copy forward to the append point 944 certain dirty data (data in the cache that is subsequently modified) and selectively certain valid data that the regions 948 stores, if any.

Figure 7 depicts another embodiment of a direct cache module 1016. In the depicted embodiment, the direct cache module 1016 may include the block I/O emulation layer 506 and the direct interface layer 508 as described above with regard to Figures 2 and 4. The direct cache module 1016, in the depicted embodiment, further includes a groomer module 1002, a frequency count module 1004, a recovery event module 1006, a write request module 710, a cache write module 712, a direct mapping module 716, a read request module 720 comprising a read miss module 730 and read retrieve module 732, and a backing store interface module 722 which, in certain embodiments, may function substantially similar to the corresponding modules of Figure 4. In general, the direct cache module 1016 grooms data of the cache 102. Grooming data, as used herein, includes processing a region of data to recover physical storage capacity of the region by copying data of the region to another location in the cache 102, such as to an append point of a log, and/or evicting, invalidating, trimming, or otherwise clearing data from the region. Another term for grooming is garbage collection.

The groomer module 1002, in certain embodiments, recovers storage capacity of a region in response to a storage capacity recovery event, as described below with regard to the recovery event module 1006. In the depicted embodiment, the groomer module 1002 includes a frequent data module 1012, and an eviction module 1014. In other embodiments, the eviction module 1014 may function independently of the groomer module 1002. In one embodiment, the groomer module 1002 recovers storage capacity of a region by copying forward or otherwise retaining certain types of data from the region to optimize cache efficiency and by evicting or otherwise clearing other types of data from the region to optimize storage efficiency of the cache 102.

In one embodiment, the groomer module 1002 relocates or otherwise retains certain types of valid data (including certain metadata) in a region to preserve the valid data, to service storage requests, or the like. In one embodiment, the groomer module 1002 may selectively relocate or copy forward to the append point 944 data identified as frequent data and or certain metadata, or the like.

In one embodiment, the groomer module 1002 uses the frequent data module 1012 to relocate, or copy forward, data identified as frequently accessed data from a region, to retain the frequently accessed data in the cache 102. The frequent data module 1012, in various embodiments, may retain cached read data identified as frequently accessed, cached write data identified as frequently accessed, or both cached read data and write data identified as frequently

accessed. The frequent data module 1012, in a further embodiment, identifies frequently accessed data based on a frequency count, a touch count, or the like, such as a map, bit field, bit array, frequent data flags, and/or other frequent data indicators. In one embodiment, the frequent data module 1012 identifies frequently accessed data of a region to copy forward, or otherwise retain, using a frequency count that the grooming candidate set module 604 maintains for the region. The frequency count, in certain embodiments, is maintained by the frequency count module 1004.

In one embodiment, the frequent data module 1012 may handle frequently accessed data of a region differently based on a grooming mode of the groomer module 1002. In certain embodiments, the groomer module 1002 may operate in a plurality of modes, such as a low pressure groom mode, a high pressure groom mode, or the like. For example, the groomer module 1002 may transition from a low pressure groom mode to a high pressure groom mode in response to a lack of, or rapidly declining amount of, available storage capacity in the cache 102, a percentage of data marked as invalid reaching a predefined threshold level, performance of the cache 102 crossing a threshold value, in response to a storage capacity recovery event described below with regard to the recovery event module 1006, or the like.

The frequent data module 1012, in one embodiment, retains cached data identified as frequently accessed data when the groomer module 1002 is in a low pressure groom mode and the frequent data module 1012 allows eviction of cached data identified as frequently accessed data when the groomer module 1002 is in a high pressure groom mode. Of course, in certain embodiments, the amount of frequently accessed data copied forward may vary in proportion to the degree of grooming pressure, so discrete levels or steps of groomer pressure modes may not be the final determining factor in determining whether frequently accessed data is copied forward. The frequent data module 1012 may examine other factors in determining whether or not to copy data forward on the log 940. By processing frequently accessed data differently in a high pressure groom mode than in a low pressure groom mode, in certain embodiments, the groomer module 1002 optimizes cache efficiency by retaining frequently accessed data when there is low grooming pressure, while optimizing storage capacity recovery when there is high grooming pressure.

In one embodiment, the frequent data module 1012 preserves or retains data by copying the data forward to an append point of the log 940. In another embodiment, the frequent data module 1012 may preserve or retain data by copying the data to another cache in a hierarchy of caches. Caches in a hierarchy may comprise VSU cache units of a single cache device 102, multiple cache devices 102, or the like. Caches in a hierarchy may be arranged based on various

characteristics of the caches, such as characteristics of the nonvolatile storage media 110 (e.g. SLC media, MLC media, storage density, media age, error rates), characteristics of a link to the host device 114 (e.g. DAS over a system bus, NAS over a data network), capacity of the caches, VSU cache unit attributes, or on other cache characteristics. By copying certain data in a persistent manner, the direct cache module 1016 may continue to service storage requests for the data after other data from a region is evicted.

The eviction module 1014, in one embodiment, evicts, trims, erases, or otherwise clears data from a region to recover the storage capacity of the region. Erasing data from a region, without relocating the data, evicts the data from the cache 102. The eviction module 1014, directly or in cooperation with the direct mapping module 716, updates the mapping structure 900 to indicate that the cache 102 no longer stores evicted data. For example, in embodiments where membership in the mapping structure 900 denotes membership in the cache, the eviction module 1014 may remove one or more entries from the mapping structure 900 for ranges of LBAs that the eviction module 1014 has evicted. In embodiments where the mapping structure 900 maintains metadata for evicted data, to track cache misses or the like, the eviction module 1014 may mark one or more entries in the mapping structure 900 as evicted in response to evicting the associated data from the cache 102.

As described above with regard to the destage module 714 of Figure 4, in embodiments where the direct cache module 1016 manages the cache 102 according to a write-back policy, the destage module 714 either destages dirty write data from a region to the backing store 118 or copies dirty write data from a region forward on the log 940 prior to the eviction module 1014 evicting data from the region, so that the dirty write data may be preserved in the backing store 118. In certain embodiments, the destage module 714 may be integrated with the groomer module 1002 and/or the eviction module 1014. In other embodiments, the eviction module 1014 evicts exclusively from the tail of the log 940, but the groomer module 1002 may perform garbage collection operations for other regions of the log 940, the destage module 714 may destage data from other regions of the log 940, or the like.

In one embodiment, the groomer module 1002 and/or the eviction module 1014 clears or erases all data in a region of the physical storage media 110 in response to the frequent data module 1012 retaining and/or copying forward frequent data (including potentially metadata) from the region. The erase operation evicts data that is not retained or copied forward from the cache 102. In a further embodiment, the frequent data module 1012 may mark data to be retained as valid and/or the eviction module 1014 may mark data to be evicted as invalid, and a separate garbage collection process of the groomer module 1002 may retain the valid data and

erase or otherwise clear the invalid data. In one embodiment, the eviction module 1014 issues a TRIM or invalidation command to the storage controller 104 to invalidate, evict, or TRIM, all the data in a given region from the cache 102. The storage controller 104 may then schedule the region to be erased or otherwise prepare the region to enter the available storage pool 946.

In certain embodiments, the eviction module 1014 operates on single regions of the sequential log-based writing structure ("log") 940. Alternatively, or in addition, the eviction module 1014 may operate on a single region or a plurality of regions as a batch, either serially or in parallel. Selection of the region, for the eviction module 1014 to operate on, is determined at least in part by a region selection policy.

In one embodiment, the region selection policy implemented by the eviction module 1014 is to select the last or oldest region in the log 940 (e.g. the tail region) as the next region for storage recovery and once the last region is processed to move, in succession in the log 940, to the next to last region (which now has become the last or oldest region). The eviction module 1014 may continue operating according to the region selection policy until a sufficient number of regions are recovered. The eviction module 1014 may identify the last or oldest region in the log 940 as the tail of the log 940. As illustrated in Figure 6, the tail region may be identified by a tail pointer 964. Alternatively, the region referenced by the append point 944 may include a pointer to the tail region, or the like.

In certain embodiments, the eviction module 1014 follows a region selection policy which requires the eviction module 1014 to only select the tail region for eviction operations. Such a region selection policy may be part of an eviction policy referred to herein as a tail-end eviction policy or a FIFO eviction policy. Each successive region eviction operation also selects the current tail region for eviction. With each region eviction, the tail moves toward the head region, to the adjacent or neighboring region. The tail region may be identified or referenced as: the region indicated/pointed to by the tail pointer 964, the region having the least recently written data, the tail of the log 940, and the like.

As described above, writing data (both user data and metadata) in the cache 102, on the storage media 110 provides a persistent record of the order in which user write commands and cache misses (user reads of data not yet in the cache 102) caused data to be written to the cache 102. Certain embodiments of the groomer module 1002 and/or eviction module 1014 may cooperate with the storage controller 104 to implement a tail-end eviction policy that evicts data from the cache 102, and log structure 940 stored on the media 110 of the cache 102, in a first-in-first-out manner. The tail-end eviction policy may also be referred to as a least recently written eviction policy, first-in-first-out eviction policy, or the like.

In one embodiment, the persistent log 940 can be scanned and read from the tail 964 to the head to recover the state of the cache 102 (rebuild the mapping structure 900) following an unexpected shutdown (e.g. crash) of the cache 102, storage controller 104, or host 114. In certain embodiments, if an unexpected shutdown (e.g. crash) occurs the persistent log 940 is not scanned and used to rebuild the mapping structure 900. Instead, data in the log 940 may be erased (either in bulk or as more storage capacity is needed for the available storage pool 946) such that an empty cache 102 is presented.

In certain embodiments, the validity information for data in a region is maintained in the mapping structure 900 stored in volatile memory such that the validity information may be lost due to a crash or sudden power failure. In one embodiment, the storage controller 104 and/or direct cache module 116 may treat all data in the log 940 as valid, until a more recent version of the data is found on the log 940 (scanning from tail to head). User data in the log 940 can also be invalidated by a storage client 504 TRIM command. This TRIM command identifies data the storage client 504 does not need retained on the cache 102 and/or the backing store 118 (TRIMmed data).

In one embodiment, neither the storage controller 104 nor the direct cache module 116 persist that a TRIM command for a particular set of data has been received from the storage client 504. TRIMmed data exists on the log 940 because a TRIM command refers to data previously stored on either the cache 102 or the backing store 118. One reason the receipt of a TRIM command may not be persisted in the log 940 or in the cache 102 may be to maintain high performance and/or may be due to a requirement of the media 110 that prevents storing metadata regarding the TRIM command in the same region as the TRIMmed data (*e.g.* flash media prevents update-in-place operations, re-write only permitted following an erase operation).

Consequently, in embodiments in which the storage controller 104 or direct cache module 116 restores the cache 102 from the log 940 after an unexpected shutdown, the TRIMmed data is identified in a recreated version of the mapping structure 900. Such behavior may be undesirable because the TRIMmed data is no longer needed by a client 504 but is occupying physical storage capacity. The storage controller 104 and/or direct cache module 116 are not aware that the data is TRIMmed data due to the shutdown. Thus, the storage controller 104 and/or direct cache module 116 treat the TRIMmed data, in this example, as valid data. To avoid a waste of space, the direct cache module 116 may simply prepare and present an empty cache 102 following an unexpected shutdown (e.g. crash).

In one embodiment, the eviction module 1014 communicates with the frequent data module 1012 and/or the destage module 714 to prepare the tail region for data eviction. Such

preparations may include the frequent data module 1012 performing, or completing, any copy forward operations for valid data in the region based on the parameters determined by the frequent data module 1012, the destage module 714 destaging or copying forward dirty write data, and the like. The interaction and cooperation of the frequent data module 1012 and the eviction module 1014 manages the trade-offs between maintaining sufficient physical storage capacity in the cache 102 for storage management using the log 940 and maximizing cache efficiency by keeping the most active data in the cache 102 as long as possible.

For example, if grooming pressure is above a certain threshold no copy forward operations may be performed, or completed, such that the tail region can be recovered as quickly as possible. Alternatively, given a particular grooming pressure, the frequent data module 1012 may only copy forward a portion of the data having some other cache favorable characteristic, while other valid data not having the characteristic may be erased such that subsequent requests for the valid data will result in a cache miss. Examples of cache favorable characteristics may include a frequency count above a certain threshold, a type of cache operation that placed the data into the cache 102, a cache unit priority level for the data, a cache unit quality-of-service level for the data, a cache unit storage capacity associated with the data, and the like. Alternatively, given a particular grooming pressure, the frequent data module 1012 may only copy forward metadata that describes the layout, format, and/or organization of the data in the tail region, while valid data will be evicted and erased such that subsequent requests for the valid data will result in a cache miss.

In one embodiment, if the frequent data module 1012 and/or the destage module 714 indicate that the tail region is prepared for eviction, the eviction module 1014 performs the eviction. In one embodiment, the eviction module 1014 evicts data of the tail region by signaling the storage controller 104 to erase the tail region. The eviction module 1014 advances the tail pointer 964 toward the head of the log 940, to the neighboring region, to designate a new tail region. In addition, eviction module 1014 may adjust or advance other points in the log 940 to account for the erasure of the old tail region. In certain embodiments, the eviction module 1014 and/or the storage controller 104 will place the old tail region in the available storage pool 946. Placing the old erased tail region in the available storage pool 946 may comprise other pointer or value adjustments which then will allow the append point 944 to move into, and reference, the old tail region as the storage controller 104 receives more write requests.

The groomer module 1002, upon recovering a region of the physical storage media 110, places the region in the available storage pool 946 which allows the cache 102 to re-use the region of the physical storage media 110 to store different data. The groomer module 1002

and/or the eviction module 1014, in one embodiment, erase existing data in a region. In a further embodiment, the groomer module 1002 and/or the eviction module 1014 allow the cache 102 to overwrite existing data in a region. Whether or not the groomer module 1002, in various embodiments, erases existing data in a region may depend on the nature of the physical storage media 110. For example, Flash media may require that cells be erased prior to reuse where magnetic media such as hard drives may not have that requirement. In an embodiment, where the groomer module 1002 does not erase data in a region, but allows the cache 102 to overwrite data in the region, the groomer module 1002, in certain embodiments, may mark the data in the region as unavailable to service read requests so that subsequent requests for data in the region return a null result or an empty set of data until the cache 102 overwrites the data.

In one embodiment, the groomer module 1002 selects regions starting with the tail region and successively progresses towards the head region or append point and performs operations to prepare the regions for eviction, without erasing the regions and recovering the storage capacity. The preparation operations may include the copy forward operations provided by the frequent data module 1012 for the selected region(s), destaging operations provided by the destage module 714 for the selected region(s), or the like. In this manner, one or more regions near, and including the tail region, may be primed or prepared for eviction such that once the eviction module 1014 processes the tail region, or region(s), the regions can be immediately erased/trimmed without a delay caused by operations of the frequent data module 1012 and/or the destage module 714. In addition, primed regions are available to continue to servicing read requests which maximizes cache efficiency.

In certain embodiments, a user or storage client 504 may adjust whether the groomer module 1002 does pre-emptive grooming, and if so, the magnitude and/or duration of pre-emptive grooming. In certain embodiments, where the workload is steady and the data working set is larger than the physical capacity of the cache, pre-emptive grooming may be advantageous in expediting storage space recovery. However, if the workload is light and the working set is smaller than the physical capacity of the cache, pre-emptive grooming may not be desired since the copy forward operations may cause unnecessary wear on the media 110, write amplification, and processing overhead.

In one embodiment, the groomer module 1002 selectively chooses a region and performs certain operations to prepare the chosen region for eviction, without erasing the chosen region and recovering the storage capacity. In selecting the region to prepare for eviction, the groomer module 1002 may select any region between, and including, a tail region and a head region (referenced by the append point 944). The groomer module 1002 may choose a region based on

a degraded condition of the media 110 for the region. Degradation of the media 110 may be evidenced by, among other factors: a failure condition, partial failure, inaccessibility, unacceptable error rate, unacceptable performance (*e.g.,* long read, program, and/or erase times), programming errors, wear level, or the like.

While the region may exhibit characteristics indicative of reduced performance or reliability, the region may still be useable for reading of valid data presently stored in the region. This may be particularly the case where the storage controller 104 and/or direct cache module 116 include an error recovery module that can tolerate and recover a certain number of read errors. By allowing the groomer module 1002 to select any region that exhibits signs of degraded operation, the groomer module 1002 preemptively avoids data loss and uncorrectable data errors that may occur if the region is re-used. In addition, potential read errors for the region can be avoided.

If the groomer module 1002 selects a region exhibiting signs of degraded operation, the groomer module 1002 may determine if any valid data can be reliably copied forward or whether the region should be retired immediately. The frequent data module 1012 copies valid data forward on the log 940 that can be reliably copied. The groomer module 1002 then retires the region and adjusts the log 940 such that the region will no longer be used. In certain embodiments, the groomer module 1002 swaps in physical capacity from one or more physical pages, erase blocks, or the like to replace the region. In such an instance, the groomer module 1002 may obtain replacement physical capacity from one or more regions of the available storage pool 946.

In certain embodiments, the groomer module 1002 preemptively selects regions for grooming moving from the tail of the log 940 towards the head 944, but no data is evicted. Alternatively, or in addition, the groomer module 1002 preemptively selects any region for grooming moving between the tail of the log 940 and the head 944 and prepares the region without evicting the data. Such a selection may be based on a determination by the direct cache module 116 that the selected region may include data associated with a one-time (very infrequent) I/O operation such as a one-time read or a one-time write of a very large set of data. The direct cache module 116 may target a given region by analyzing the workflow, the average size of ranges for I/O operations, and the like. In this manner, the direct cache module 116 may mitigate the impact of a large I/O operation that "poisons" or contaminates the cache 102. By preemptively copying any valid data forward, the cache module 116 prepares the selected regions for rapid recovery once the tail of the log 940 moves to these regions.

In certain embodiments, the groomer module 1002 operates on a plurality of regions simultaneously. In certain embodiments, the regions processed by the groomer module 1002 may be selected in order from the tail of the log 940 to the head according to the region selection policy defined above (select the tail 948 first, and then each region in the log 940 in succession, towards the head). Operating on a plurality of regions may permit the direct cache module 1016 to balance the overhead associated with operating the groomer module 1002 against the grooming pressure (need to recover more regions such that the storage controller 104 has available capacity). In addition, the number of regions evicted by the eviction module 1014 impacts the cache efficiency. Consequently, the direct cache module 1016 factors in the cache efficiency in determining the number of regions for the groomer module 1002 to process and evict. In certain embodiments, the eviction module 1014 guarantees that the eviction module 1014 will evict regions in tail-end or FIFO order.

The number of regions processed may be influenced by the grooming pressure, the duty cycle for the groomer, the workload allocated to the groomer module 1002, or the like. For example, in one embodiment, the groomer module 1002 may operate on a workload quota basis in which the frequent data module 1012 and eviction module 1014 process each region in turn from the tail region 948 toward the head (pointed to by append point 944) until a certain amount of valid frequent data has been copied forward on the log 940, a certain number of regions have been processed, or the like.

As described below, the recovery event module 1006 may trigger grooming and/or eviction in response to a storage capacity recovery event. In a further embodiment, the groomer module 1002 may perform a garbage collection operation to clear data from a tail region of the log 940. If the tail region stores exclusively invalid data, grooming the tail region provides additional storage capacity for the cache 102 without evicting data from the cache 102. Upon clearing the invalid data from the current tail region of the log 940, the groomer module 1002 may designate a neighboring region to the present tail region of the log 940 as the tail region.

Similarly, or in addition, the frequent data module 1012 and/or eviction module 1014 may operate on one or more regions simultaneously. For example, operation of the frequent data module 1012 may be optimal, if a certain quantity of valid data is copied forward on the log 940. Therefore, if the tail region 948 does not have enough valid data to be copied forward, the frequent data module 1012 may advance to the region prior to the tail, and repeatedly move to the next region, until the required quantity of valid data is copied forward. Similarly, the frequent data module 1012 may get as much frequent data copied forward during an assigned time slice as possible, even preemptively processing regions.

Similarly, the eviction module 1014 may operate on more than one region in batch even if the frequent data module 1012 has not processed a region. As mentioned above, the grooming pressure may cause the eviction module 1014 to evict more than one region at a time. In certain embodiments, the eviction module 1014 is restricted to evicting data and recovering regions in the order that the regions exist in the log 940 moving from the tail, region 948 (pointed to by 964) toward the head (pointed to by 944) (described above as a tail-end eviction policy).

As described above, writing data in the cache 102, on the storage media 110 provides a persistent record of the order in which user write commands and cache misses caused data to be written to the cache 102. This persistent record can be replayed/reviewed/scanned to recover the state of the cache 102 following an unexpected shutdown of the cache 102 or host. In embodiments in which the direct cache module 116 evicts data from the cache 102 using the tail-end eviction policy and writes data to the cache using an append-only, log-based writing structure 940, the state of the cache 104 can be maintained and restored following an unexpected shutdown.

Advantageously, the issuing of TRIM commands by the direct cache module 116 to the storage controller 104 facilitates operation of the groomer module 1002 because each region of the log-based writing structure 940 contains minimal valid data. The valid data may, or may not, be copied forward on the log 940 to maximize cache efficiency.

The number of regions, the eviction module 1014 evicts with each eviction operation is not limited, but does have a potential cost in cache efficiency. Given a sufficiently high grooming pressure, the eviction module 1014 may evict a set of successive regions from the tail end of the log 940 up to the size of the log 940. In certain embodiments, the eviction module 1014 may evict no more than half the size of the log 940 in order to mitigate cache thrashing conditions. Such multiple region evictions may, in certain embodiments, include permitting the frequent data module 1012 to copy valid data forward on the log 940. In this manner, a very high quantity of physical storage capacity may be recovered very rapidly. Such rapid storage space recovery may be in response to a rapid increase in cache misses and/or user writes that cause cache writes when the cache 102 is in write-through mode.

In one embodiment, the frequency count module 1004 provides a data use metric, such as a frequency count or touch count for one or more given regions identified by the groomer module 1002, and/or for data of a region, such as a page, ECC codeword, packet and/or sector. A data use metric, as used herein, comprises a rate at which data of the cache 102 is accessed. A data use metric may indicate data access at a region level, such as a logical erase block level, at a packet level, at a sector level, or the like. In another embodiment, a data use metric may include

a VSU cache unit attribute as described above with regard to Figure 1A. In the depicted embodiment of Figure 7, with the direct cache module 1016 operating in write-through mode, or write around mode, the given one or more regions comprise one or more regions starting from the tail region (region 948 in the log 940 for Figure 6) moving toward the head region. The head region is the region associated with the append point 944 and the tail region the region associated with the tail pointer 964. Those of skill in the art, in light of this disclosure, will recognize a variety of ways to identify the tail of the log structure 940 other than using a tail pointer 964, all such variations are within the scope of the present invention.

As described above, in various embodiments, a frequency count or other data use metric may be relative to the life of cached data in the cache 102, relative to a predefined time period such as one or more periods of the circular grooming data structure, or the like. In one embodiment, the frequency count module 1004 identifies valid, clean data as satisfying a data use metric (e.g. as frequently accessed) and includes the data in a frequency count in response to a predefined number of accesses of the data in a predefined time period. For example, in one embodiment, the frequency count module 1004 identifies data as recent data in response to a first access or read of the data and identifies the data as frequent data, satisfying a data use metric, in response to one or more subsequent accesses or reads of the data, or the like.

In one embodiment, the frequency count module 1004 determines frequency of access for regions in the log structure 940 and/or for packets, sectors, or other data of a region using a history of access to the mapping structure 900. In certain embodiments, the frequency count module 1004 cooperates with the direct mapping module 716 to determine frequency of access for regions or other data in the log structure 940. The frequency count module 1004, in a further embodiment, identifies areas of high frequency, "hot," use and/or low frequency, "cold," use by monitoring accesses of branches or nodes in the mapping structure 900. The frequency count module 1004, in a further embodiment, determines a frequency count of access to a branch, directed edge, or node in the mapping structure. In one embodiment, the count associated with each node of a b-tree like mapping structure may be incremented for each I/O read operation and/or each I/O write operation that visits the node in a traversal of the mapping structure 900. Of course, in certain embodiments, separate read counts and write counts may be maintained for each node. Certain counts may be aggregated to different levels in the mapping structure in other embodiments.

The storage controller 104, in one embodiment, provides a frequency count or other data use metric, for a particular region to the frequency count module 1004 at the request of the frequency count module 1004. In another embodiment, the frequency count module 1004 counts

or tallies blocks/sectors/packets of data identified as frequently accessed data for a selected region. For example, in one embodiment, the frequency count module 1004 maintains a frequency count for one or more regions at or near a tail of the log structure 940. The frequency count module 1004 may count or tally frequently accessed data by traversing a map, a bit field, a bit array, frequent data flags, or the like, for a particular region and incrementing a counter for each block/sector/packet of data identified as frequently accessed, or the like. In one embodiment, the storage controller 104 provides a map, a bit field, a bit array, frequent data flags, or the like for a particular region to the frequency count module 1004. In another embodiment, the frequency count module 1004 maintains a map, a bit field, a bit array, frequent data flags, or the like indicating which data in a selected region is frequently accessed data.

In one embodiment, the frequency count module 1004 clears a frequency count or other data use metric for a block/sector/packet and/or region after a predefined period of time. By clearing the frequency counts for blocks/sectors/packets and/or regions of the cache 102, data that was used frequently may transition to being used less frequently and thus become more likely to be evicted once that data is in the region at the tail (indicated by pointer 964) of the log 940. In other embodiments, the frequency count module 1004 may clear frequency counts every N periods in the predefined time period, for example every other period, every third period, every fourth period, etc.

In one embodiment, the recovery event module 1006 detects and/or initiates a storage capacity recovery event for the cache 102. In one embodiment, the recovery event module 1006 may initiate a storage capacity recovery event periodically, when a predefined period of time has elapsed since a previous storage capacity recovery event or the like. In certain embodiments, the recovery event module 1006 initiates a storage capacity recovery event by signaling the groomer module 1002 and/or eviction module 1014 to recover one more regions from log-based writing structure 940. A storage capacity recovery event is an event that triggers recovery of storage capacity of the cache 102. The recovery event module 1006, in certain embodiments, monitors the storage controller 104 and/or the cache 102 for the occurrence of a storage capacity recovery event. In another embodiment, the recovery event module 1006 may receive a storage capacity recovery event notification, such as a storage capacity recovery request, an eviction request, or the like, from another module, from the storage controller 104, from the host device 114, from a cache client, or the like.

In one embodiment, a storage capacity recovery event includes a grooming pressure for the cache exceeding a predefined grooming pressure threshold. In another embodiment, a storage capacity recovery event may include an available storage capacity of the cache 102

falling below a predefined available capacity threshold. For example, suppose the available storage pool 946 comprises 200 regions and the predefined available capacity threshold is 150 regions. In one embodiment, the predefined available capacity threshold is equal to or greater than one region, so that at least one region is available for the storage controller 104 to insert data into the log 940. The predefined available capacity threshold, in a further embodiment, is large enough so that the storage controller 104 may write any in-flight data to the nonvolatile storage media 110 should a power failure or other unexpected shutdown occur. In-flight data includes data of a write request from the host device 114 or another client that is not yet persisted in the non-volatile storage media 110 of the cache 102 or in the backing store 118. For example, the storage controller 104 and/or the cache 102 may store in-flight data in one or more buffers of the write data pipeline 106, or the like before the data is persisted to the non-volatile storage media 110.

As the storage controller 104 fills a region, the append point 944 moves to a next region supplied by the available storage pool 946. In one embodiment, no grooming operations may be performed until the number of regions in the available storage pool 946 falls to, or below, the predefined available capacity threshold, such as 150 regions or the like. In one embodiment, once the append point 944 progresses to the first region of the 150 regions (or other predefined available capacity threshold), a storage capacity recovery event may be triggered or initiated. In certain embodiments, the predefined available capacity threshold is set by a user, by the storage client 504, by the storage controller 104, and/or may be dynamically adjusted based on a measurement of the workload.

A storage capacity recovery event, in a further embodiment, may include the storage controller 104 receiving a write request for a set of data having a size that will adversely impact the available physical storage capacity of the cache 102 once satisfied. The storage controller 104 and cache 102 are configured to accept write requests for a range of logical addresses in a single write request. In addition, when the storage controller 104 serves as a cache 102 for a backing device 118, a read request from the storage client 504 may result in a write within the storage controller 104 to initially load data into the cache 102 (a user read miss). Depending on the relative sizes of the log 940, the available storage pool 946 size, and the data associated with a received write request (or read causing a cache miss), the data needed to satisfy the request may trigger a storage capacity recovery event.

For example, suppose the storage block/sector size is 512 bytes and storage client 504 requests a storage operation for a range of 6000 blocks/sectors within a range of logical blocks/sectors. Satisfying such a request will require about 3,072,000 bytes. If the cache 102 or

storage controller 104 determines that the available storage capacity in a region pointed to by the append point 944 and the number of regions in the available storage pool 946 are insufficient to satisfy the request, then a storage capacity recovery event may be triggered. Alternatively, if the size of data for a pending storage request exceeds a particular threshold, one or more of the cache 102 and storage controller 104 may trigger a storage capacity recovery event. In various other embodiments, a storage capacity recovery event may be triggered by an error detection rate reaching a threshold value, performance crossing a threshold value, a scheduled garbage collection or grooming cycle, a quantity of valid data in the cache 102 or certain regions of the cache 102 reaching a certain threshold, or the like.

A storage capacity recovery event, in another embodiment, may include the non-volatile storage media 110 satisfying an error condition. In one embodiment, the non-volatile storage media 110 satisfies an error condition in response to one or more regions having an error rate, such as an Uncorrectable Bit Error Rate ("UBER"), a Correctable Bit Error Rates ("BER"), or the like that exceeds an error threshold. In another embodiment, the non-volatile storage media 110 may satisfy an error condition in response to a Program / Erase ("PE") cycle count, a read frequency, and/or other non-volatile solid state storage specific attribute satisfying an error threshold. For example, the groomer module 1002 may retire one or more regions as described above, triggering a storage capacity recovery event for the recovery event module 1006 so that the groomer module 1002 and/or the eviction module 1014 recover storage capacity of one or more regions to compensate for the retired regions.

In one embodiment, the write request module 710 uses a write-through policy, performing a synchronous write to the backing store 118 for each write request. In one embodiment, the write request module 710 does not send acknowledgement of the write request to a storage client 504 until acknowledgement is received from both the storage controller 104 and the backing store 118. In another embodiment, the write request module 710 does send acknowledgement of the write request to a storage client 504 once the storage controller 104 acknowledges the write request, and signals an error if the backing store 118 fails to acknowledge the write request. As described above, in certain embodiments the write request module 710 determines that the write request includes blocks/sectors/packets that overlap with other pending write requests, read requests, and TRIM (invalidate) requests, such overlap conditions may result in a write request not changing the cache 102 or the backing store as described above.

Advantageously, in certain embodiments, write amplification is reduced because the cache 102 operates under a write-through or write around caching policy and the frequent data

module 1012 limits data copied forward from the tail region(s) (identified by the tail pointer 964) to frequent data. As described above, in one embodiment, the groomer module 1002 copies certain valid data (which may include certain metadata) in a region forward to the current append point of the log-based append-only writing structure 940 of the cache 102 before recovering the physical storage capacity of the region.

Operating in write-through mode or write around mode means there is no data in the cache 102 that is not already stored on the backing store 118, therefore there is no need to manage dirty data in the cache 102. Consequently, the direct cache module 116 focuses on maximizing cache efficiency and maintaining sufficient physical capacity in the available storage pool 946. If the pressure for storage space in the cache is sufficiently high, the groomer module 1002 may evict valid user write data from the tail region 964 since the data is preserved on the backing store 118. For example, in one embodiment, the eviction module 1014 may mark certain data as conditionally evictable, or the like, and the groomer module 1002 may evict the conditionally evictable data in response to increased grooming pressure.

Figure 8 depicts one embodiment of a method 1100 for managing eviction of data. The method 1100 begins, and the cache write module 712 stores 1102 data on non-volatile storage media 110 of a non-volatile storage device 102 sequentially using a log-based storage structure 122 having a head region 128 and a tail region 124. The direct cache module 1016 caches 1104 data on the non-volatile storage media 110 using the log-based storage structure 122. The data, in one embodiment, is associated with storage operations between a host device 114 and a backing store storage device 118. The eviction module 1014 evicts 1106 data of at least one region 124 in succession from the log-based storage structure 940 starting with the tail region 124 and progressing toward the head region 128, and the method 1100 ends.

Figure 9 depicts another method 1200 for managing eviction of data. The method 1200 begins, and the groomer module 1002 determines 1202 whether a tail region 124 of a sequential log-based storage structure 122 comprises exclusively invalid data. If the groomer module 1002 determines 1202 that the tail region 124 comprises exclusively invalid data, the groomer module 1002 clears 1204 data from the tail region 124 during a garbage collection operation and designates a neighboring region to the present tail region 124 as the tail region 124, otherwise the method 1200 continues.

The recovery event module 1006 determines 1206 whether a storage capacity recovery event has occurred for the non-volatile storage device 102, such as a number of regions in an available storage pool 946 satisfying a threshold, an amount of available storage capacity for the non-volatile storage device 102 falling below a threshold, an eviction request from a cache client,

an error condition for the non-volatile storage media 110, a predefined period of time elapsing since a previous storage capacity recovery event, or the like. If the recovery event module 1006 determines 1206 that a storage capacity recovery event has not occurred, the method 1200 returns as the groomer module 1002 continues to determine 1202 whether the tail region 124 of the sequential log-based storage structure 940 comprises exclusively invalid data, and the like.

If the recovery event module 1006 determines 1206 that a storage capacity recovery event has occurred, the destage module 714 either destages 1208 dirty data to the backing store 118 or copies forward 1208 the dirty data from the tail region 124 to an append point 128 of the sequential log-based storage structure 940. The frequent data module 1012 determines 1210 whether any of the valid, clean data of the tail region 124 satisfies a data use metric, such as a frequency count threshold, a touch count threshold, or the like. If the frequent data module 1012 determines 1210 that any valid, clean data of the tail region 124 satisfies the data use metric, the frequent data module 1012 copies 1212 the valid, clean data to retain the valid, clean data in the cache 102 or in another cache 102 of a cache hierarchy, otherwise the method 1200 continues. For example, the frequent data module 1012 may copy the valid, clean data forward on the sequential log-based storage structure 122 to the append point 128, copy the valid, clean data to another cache 102 in a hierarchy of multiple cache devices 120, or the like to persistently cache the valid, clean data instead of evicting it.

The eviction module 1014 evicts 1214 remaining data from the tail region 124, erasing the remaining data and updating a mapping structure 900 for the cache 102 to indicate that the cache 102 no longer stores the data. The groomer module 1002 designates 1216 a neighboring region to the present tail region 124 of the sequential log-based storage structure 122 as the tail region 124 and the method 1200 returns as the groomer module 1002 continues to determine 1202 whether the tail region 124 of the sequential log-based storage structure 940 comprises exclusively invalid data, and the like.

The present invention may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiments are to be considered in all respects only as illustrative and not restrictive. The scope of the invention is, therefore, indicated by the appended claims rather than by the foregoing description. All changes which come within the meaning and range of equivalency of the claims are to be embraced within their scope.

CLAIMS

1. A method for managing eviction of data, the method comprising:

storing data on non-volatile storage media sequentially using a log-based storage structure having a head region and a tail region;

caching data on the non-volatile storage media using the log-based storage structure, the data associated with storage operations between a host and a backing store storage device; and

evicting data of at least one region in succession from the log-based storage structure starting with the tail region and progressing toward the head region.

2. The method of Claim 1, further comprising evicting regions from the log-based storage structure in a first-in-first-out order comprising an order that regions of the log-based storage structure are added to the log-based storage structure.

3. The method of Claim 1, further comprising, designating a neighboring region to a present tail region of the log-based storage structure as tail region of the log-based storage structure in response to successfully evicting data from the present tail region.

4. The method of Claim 1, further comprising evicting the data of the at least one region in response to a storage capacity recovery event.

5. The method of Claim 4, wherein the storage capacity recovery event comprises a number of regions in an available storage pool satisfying a threshold.

6. The method of Claim 4, wherein the storage capacity recovery event comprises one or more of an amount of available storage capacity for the non-volatile storage media falling below a threshold, a predefined period of time elapsing since a previous storage capacity recovery event, an eviction request from a cache client, and the non-volatile storage media satisfying an error condition.

7. The method of Claim 1, further comprising clearing data from a present tail region of the log-based storage structure during a garbage collection operation and designating a neighboring region to the present tail region of the log-based storage structure as tail region of the log-based storage structure in response to the present tail region comprising exclusively invalid data.

8. The method of Claim 1, further comprising selectively copying valid, clean data out of the at least one region to retain the valid, clean data in response to the valid, clean data satisfying a data use metric.

9. The method of Claim 1, further comprising destaging dirty write data from the at least one storage region to the backing store prior to evicting data of the at least one region.

10. The method of Claim 1, further comprising copying dirty write data from the at least one region forward on the log-based storage structure in response to evicting data of the at least one region.

11. The method of Claim 1, further comprising evicting a plurality of regions in order, the order defined by regions in the log-based storage structure moving from the tail of the log towards the head of the log.

12. The method of Claim 1, wherein the non-volatile storage media caches data according to a write through cache policy.

13. An apparatus for managing eviction of data, the apparatus comprising:

a cache write module that stores data on a non-volatile storage device sequentially using a log-based storage structure having a head region and a tail region;

a direct cache module that caches data on the non-volatile storage device using the log-based storage structure, the data associated with storage operations between a host and a backing store storage device; and

an eviction module that evicts data of at least one region in succession from the log-based storage structure starting with the tail region and progressing toward the head region.

14. The apparatus of Claim 13, further comprising a frequent data module that selectively copies valid, clean data from the at least one region to an append point in the log-based storage structure in response to the valid, clean data satisfying a data use metric.

15. The apparatus of Claim 13, further comprising a destage module that one of, destages dirty write data from the at least one storage region to the backing store prior to evicting data of the at least one region; and copies dirty write data from the at least one region forward on the log-based storage structure in response to evicting data of the at least one region.

16. The apparatus of Claim 13, wherein the at least one region comprises the tail of the log-based storage structure, the tail comprising an oldest written region of the log-based storage structure such that the eviction module evicts data in a first-in-first-out order.

17. The apparatus of Claim 13, wherein the eviction module evicts a plurality of regions in order, the order defined by regions in the log-based storage structure moving from the tail of the log towards the head of the log in a first-in-first-out order.

18. A system for managing eviction of data, the system comprising:

a processor;

a storage controller for a non-volatile storage device, the non-volatile storage

device in communication with the processor over one or more communications buses, the storage controller storing data on the non-volatile storage device sequentially using a log-based storage structure having a head region and a tail region;

a cache controller in communication with the storage controller, the cache controller caching data on the non-volatile storage device through the storage controller using the log-based storage structure, the data associated with storage operations between the processor and a backing store storage device; and

an eviction module of the cache controller, the eviction module evicting, through the storage controller, data of at least one region in succession from the log-based storage structure starting with the tail region and progressing toward the head region.

19. The system of Claim 18, further comprising a host computer system, the host computer system comprising the processor, wherein the storage controller and the cache controller each comprise a device driver executing on the processor of the host computer system.

20. The system of Claim 18, wherein the eviction module evicts regions from the log-based storage structure in a first-in-first-out order comprising an order that regions of the log-based storage structure are added to the log-based storage structure.

PAGE 1/13

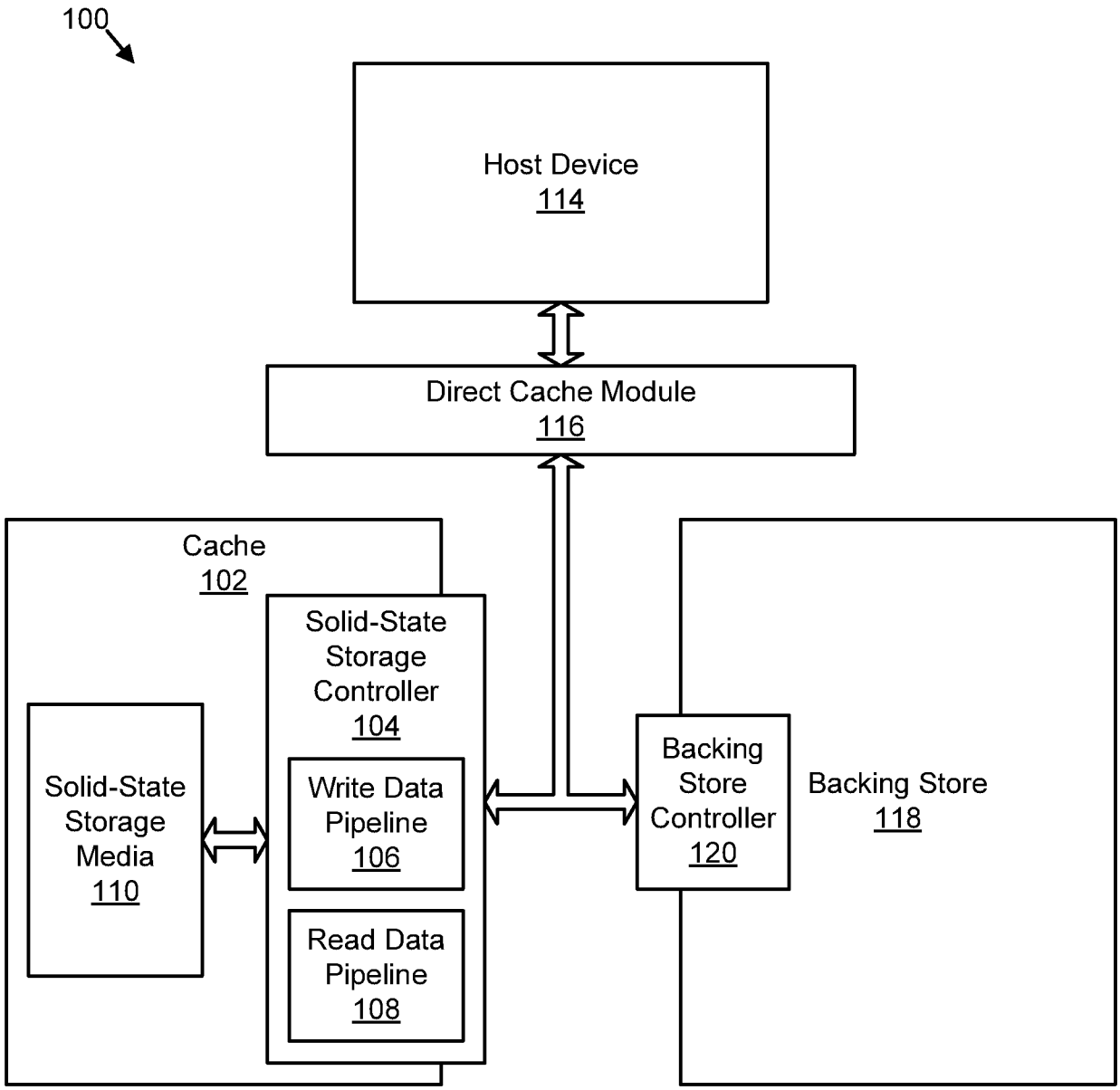


FIG. 1A

PAGE 2/13

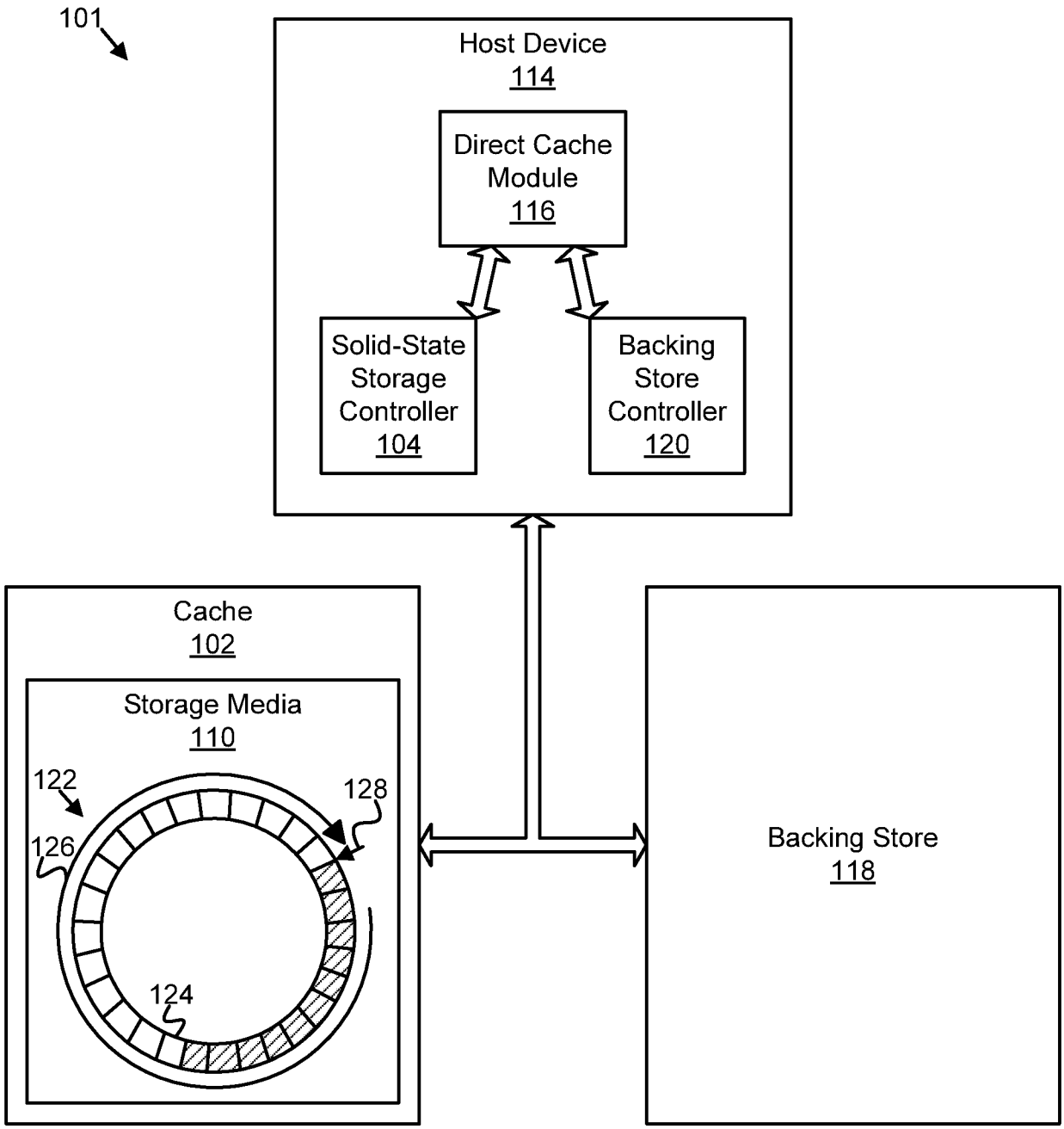


FIG. 1B

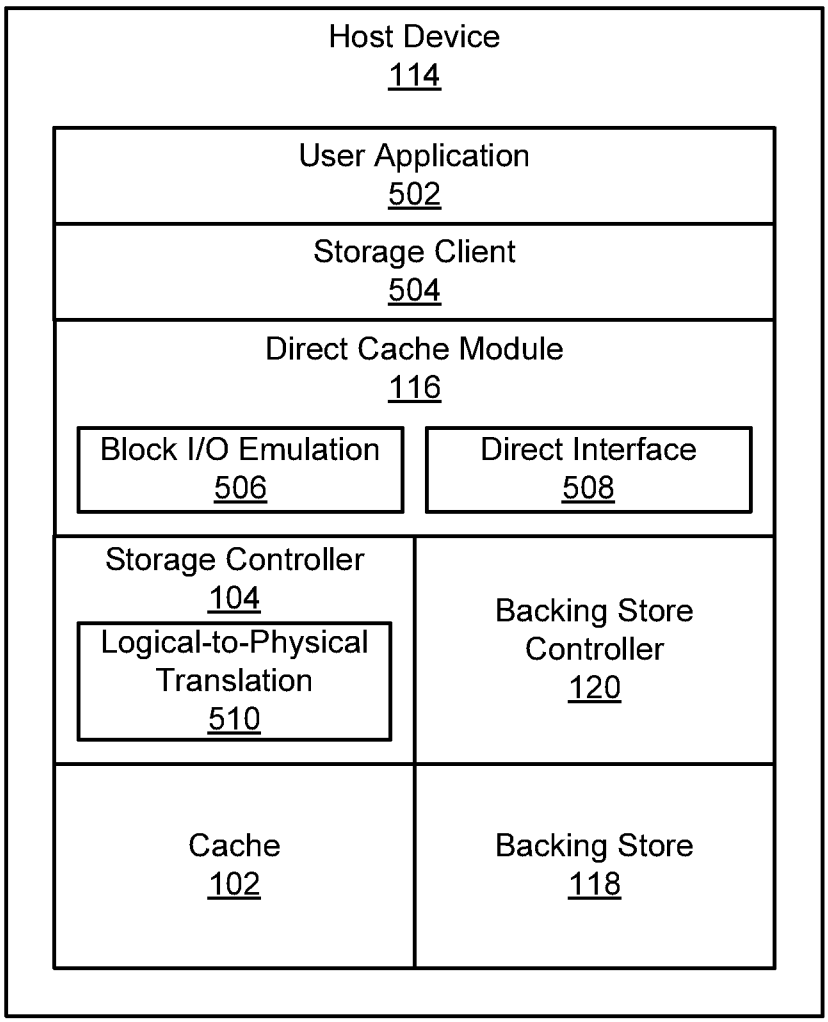

Host Device
114
User Application
502
Storage Client
504
Direct Cache Module
116
Block I/O Emulation
506
Direct Interface
508
Storage Controller
104
Logical-to-Physical Translation
510
Backing Store Controller
120
Cache
102
Backing Store
118

FIG. 2

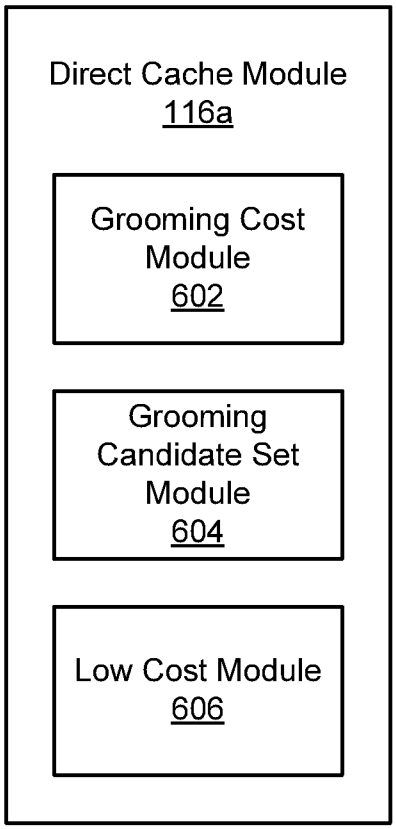

Direct Cache Module
116a
Grooming Cost Module
602
Grooming Candidate Set Module
604
Low Cost Module
606

FIG. 3

PAGE 5/13

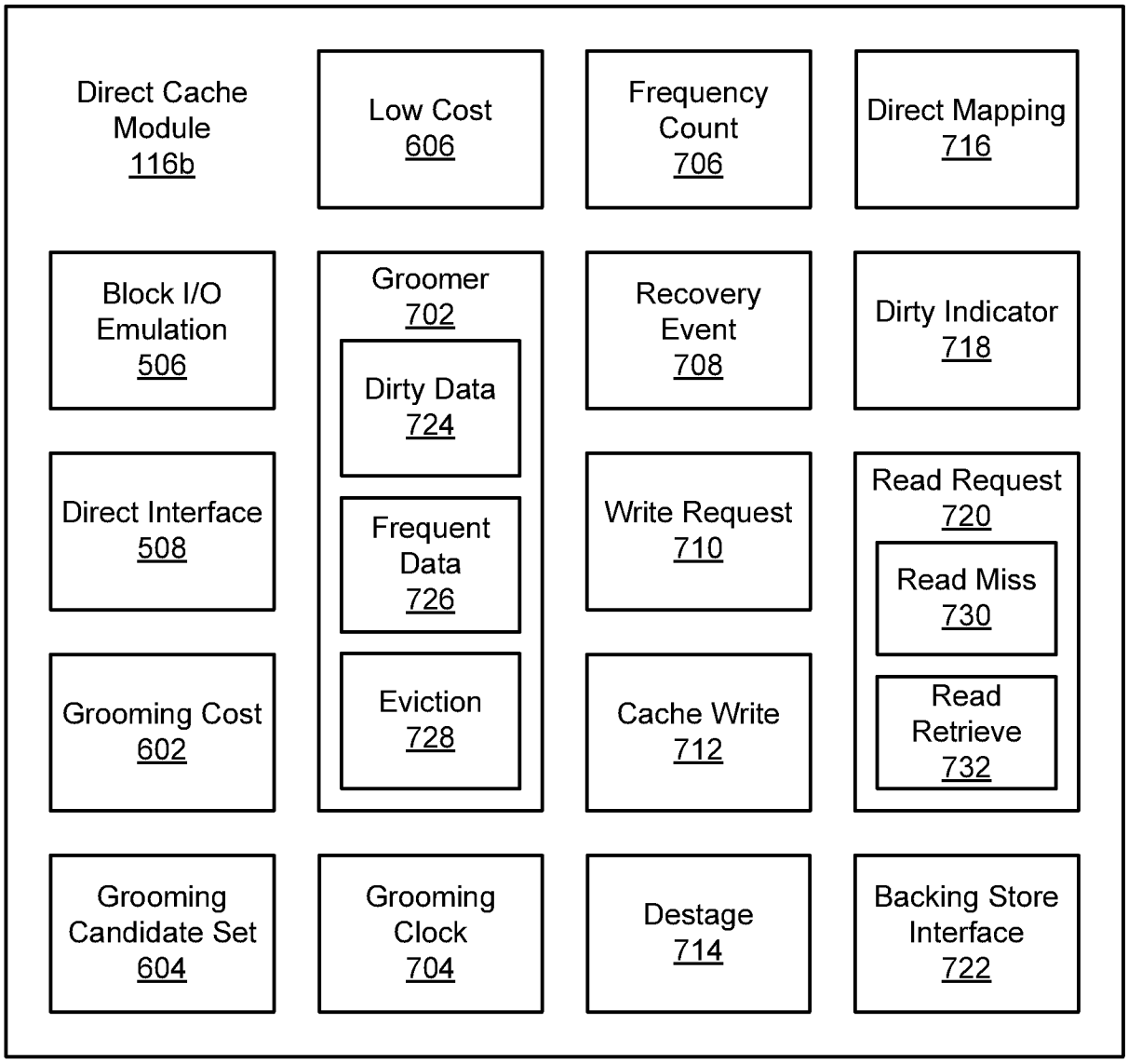
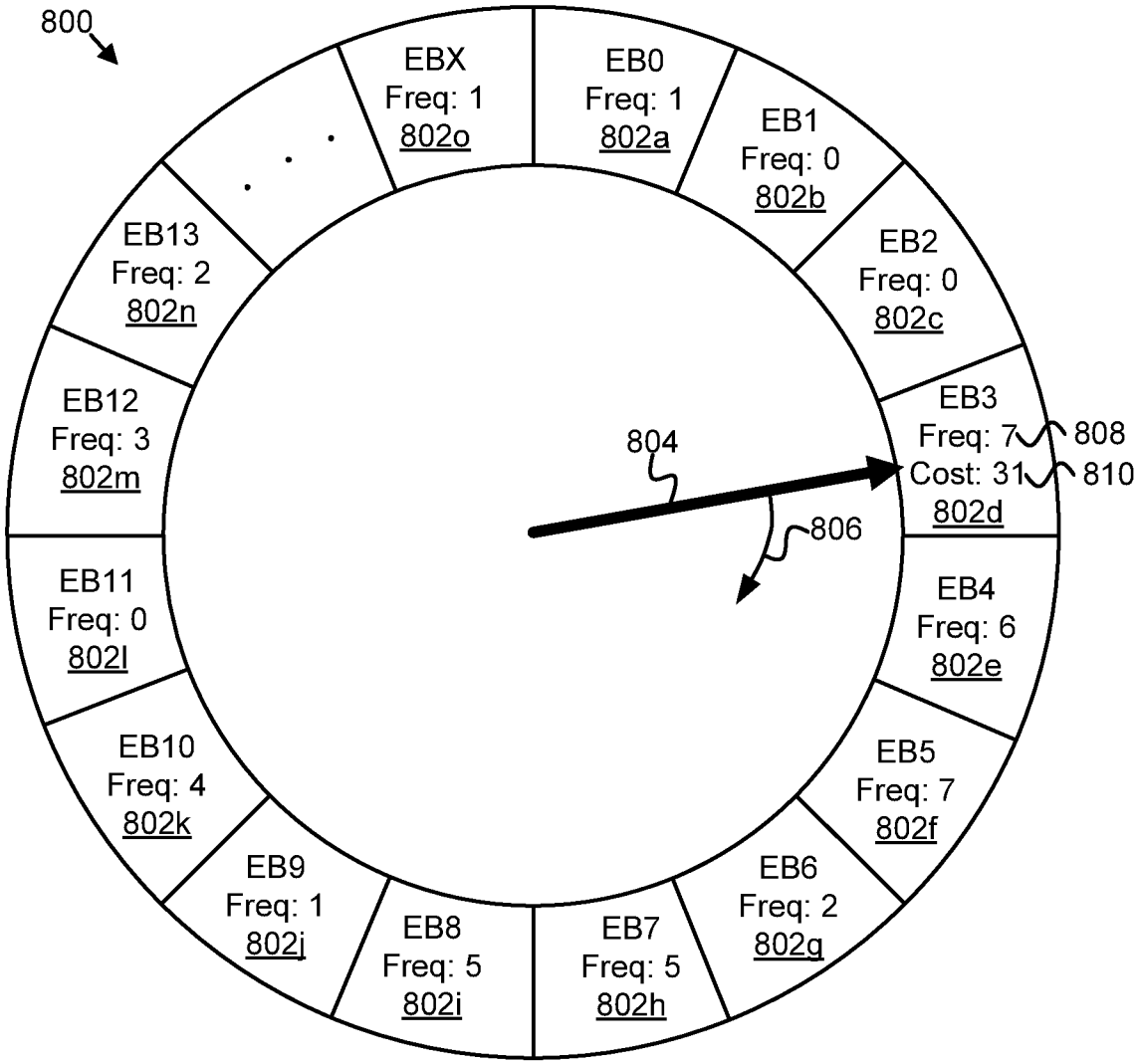


FIG. 4

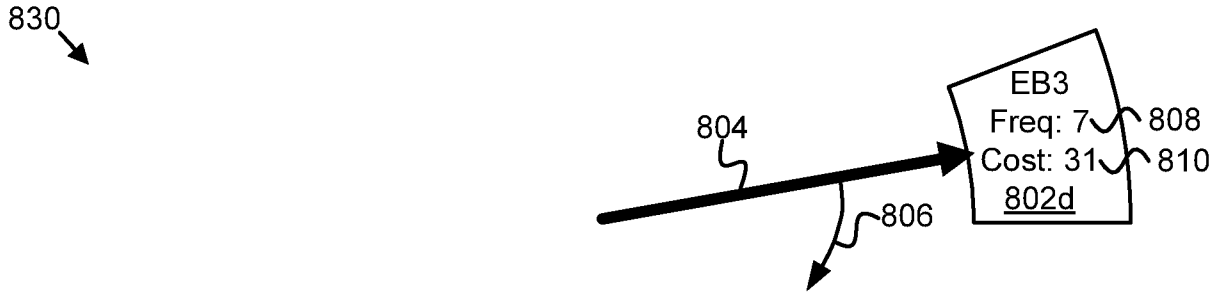
PAGE 6/13



Grooming Candidate Set
820a

810 / 808	EB: 6 Cost: 10 Freq: 4 822a	EB: 2 Cost: 15 Freq: 8 822b	EB: 10 Cost: 17 Freq: 8 822c	EB: 9 Cost: 23 Freq: 4 822d	EB: 13 Cost: 30 Freq: 8 822e	EB: 7 Cost: 33 Freq: 9 822f	EB: 12 Cost: 41 Freq: 8 822g

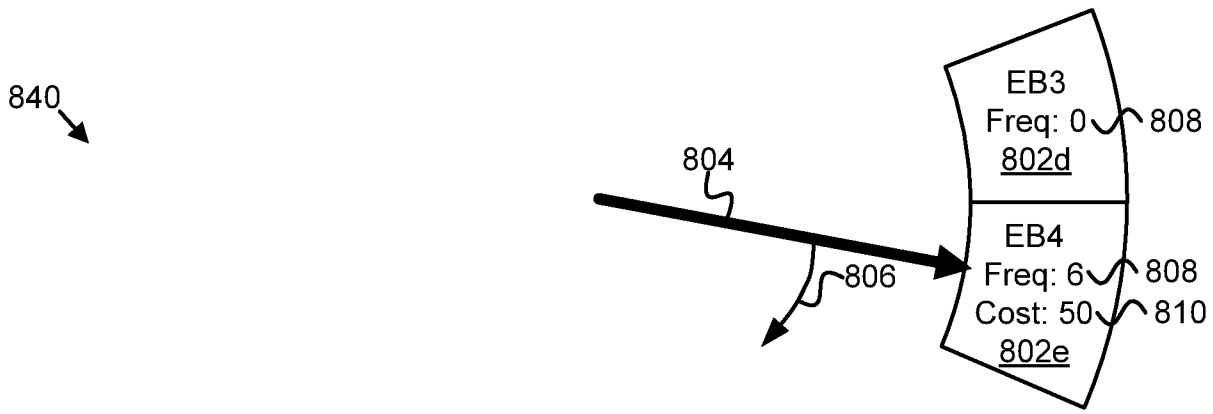
FIG. 5A



Grooming Candidate Set
820b

EB: 2 Cost: 15 Freq: 8 822a	EB: 10 Cost: 17 Freq: 8 822b	EB: 9 Cost: 23 Freq: 4 822c	EB: 13 Cost: 30 Freq: 8 822d	EB: 3 Cost: 31 Freq: 7 822e	EB: 7 Cost: 33 Freq: 9 822f	EB: 12 Cost: 41 Freq: 8 822g

FIG. 5B

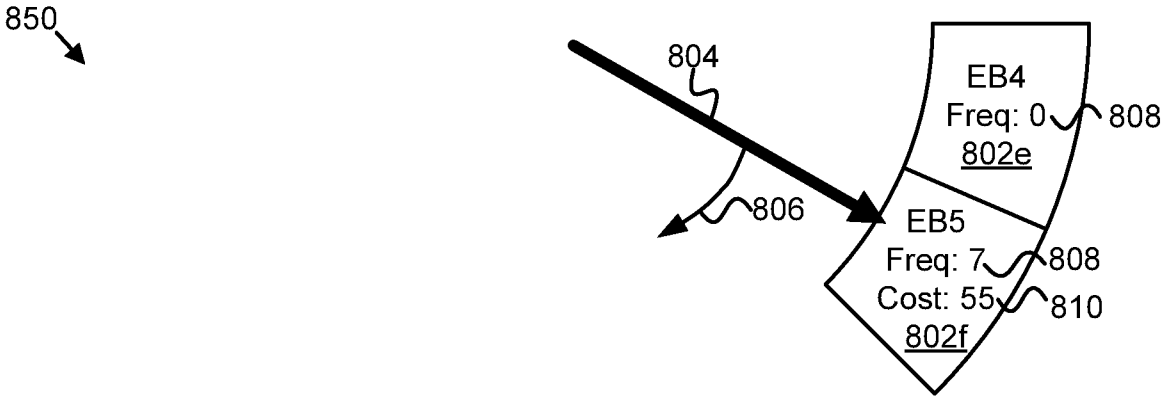


Grooming Candidate Set
820c

EB: 2 Cost: 15 Freq: 8 822a	EB: 10 Cost: 17 Freq: 8 822b	EB: 9 Cost: 23 Freq: 4 822c	EB: 13 Cost: 30 Freq: 8 822d	EB: 3 Cost: 31 Freq: 7 822e	EB: 7 Cost: 33 Freq: 9 822f	EB: 12 Cost: 41 Freq: 8 822g

FIG. 5C

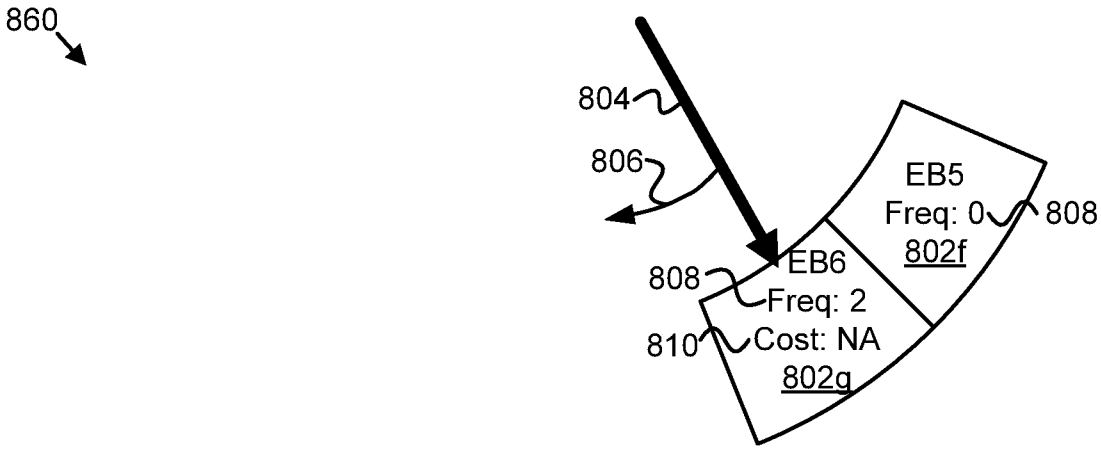
PAGE 8/13



Grooming Candidate Set
820d

EB: 2 Cost: 15 Freq: 8 822a	EB: 10 Cost: 17 Freq: 8 822b	EB: 9 Cost: 23 Freq: 4 822c	EB: 13 Cost: 30 Freq: 8 822d	EB: 3 Cost: 31 Freq: 7 822e	EB: 7 Cost: 33 Freq: 9 822f	EB: 12 Cost: 41 Freq: 8 822g

FIG. 5D

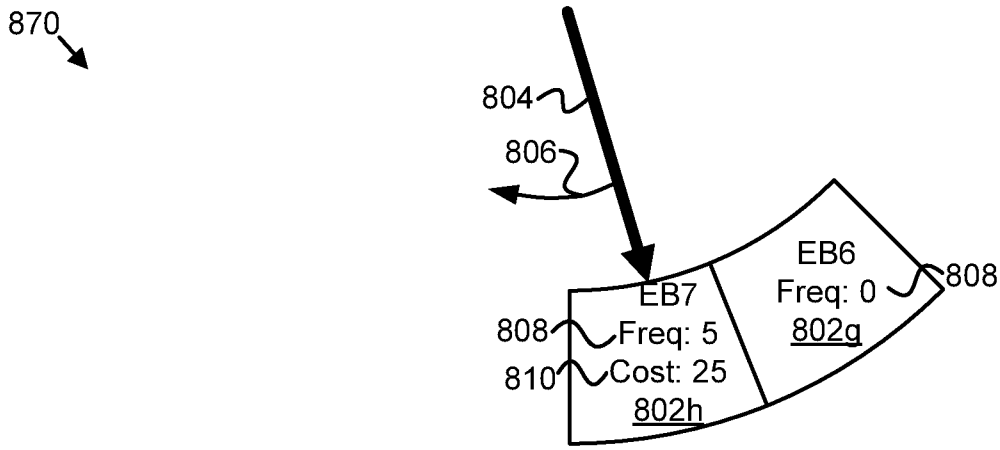


Grooming Candidate Set
820e

EB: 2 Cost: 15 Freq: 8 822a	EB: 10 Cost: 17 Freq: 8 822b	EB: 9 Cost: 23 Freq: 4 822c	EB: 13 Cost: 30 Freq: 8 822d	EB: 3 Cost: 31 Freq: 7 822e	EB: 7 Cost: 33 Freq: 9 822f	EB: 12 Cost: 41 Freq: 8 822g

FIG. 5E

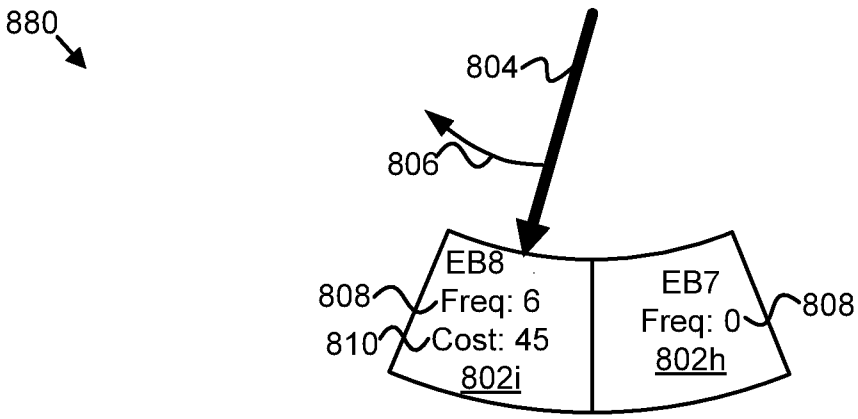
PAGE 9/13



Grooming Candidate Set
820f

EB: 10 Cost: 17 Freq: 8 822a	EB: 9 Cost: 23 Freq: 4 822b	EB: 7 Cost: 25 Freq: 5 822c	EB: 13 Cost: 30 Freq: 8 822d	EB: 3 Cost: 31 Freq: 7 822e	EB: 12 Cost: 41 Freq: 8 822f	822g

FIG. 5F



Grooming Candidate Set
820g

EB: 10 Cost: 17 Freq: 8 822a	EB: 9 Cost: 23 Freq: 4 822b	EB: 7 Cost: 25 Freq: 5 822c	EB: 13 Cost: 30 Freq: 8 822d	EB: 3 Cost: 31 Freq: 7 822e	EB: 12 Cost: 41 Freq: 8 822f	EB8 Cost: 45 Freq: 6 822g

FIG. 5G

PAGE 10/13

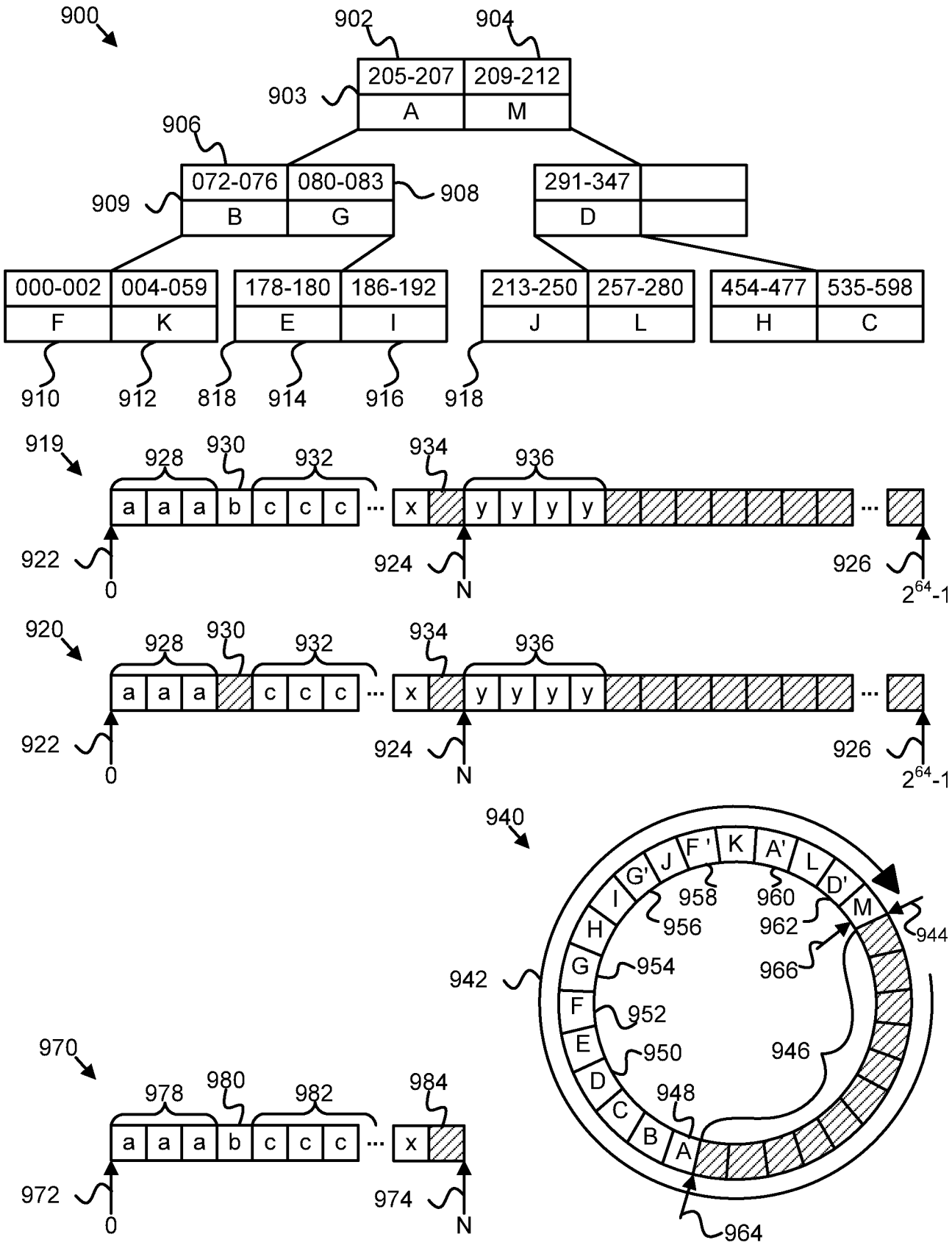


FIG. 6

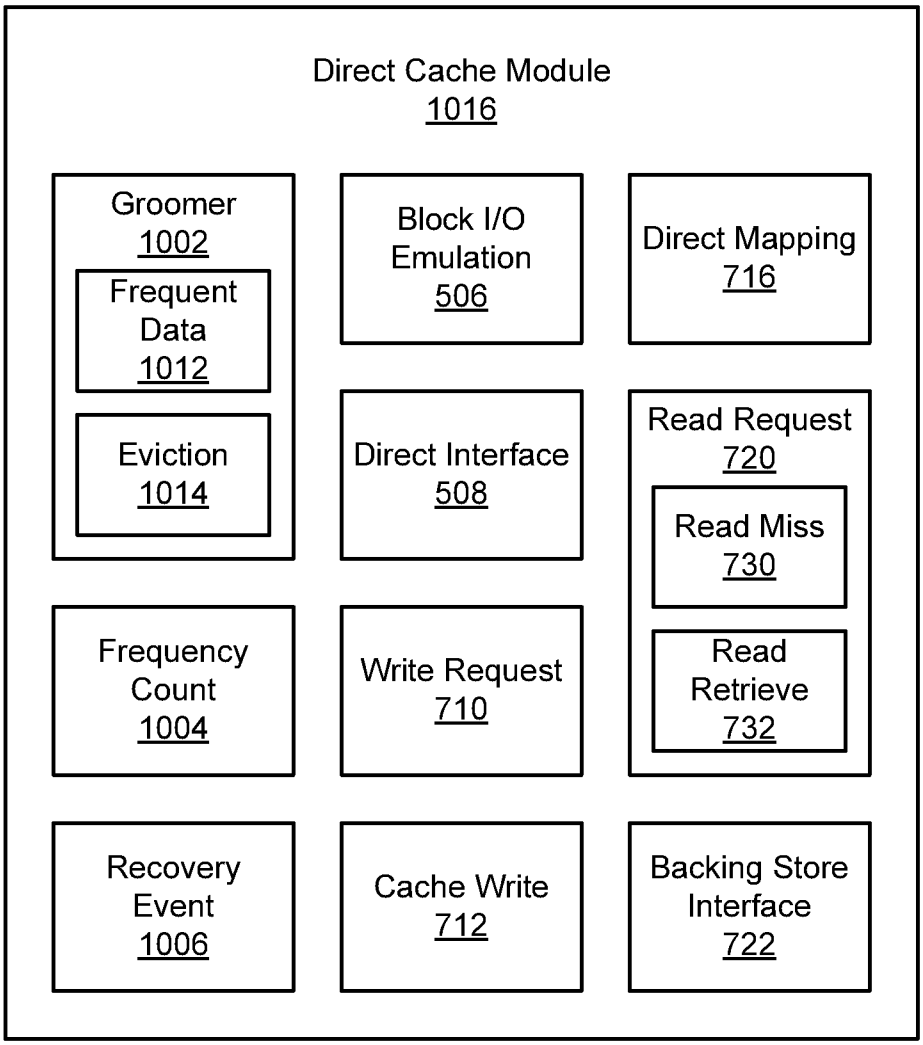

Direct Cache Module
1016
Groomer
1002
Frequent Data
1012
Eviction
1014
Block I/O Emulation
506
Direct Mapping
716
Direct Interface
508
Read Request
720
Read Miss
730
Read Retrieve
732
Frequency Count
1004
Write Request
710
Recovery Event
1006
Cache Write
712
Backing Store Interface
722

FIG. 7

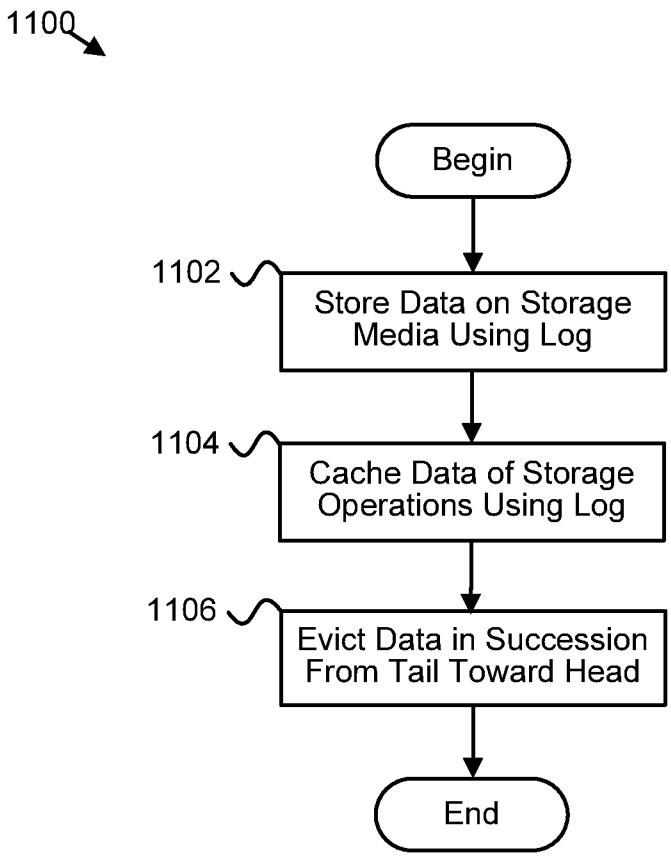

1100
Begin
1102
Store Data on Storage Media Using Log
1104
Cache Data of Storage Operations Using Log
1106
Evict Data in Succession From Tail Toward Head
End

Fig. 8

PAGE 13/13

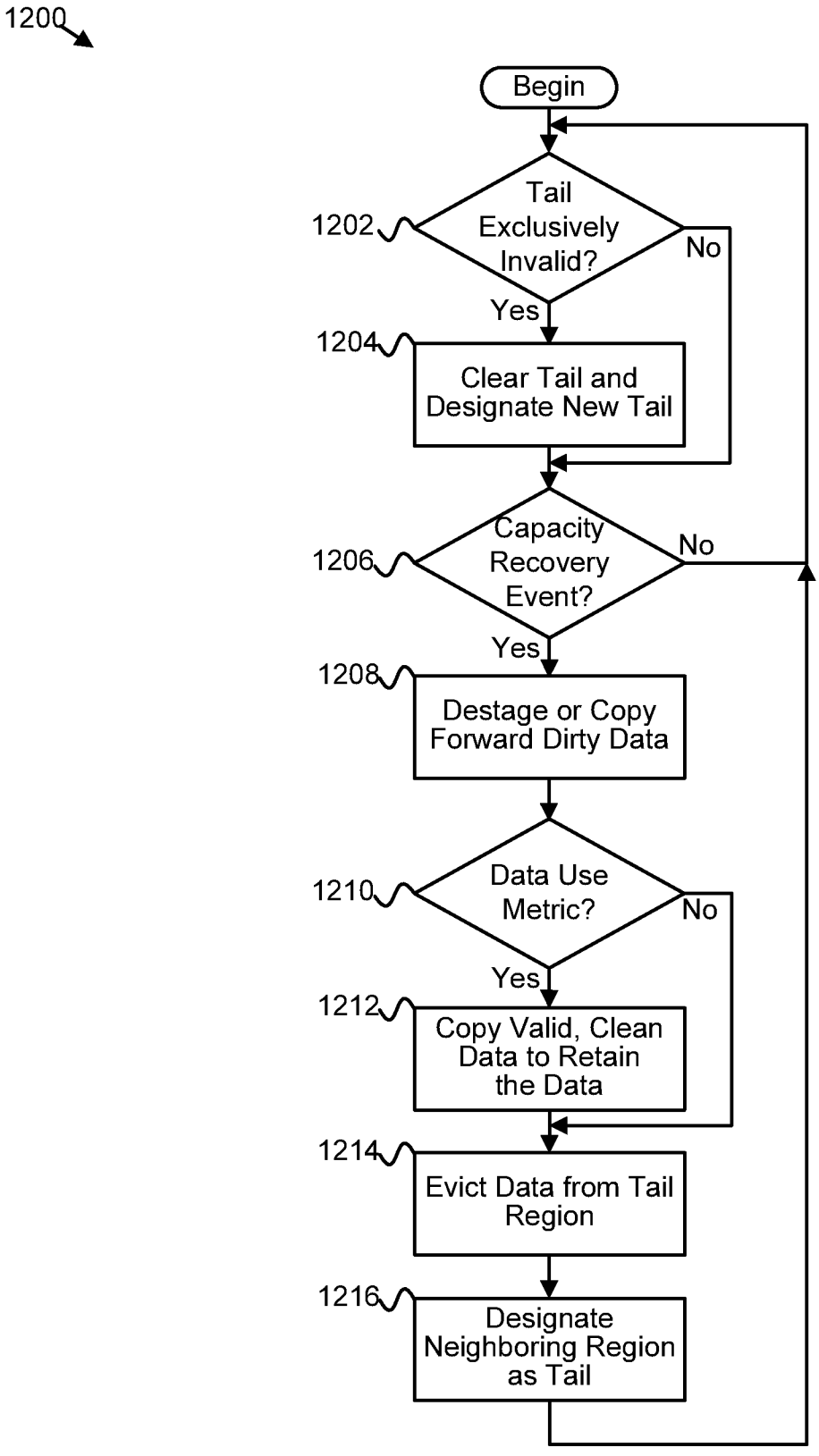


Fig. 9