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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
2 February 2012 (02.02.2012)

PCT

(10) International Publication Number
WO 2012/016209 A2

(51) International Patent Classification:
G06F 12/08 (2006.01) *G06F 12/16* (2006.01)

(21) International Application Number:
PCT/US2011/046005

(22) International Filing Date:
29 July 2011 (29.07.2011)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
12/847,952 30 July 2010 (30.07.2010) 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) Inventor; and

(75) Inventor/Applicant *(for US only)*: **FLYNN, David** [US/US]; 8856 Shady Meadow Drive, Sandy, Utah 84093 (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, RO, RS, RU, 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.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, 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))*

(54) Title: APPARATUS, SYSTEM, AND METHOD FOR REDUNDANT WRITE CACHING

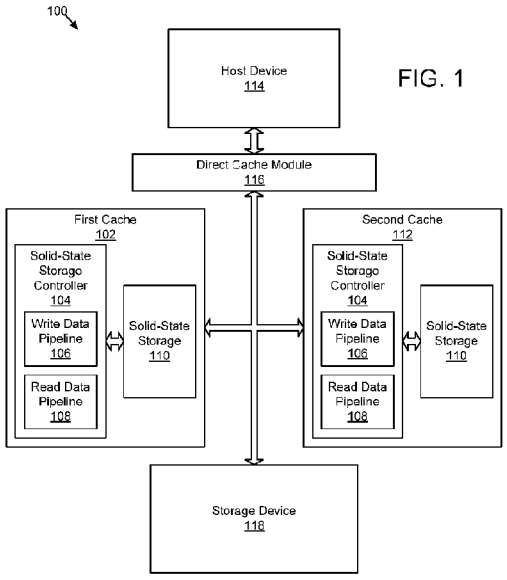

FIG. 1

(57) Abstract: An apparatus, system, and method are disclosed for redundant write caching. The apparatus, system, and method are provided with a plurality of modules including a write request module (602), a first cache write module (608), a second cache write module (610), and a trim module (606). The write request module (602) detects a write request to store data on a storage device (118). The first cache write module (608) writes data of the write request to a first cache (102). The second cache write module (610) writes the data to a second cache (112). The trim module 606 trims the data from one of the first cache (102) and the second cache (112) in response to an indicator that the storage device (118) stores the data. The data remains available in the other of the first cache (102) and the second cache (112) to service read requests.

WO 2012/016209 A2

APPARATUS, SYSTEM, AND METHOD FOR REDUNDANT WRITE CACHING

FIELD OF THE INVENTION

This invention relates to caching data and more particularly relates to caching write data for a data storage device.

DESCRIPTION OF THE RELATED ART

In general, caching data is advantageous because data that is accessed often or that is loaded as part of a software application or an operating system may be stored in a cache that has faster access times than a storage device that is used as a backing store with the cache. Because of the cache's faster access times, subsequent accesses can be much quicker than when the data must be accessed directly from the backing store storage device, such as a hard disk drive ("HDD"), an optical drive, tape storage, etc. Several caches are typically included in a computer device to speed data accesses for processors and other computer components.

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 systems for caching data. Accordingly, the present invention has been developed to provide a method, apparatus, and computer program product that overcome many or all of the above-discussed shortcomings in the art.

The method of the present invention is presented for write caching. The method in the disclosed embodiments includes the steps to carry out the functions of redundant write caching. In one embodiment, the method includes detecting a write request to store data on a storage device. The method, in a further embodiment, includes writing data of the write request to a first cache. In another embodiment, the method includes mirroring the data to a second cache. In one embodiment, the method includes clearing the data from one of the first cache and the second cache in response to an indicator that the storage device stores the data. The data, in one embodiment, remains available in the other of the first cache and the second cache to service read requests.

Reference throughout this specification to features, advantages, or similar language does not imply that all of the features and advantages that may be realized with the present invention should be or are in any single embodiment of the invention. Rather, language referring to the features and advantages is understood to mean that a specific feature, advantage, or characteristic described in connection with an embodiment is included in at least one embodiment of the

present invention. 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 present invention will become more fully apparent from the following description and appended claims, or may be learned by the practice of the invention 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 1 is a schematic block diagram illustrating one embodiment of a system for redundant write caching 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 embodiments of a mapping structure, a logical address space, a combined logical address space, a sequential, log-based append-only writing structure, and a storage device address space in accordance with the present invention;

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

Figure 5 is a schematic block diagram illustrating another embodiment of a direct cache module in accordance with the present invention;

Figure 6A is a schematic block diagram illustrating one embodiment of a first cache and a second cache in accordance with the present invention;

Figure 6B is a schematic block diagram illustrating another embodiment of a first cache and a second cache in accordance with the present invention;

Figure 6C is a schematic block diagram illustrating one embodiment of a first cache, a second cache, and a storage device in accordance with the present invention;

Figure 7 is a schematic flow chart diagram illustrating one embodiment of a method for redundant write caching in accordance with the present invention;

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

Figure 9 is a schematic flow chart diagram illustrating one embodiment of a method for

cleaning a cache 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 1 depicts one embodiment of a system 100 for redundant write caching in accordance with the present invention. The system 100, in the depicted embodiment, includes a first cache 102, a second cache 112, a host device 114, a direct cache module 116, and a storage device 118. The first cache 102 and the second cache 112, in the depicted embodiment, each include 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 mirrors data of write requests to both the first cache 102 and the second cache 112. In one embodiment, the system 100 virtually stripes cached data across the first cache 102 and the second cache 112 once the storage device 118 stores the data. The system 100, in one embodiment, preserves the benefits of redundant caching without decreasing the available storage space in the first cache 102 and the second cache 112 by half.

In the depicted embodiment, the system 100 includes two caches 102, 112. In another embodiment, the system 100 may include more than two caches 102, 112. In general, the first cache 102 and the second cache 112 serve as read and/or write caches for the storage device 118. In the depicted embodiment, the first cache 102 and the second cache 112 are each separate data storage devices. In a further embodiment, the first cache 102 and the second cache 112 may both be part of a single data storage device. For example, the first cache 102 and the second cache 112 may be separate partitions, regions, or other sections of a single data storage device.

In the depicted embodiment, the first cache 102 and the second cache 112 are each non-volatile, solid-state storage devices, 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"), phase change RAM ("PRAM"), etc. In further embodiments, the first cache 102

and/or the second cache 112 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.

In general, the first cache 102 and the second cache 112 cache data for the storage device 118. The storage device 118, in one embodiment, is a backing store associated with the first cache 102 and the second cache 112. The storage device 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 storage device 118 may have a greater data storage capacity than the first cache 102 and/or the second cache 112. In another embodiment, the storage device 118 may have a higher latency, a lower throughput, or the like, than the first cache 102 and/or the second cache 112.

The storage device 118 may have a higher latency, a lower throughput, or the like due to properties of the storage device 118 itself, or due to properties of a connection to the storage device 118. For example, in one embodiment, the first cache 102, the second cache 112, and the storage device 118 may each include non-volatile, solid-state storage media 110 with similar properties, but the storage device 118 may be in communication with the host device 114 over a data network, while the first cache 102 and the second cache 112 may be directly connected to the host device 114, causing the storage device 118 to have a higher latency relative to the host 114 than the first cache 102 and the second cache 112.

In the depicted embodiment, the first cache 102, the second cache 112, and the storage device 118 are in communication with the host device 114 through the direct cache module 116. One or more of the first cache 102, the second cache 112, and the storage device 118, in one embodiment, are 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 first cache 102, the second cache 112, and/or the storage device 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, one or more of the first cache 102, the second cache 112, and the storage device 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 first cache 102, the second cache 112, and/or the storage device 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 first cache 102 and/or the second cache 112 may be in the form of a dual-inline memory module ("DIMM"), a daughter card, or a micro-module. In another embodiment, the first cache 102 and/or the second cache 112 may be elements within a rack-mounted blade. In another embodiment, the first cache 102 and/or the second cache 112 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 first cache 102 and/or the second cache 112 are integrated directly onto a higher level assembly without intermediate packaging. In the depicted embodiment, the first cache 102 and the second cache 112 each includes one or more solid-state storage controllers 104 with a write data pipeline 106 and a read data pipeline 108 and each includes a solid-state storage media 110.

In a further embodiment, instead of being connected directly to the host device 114 as DAS, one or more of the first cache 102, the second cache 112, and the storage device 118 may be connected to the host device 114 over a data network. For example, the first cache 102, the second cache 112, and/or the storage device 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 one or more of the first cache 102, the second cache 112, and the storage device 118.

In one embodiment, at least one of the first cache 102 and the second cache 112 is connected directly to the host device 114 as a DAS device. In a further embodiment, the first cache 102 may be directly connected to the host device 114 as a DAS device and another device, such as the second cache 112 or the storage device 118, may be directly connected to the first cache 102. For example, the first cache 102 may be connected directly to the host device 114, and the second cache 112 and/or the storage device 118 may be connected directly to the first 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 first cache 102, the second cache 112, and the storage device 118 suitable for use in the system 100.

The system 100 includes the host device 114 in communication with the first cache 102, the second cache 112, and the storage device 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 for the storage device 118 and manages the caching of data in the first cache 102 and the second cache 112. In one embodiment, the direct cache module 116 comprises a software application, file system filter driver, or the like.

The direct cache module 116, in various embodiments, may include one or more software drivers on the host device 114, one or more storage controllers, such as the solid-state storage controllers 104 of the first cache 102 and the second cache 112, a combination of one or more software drivers and storage controllers, 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 one embodiment, the host device 114 loads one or more device drivers for the first cache 102, the second cache 112, and/or the storage device 118 and the direct cache module 116 communicates with the one or more device drivers on the host device 114. In another embodiment, the direct cache module 116 may communicate directly with a hardware interface of the first cache 102, the second cache 112, and/or the storage device 118. In a further embodiment, the direct cache module 116 may be integrated with the first cache 102, the second cache 112, and/or the storage device 118.

In one embodiment, the first cache 102, the second cache 112 and/or the storage device

118 have block device interfaces that support block device commands. For example, one or more of the first cache 102, the second cache 112, and the storage device 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 first cache 102, the second cache 112, and/or the storage device 118 using block device commands to read, write, and clear (or trim) data.

In one embodiment, the direct cache module 116 serves as a proxy for the storage device 118, receiving read and write requests for the storage device 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 storage device 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 both the first cache 102 and the second cache 112 or by retrieving read data from one of the first cache 102, the second cache 112, or the storage device 118 and returning the read data to the host device 114.

To provide increased data redundancy while also increasing the usable capacity of the first cache 102 and the second cache 112, in one embodiment, the direct cache module 116 caches data from a write request redundantly to both the first cache 102 and the second cache 112 and removes the data from one of the first cache 102 and the second cache 112 once the storage device 118 stores the data. The direct cache module 116, in one embodiment, leaves the data in the other of the first cache 102 and the second cache 112 so that the data remains available to service read requests from the host device 114.

In one embodiment, the direct cache module 116 determines which of the first cache 102 and the second cache 112 to leave the data in and which to remove the data from based on a deterministic protocol, so that the direct cache module 116 can determine which of the first cache 102 and the second cache 112 to send read requests and write requests to without maintaining and accessing a separate record of where data is cached. In further embodiments, the direct cache module 116 may send read requests and write requests to both the first cache 102 and the second cache 112 or maintain a data structure that includes a record of what data is cached in each of the first cache 102 and the second cache 112.

In one embodiment, the direct cache module 116 uses a deterministic protocol that is based on logical or physical addresses of data that the direct cache module 116 caches. For example, the direct cache module 116 may assign even addresses to the first cache 102 and odd

addresses to the second cache 112, or the like. This or a similar deterministic protocol, in one embodiment, logically and/or physically stripes data between the first cache 102 and the second cache 112 as the direct cache module 116 clears data from the first cache 102 and the second cache 112 in response to the storage device 118 storing the data.

In one embodiment, logical addresses in the first cache 102 and the second cache 112 correspond to logical and/or physical addresses in the storage device 118. For example, in one embodiment, the first cache 102 and the second cache 112 share a logical address space comprising the logical storage capacity of both the first cache 102 and the second cache 112. The logical address space may correspond to the physical address space of the storage device 118. Logical addresses of the first cache 102 and the second cache 112, in this embodiment, are directly mapped to corresponding physical addresses of the storage device 118.

Alternatively, in certain embodiments, logical addresses of the first cache 102 and the second cache 112 are directly mapped to corresponding logical addresses of the storage device 118. Directly mapping logical addresses of the first cache 102 and the second cache 112 to addresses of the storage device 118, in one embodiment, provides a one-to-one relationship between the addresses of the storage device 118 and the logical addresses of the first cache 102 and the second cache 112. Sharing a logical address space that is directly mapped to the logical or physical address space of the storage device 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, the shared logical address space of the first cache 102 and the second cache 112 is a sparse address space that is larger than the physical storage capacity of the first cache 102 and the second cache 112, either alone or together. This allows the storage device 118 to have a larger storage capacity than the first cache 102 and the second cache 112, while maintaining a direct mapping between the logical addresses of the first cache 102 and the second cache 112 and logical or physical addresses of the storage device 118. The shared 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 first cache 102 and the second cache 112 using logical addresses, the first cache 102 and the second cache 112 directly map the logical addresses to distinct physical addresses on the first cache 102 and the second cache 112. As the direct cache module 116 clears data from one or the other of the first cache 102 and the second cache 112, the physical addresses and associated physical storage media, the solid state storage media 110 in the depicted embodiment, are freed to store data for other logical addresses.

In another embodiment, logical addresses of a single cache, such as the first cache 102 or

the second cache 112, are directly mapped to corresponding logical addresses of the storage device 118. Directly mapping logical addresses of the single cache to addresses of the storage device 118, in one embodiment, provides a one-to-one relationship between the addresses of the storage device 118 and the logical addresses of the first cache 102 and the second cache 112. Sharing a logical address space that is directly mapped to the logical address space or physical address space of the storage device 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, the shared logical address space of the single cache is a sparse address space that is larger than the physical storage capacity of the single cache. This allows the storage device 118 to have a larger storage capacity than the single cache, while maintaining a direct mapping between the logical addresses of the single cache and logical addresses and/or physical addresses of the storage device 118. The shared 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 single cache using logical addresses, the single cache directly maps the logical addresses to distinct physical addresses on the single cache. As the direct cache module 116 clears data from the single cache, the physical addresses and associated physical storage media, the solid state storage media 110 in the depicted embodiment, are freed to store data for other logical addresses. In one embodiment, the physical addresses and associated physical storage media store the data using a log-based, append only writing structure such that data evicted from the cache or overwritten by a subsequent write request invalidates other data in the log such that a garbage collection process can recover the physical capacity.

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 Figure 1. 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 Figure 1, described above. The direct cache module 116, in the depicted embodiment, is in communication with the first cache 102, the second cache 112, and the storage device 118.

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 first cache 102, the second cache 112, and the storage device 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 one or more of the first cache 102, the second cache 112, and the storage device 118 directly. The storage client 504, in one embodiment, reads data from and writes data to the storage device 118 through the direct cache module 116, which uses the first cache 102 and the second cache 112 to cache read data and write data for the storage device 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 first cache 102, the second cache 112, and the storage device 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. As described above with regard to the direct cache module 116 depicted in the embodiment of Figure 1, in various embodiments, the direct cache module 116 may be embodied by one or more of a storage controller of the first cache 102, a storage controller of the second cache 112, and/or a storage controller of the storage device 118; a separate hardware controller device that interfaces with the first cache 102, the second cache 112, and the storage device 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 first cache 102, the second cache 112, and/or the storage device 118. The direct cache module 116 may communicate with the first cache 102, the second cache 112, and/or the storage device 118 through device drivers loaded on the host device 114, through a storage controller of the first cache 102, a storage controller of the second cache 112, and/or a storage controller of the storage device 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, 112, and/or one or more backing stores 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,

one or more of the first cache 102, the second cache 112, and the storage device 118 either include a distinct block I/O emulation layer 506 or are 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 managed by the storage manager. 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 first cache 102, the second cache 112, and/or the storage device 118 may not directly or necessarily associate logical block addresses with particular physical blocks. The direct cache module 116, the first cache 102, the second cache 112, and/or the storage device 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 first cache 102, the second cache 112, and/or the storage device 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 first cache 102, the second cache 112, and/or the storage

device 118 with the storage client 504. Similarly, the direct cache module 116, in one embodiment, may communicate with the first cache 102, the second cache 112, and/or the storage device 118 through direct interface layers 508.

A direct cache module 116 using the direct interface 508 may store data on the first cache 102, the second cache 112, and/or the storage device 118 as blocks, sectors, pages, logical blocks, logical pages, erase blocks, logical erase blocks, ECC chunks or in any other format or structure advantageous to the technical characteristics of the first cache 102, the second cache 112, and/or the storage device 118. For example, in one embodiment, the storage device 118 comprises a hard disk drive and the direct cache module 116 stores data on the storage device 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 first cache 102, the second cache 112, and/or the storage device 118. The direct cache module 116, the first cache 102, the second cache 112, and/or the storage device 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 first cache 102 and the second cache 112 each include a logical-to-physical translation layer 510. In a further embodiment, the storage device 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 first cache 102, the second cache 112, and the storage device 118. In another embodiment, the direct cache module 116 maintains a distinct logical-to-physical translation layer 510 for each of the first cache 102, the second cache 112, and the storage device 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 first cache 102, the second cache 112, and/or the storage device 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 first cache 102 and on the media of the second cache 112. 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 first cache 102 and the second cache 112, but is an abstract

reference to the data.

In one embodiment, the first cache 102 and the second cache 112 share a logical address space. The direct cache module 116, in one embodiment, mirrors data corresponding to a logical address in the shared address space to both the first cache 102 and the second cache 112, so that both the first cache 102 and the second cache 112 can service subsequent read requests for the data of the logical address. In a further embodiment, the direct cache module 116 may assign separate portions of the logical address space to the first cache 102 and the second cache 112, so that the first cache 102 and the second cache 112 each cache data corresponding to distinct portions of the shared address space. For example, in one embodiment, the direct cache module 116 may divide the shared logical address space into even and odd addresses and assign the even addresses to one of the first cache 102 and the second cache 112 and assign the odd addresses to the other of the first cache 102 and the second cache 112.

In a further embodiment, the direct cache module 116 initially mirrors write request data to both the first cache 102 and the second cache 112, regardless of a logical address corresponding to the write request, and transitions to caching data in one of the first cache 102 and the second cache 112 based on logical addresses as the write request data is subsequently written to the storage device 118. At least a portion of the shared address space of the first cache 102 and the second cache 112, in one embodiment, corresponds to a physical or logical address space for the storage device 118, so that the direct cache module 116 can use the same address identifier for data cached in the first cache 102 and the second cache 112 and for data stored in the storage device 118, without additional mapping or translation.

In the depicted embodiment, the first cache 102 and the second cache 112 manage the physical block addresses in the distinct, separate physical address spaces of the first cache 102 and the second cache 112. 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 of the first cache 102 and the physical media of the second cache 112 at which to perform data operations.

Furthermore, in one embodiment, the logical address space of the first cache 102 and the second cache 112 is substantially larger than the physical address space. 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, sparse address space also allows the first cache 102 and the second cache 112 to cache data for a storage device 118 with a larger address space (i.e. a larger storage capacity) than the combined physical address spaces of the first cache 102 and the second cache 112.

In one embodiment, the logical-to-physical translation layers 510 each include a map or index that maps logical block addresses to physical block addresses. The map 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 remains 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 506 layer, the storage client 504 may erroneously believe that the direct cache module 116, the first cache 102, the second cache 112, and/or the storage device 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 first cache 102, the second cache 112, and/or the storage device 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 first cache 102, the second cache 112, and/or the storage device 118 corresponding to the physical block. As the number of allocated blocks increases, the performance of the first cache 102, the second cache 112, and/or the storage device 118 may suffer depending on the configuration of the first cache 102, the second cache 112, and/or the storage device 118.

Specifically, in certain embodiments, the first cache 102, the second cache 112, and/or the storage device 118 is 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 first cache 102, the second cache 112, and/or the storage device 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 a single cache, either first cache 102 or second cache 112, 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 first cache 102, the second cache 112, and/or the storage device 118 achieves a high write throughput and a high number of I/O operations per second (IOPS). The first cache 102, the second cache 112, and/or the storage device 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.

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 first cache 102, the second cache 112, and/or the storage device 118 may relocate valid blocks 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 first cache 102, the second cache 112, and/or the storage device 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. 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 first cache 102, the second cache 112, and/or the storage device 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 first cache 102, the second cache 112, and/or the storage device 118 such that subsequent requests for data in the physical block return a null result or an empty set of data.

One example of a command or message for such deallocation is the "TRIM" function of the "Data Set Management" command under the T13 technical committee command set specification maintained by INCITS. 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 first cache 102, the second cache 112, and/or the storage device 118. In one embodiment, the direct cache module 116 clears data that is mirrored to both the first cache 102 and the second cache 112 from one of the caches 102, 112 in response to the storage device 118 storing the data. As used herein, clearing or trimming 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 data from one of the first cache 102 and the second cache 112 in response to the storage device 118 storing the data frees storage capacity in the first cache 102 and the second cache 112 to cache more data for the storage device 118 while maintaining the benefits of redundant write caching.

Figure 3 depicts one embodiment of a mapping structure 1000, a logical address space 1020 of the first cache 102 and/or the second cache 112, a combined logical address space 1019

that is accessible to a storage client, a sequential, log-based, append-only writing structure 1040, and a storage device address space 1070 of the backing store storage device 118. The mapping structure 1000, in one embodiment, is maintained by the direct cache module 116 and/or the logical-to-physical translation layer 510. The mapping structure 1000, in the depicted embodiment, is a B-tree. Further, instead of links that map to entries in a reverse map, the nodes of the mapping structure 1000 include direct references to physical locations in the first cache 102 and/or the second cache 112. The mapping structure 1000, in various embodiments, may be used either with or without a reverse map. The references in the mapping structure 1000 may include alpha-numerical characters, hexadecimal characters, pointers, links, and the like.

The mapping structure 1000, 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 1000 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 first cache 102 and/or the second cache 112 to a physical location in the storage media 110 for the first cache 102 and/or the second cache 112. 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 first cache 102 and/or the second cache 112 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 first cache 102 and/or the second cache 112. In the depicted embodiment, the capital letters 'A' through 'M' are also depicted in the writing structure 1040 which represents the physical storage media 110 of the first cache 102 and/or the second cache 112. 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 1000 denotes membership (or storage) in the first cache 102 and/or the second cache 112. In another embodiment, an entry may further include an indicator of whether the first cache 102 or the

second cache 112 stores data corresponding to a logical block within the range of logical addresses, data of the reverse map 922 described above, and/or other data.

In the depicted embodiment, the root node 908 includes entries 1002, 1004 with noncontiguous ranges of logical addresses. A "hole" exists at logical address "208" between the two entries 1002, 1004 of the root node. In one embodiment, a "hole" indicates that the first cache 102 and/or the second cache 112 do not store data corresponding to one or more logical addresses corresponding to the "hole." In one embodiment, the first cache 102 and/or the second cache 112 support 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."

In the embodiment depicted in Figure 3, a garbage collection module or a trim module, such as the garbage collection module 710 described below with regard to Figure 5 or the trim module 606 described below with regard to Figure 4, trims data of a single logical address "208" and splits the range of logical addresses into two separate entries 1002, 1004. In one embodiment, the logical-to-physical translation layer 510 may rebalance the mapping structure 1000, 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 first cache 102 and/or the second cache 112 to store dynamically selected and/or variable lengths of logical block ranges.

In the depicted embodiment, similar "holes" or noncontiguous ranges of logical addresses exist between the entries 1006, 1008 of the node 914, between the entries 1010, 1012 of the left child node of the node 914, between entries 1014, 1016 of the node 918, and between entries of the node 1018. 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 1006 of the node 914 and the right entry 1012 of the left child node of the node 914.

The "hole" at logical address "003," in the depicted embodiment, can also be seen in the logical address space 1020 at logical address "003" 1030. The hash marks at logical address "003" 1040 represent an empty location, or a location for which the first cache 102 and/or the second cache 112 does not store data. In the depicted embodiment, storage device address "003" 1080 of the storage device address space 1070 does store data (identified as 'b'), indicating that a garbage collection module or a trim module, such as the garbage collection module 710

described below with regard to Figure 5 or the trim module 606 described below with regard to Figure 4, evicted data from logical address "003" 1030 of the first cache 102 and/or the second cache 112. The "hole" at logical address 1034 in the logical address space 1020, however, has no corresponding data in storage device address 1084, indicating that the "hole" is due to one or more block I/O requests with noncontiguous ranges, a trim or other deallocation command to the first cache 102, the second cache 112, and the storage device 118, or the like.

The "hole" at logical address "003" 1030 of the logical address space 1020, however, in one embodiment, is not viewable or detectable to a storage client. In the depicted embodiment, the combined logical address space 1019 represents the data that is available to a storage client, with data that is stored in the first cache 102, data that is stored in the second cache 112, and data that is stored in the storage device 118 but not in the first cache 102 or the second cache 112. As described below, the read miss module 718 of Figure 5 handles misses and returns requested data to a requesting entity. In the depicted embodiment, if a storage client requests data at logical address "003" 1030, a read miss module 718 will retrieve the data from the storage device 118, as depicted at address "003" 1080 of the storage device address space 1070, and return the requested data to the storage client. In other embodiments, if a miss is for data that the storage device 118 does not store, and one or more corresponding logical addresses are unallocated, the first cache 102, the second cache 112, or the direct cache module 116 may return zeros, an empty set, an error, or the like to the requesting storage client.

The logical address space 1020 of the first cache 102 and/or the second cache 112, in the depicted embodiment, is larger than the physical storage capacity and corresponding storage device address space 1070 of the storage device 118. In the depicted embodiment, the first cache 102 and/or the second cache 112 have a 64 bit logical address space 1020 beginning at logical address "0" 1022 and extending to logical address "264-1" 1026. The storage device address space 1070 begins at storage device address "0" 1072 and extends to storage device address "N" 1074. Storage device address "N" 1074, in the depicted embodiment, corresponds to logical address "N" 1024 in the logical address space 1020 of the first cache 102 and/or the second cache 112. Because the storage device address space 1070 corresponds to only a subset of the logical address space 1020 of the first cache 102 and/or the second cache 112, the rest of the logical address space 1020 may be shared with an additional cache, may be mapped to a different storage device 118, may store data in the first cache 102 and/or the second cache 112 (such as a non-volatile memory cache) that is not stored in the storage device 1070, or the like.

For example, in the depicted embodiment, the first range of logical addresses "000-002" 1028 stores data corresponding to the first range of storage device addresses "000-002" 1078.

Data corresponding to logical address "003" 1030, as described above, is unallocated in the first cache 102 and/or the second cache 112 forming a "hole." The second range of logical addresses "004-059" 1032 corresponds to the second range of storage device addresses "004-059" 1082. However, the final range of logical addresses 1036 extending from logical address "N" 1024 extends beyond storage device address "N" 1074. No storage device address in the storage device address space 1070 corresponds to the final range of logical addresses 1036. The sequential, log-based, append-only writing structure 1040, in the depicted embodiment, is a logical representation of the log preserved in the physical storage media 110 of the first cache 102 and/or the second cache 112. In certain embodiments, the first cache 102 and/or the second cache 112 store data sequentially, appending data to the writing structure 1040 at an append point 1044. The first cache 102 and/or the second cache 112, in a further embodiment, use a storage space recovery process, such as the trim module 606 and/or the garbage collection module 710 (described below with regard to Figures 4 and 5) that re-uses non-volatile storage media 110 storing deallocated, evicted, or unused logical blocks. Non-volatile storage media 110 storing deallocated, evicted, or unused logical blocks, in the depicted embodiment, is added to an available storage pool 1046 for the first cache 102 and/or the second cache 112. By clearing certain data from the first cache 102 and/or the second cache 112 and adding the physical storage capacity corresponding to the cleared data back to the available storage pool 1046, in one embodiment, the writing structure 1040 is ring-like and has a theoretically infinite capacity.

In the depicted embodiment, the append point 1044 progresses around the log-based, append-only writing structure 1040 in a circular pattern 1042. In one embodiment, the circular pattern 1042 wear balances the solid-state storage media 110, increasing a usable life of the solid-state storage media 110. In the depicted embodiment, the trim module 606 and/or the garbage collection module 710 described below with regard to Figures 4 and 5 have marked several blocks 1048, 1050, 1052, 1054 as invalid, represented by an "X" marking on the blocks 1048, 1050, 1052, 1054. The garbage collection module 710, in one embodiment, will recover the physical storage capacity of the invalid blocks 1048, 1050, 1052, 1054 and add the recovered capacity to the available storage pool 1046. In the depicted embodiment, modified versions of the blocks 1048, 1050, 1052, 1054 have been appended to the writing structure 1040 as new blocks 1056, 1058, 1060, 1062 in a read, modify, write operation or the like, allowing the original blocks 1048, 1050, 1052, 1054 to be recovered. In further embodiments, the garbage collection module 710 may copy forward to the append point 1044 any dirty data and selectively any valid data that the blocks 1048, 1050, 1052, 1054 store, if any.

Figure 4 depicts one embodiment of the direct cache module 116. In the depicted embodiment, the direct cache module 116 includes a write request module 602, a cache write module 604, and a trim module 606. The direct cache module 116 of Figure 4, in one embodiment, is substantially similar to the direct cache module 116 described above with regard to Figure 1 and/or Figure 2. In general, the direct cache module 116 mirrors dirty data that the storage device 118 does not yet store to provide redundancy for the data. Once the direct cache module 116 or another entity writes the data to the storage device 118, in general, the direct cache module 116 discards the mirror of the data to increase caching capacity of the direct cache module 116.

In one embodiment, the write request module 602 detects a write request to store data on the storage device 118. The write request module 602 may detect the 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 storage device 118), or the like. In one embodiment, the host device 114 sends the write request. The direct cache module 116, 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 602.

A write request, in one embodiment, includes data that is not stored on the storage device 118. Data that is not stored on the storage device 118, in various embodiments, includes new data not yet stored on the storage device 118, modifications to data that is stored on the storage device 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 storage device 118 by way of a Direct Memory Access ("DMA") or Remote DMA ("RDMA") operation. 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 first cache 102, the second cache 112, and/or the storage device 118. The write request module 602, in one embodiment, sends the data of the detected write request and the corresponding address or set of addresses to the first cache write module 608 and/or the second cache write module 610 for caching the data in the first cache 102 and the second cache 112. The write request module 602 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 604, in one embodiment, writes data of the write request to the

first cache 102 and to the second cache 112, mirroring the data. In the depicted embodiment, the cache write module 604 includes a first cache write module 608 and a second cache write module 610. In one embodiment, the first cache write module 608 writes the data of the write request to the first cache 102 and the second cache write module 610 writes the data of the write request to the second cache 112. The first cache write module 608 and the second cache write module 610, in one embodiment, write the data substantially simultaneously. In a further embodiment, the first cache write module 608 may write the data and transmit the data to the second cache write module 610 for mirroring, or vice versa.

The trim module 606, in one embodiment, transitions data stored in the first cache 102 and the second cache 112 from a mirrored state to a logically striped state, conserving storage capacity in the first and second caches 102, 112 once the storage device 118 stores the data. In one embodiment, the trim module 606 transitions data from a mirrored state to a striped state by clearing or trimming the data from either one of the first cache 102 and the second cache 112 in response to an indicator that the storage device 118 stores the data. The data that the trim module 606 clears or trims, in one embodiment, remains available in the other of the first cache 102 and the second cache 112 to service read requests.

The trim module 606, in one embodiment, receives a communication that includes an indicator that the storage device 118 stores the data. For example, in one embodiment, a cleaner module such as the cleaner module 708 described below with regard to Figure 5 may send an indicator to the trim module 606 that the storage device 118 stores the data, the data has been persisted. In a further embodiment, the trim module 606 queries the storage device 118 to determine that the storage device 118 stores the data (e.g. by way of an indicator), scan the storage device 118 for the data, or the like.

In a further embodiment, the trim module 606 may query or scan the first cache 102 and/or the second cache 112 for an indicator that the data in the caches 102, 112 is clean and is thus stored in the storage device 118. In another embodiment, the trim module 606 checks a clean/dirty data structure that includes an indicator whether the storage device 118 stores the data. The trim module 606, in a further embodiment, references indicators in a mapping of logical addresses to physical media addresses to determine whether the storage device 118 stores the data. In light of this specification, other indicators suitable for use by the trim module 606 will be apparent to one of skill in the art, and are considered embodiments of the present invention.

In one embodiment, the trim module 606 delays clearing the data for a period of time after the indicator that the storage device 118 stores the data, until after a predefined event

occurs. Delaying clearing the data, in various embodiments, may decrease a processing load of the first cache 102 and/or the second cache 112, may provide redundant caching of the data for a longer period of time, may increase read performance for the data for a longer period of time, or the like. For example, the trim module 606 may clear the data in response to a predefined event such as a lack of available storage capacity in the first cache 102 and/or the second cache 112, a scheduled cache clearing cycle, a cache ejection event, or another predefined event. A cache clearing cycle, in one embodiment, is a periodic or dynamic procedure that the trim module 606 may perform to routinely clear data from the first cache 102 and/or the second cache 112. A cache clearing cycle may be scheduled as a periodic maintenance procedure, scheduled dynamically in response to the occurrence of a predefined event, or the like.

In a further embodiment, the trim module 606 clears or trims data from a cache 102, 112 in response to an indicator that the data is stored in the storage device 118by sending a TRIM command to a selected cache 102, 112 with an address (or set of addresses, such as a range of addresses) of data to be trimmed. In one embodiment, a cache selection module (discussed below with regard to Figure 5) selects one of the first cache 102 and the second cache 112 for the trim module 606 to clear or trim the data from. In various embodiments, the trim module 606 may clear data exclusively from the first cache 102, exclusively from the second cache 112, alternating between the first cache 102 and the second cache 112, based on a logical address of the data, according to a deterministic protocol, based on one or more attributes of the first cache 102 and the second cache 112, or the like.

Figure 5 depicts another embodiment of the direct cache module 116. In the depicted embodiment, the direct cache module 116 includes the block I/O emulation layer 506, the direct interface layer 508, the write request module 602, the cache write module 604, and the trim module 606, substantially as described above with regard to Figures 2 and 4. The direct cache module 116, in the depicted embodiment, further includes a cache selection module 702, a read request module 704, a write acknowledgement module 706, a cleaner module 708, a garbage collection module 710, a cache replace module 712, and a storage device interface module 714.

In one embodiment, the cache selection module 702 selects either the first cache 102 or the second cache 112 for the trim module 606 so that the trim module 606 can clear data from the selected cache 102, 112. In a further embodiment, the cache selection module 702 selects a cache 102, 112 for the trim module 606 to trim data by selecting either the first cache 102 or the second cache 112 in which to maintain the data. In the depicted embodiment, the cache selection module 702 includes a deterministic protocol module 714 and a cache attribute module 716.

In one embodiment, the cache selection module 702 uses the deterministic protocol

module 714 to select either the first cache 102 or the second cache 112 according to a deterministic protocol. A deterministic protocol, as used herein, is a predictable, repeatable pattern. In an embodiment where the deterministic protocol module 714 selects one of the first cache 102 and the second cache 112 based on a deterministic protocol, other modules, such as the read request module 704, can determine which of the first cache 102 and the second cache 112 stores data based on the deterministic protocol, without maintaining a separate data structure tracking locations for data within the caches 102, 112 and without querying the first cache 102 and/or the second cache 112 to determine which cache 102, 112 stores specific data.

In one embodiment, the first cache 102 and the second cache 112 share a logical address space, and the deterministic protocol module 714 uses a deterministic protocol that is based on a logical address that is associated with data. For example, the deterministic protocol module 714, in one embodiment, divides logical addresses into two groups and assigns one group to the first cache 102 and assigns the second group to the second cache 112. Examples of groups of logical addresses include even logical addresses and odd logical addresses, high logical addresses and low logical addresses, and the like.

In one embodiment, the deterministic protocol module 714 logically stripes data between the first cache 102 and the second cache 112 by alternating selection of the first cache 102 and the second cache 112 for consecutive logical addresses (or ranges of addresses), effectively assigning even logical addresses to one cache 102, 112 and odd logical addresses to the other cache 102, 112. Data is logically striped across the first cache 102 and the second cache 112, in one embodiment, because data corresponding to consecutive logical addresses is stored in different caches 102, 112. In an embodiment where the first cache 102 and the second cache 112 map logical addresses to distinct physical addresses, data that is logically striped between the caches 102, 112 may or may not be physically striped on the storage media 110 of the first cache 102 and the storage media 110 of the second cache 112. Logical striping of data, in one embodiment, retains the load balancing, capacity, and other advantages of physical data striping.

In another embodiment, the deterministic protocol module 714 exclusively selects the first cache 102, exclusively selects the second cache 112, or the like. For example, in various embodiments, one of the caches 102, 112 may have a smaller storage capacity, may have lower reliability, may have a different type of storage media 110, may have a lower data throughput, or the like and the deterministic protocol module 714 may exclusively select that cache for the trim module 606 to clear data. In other embodiments, the deterministic protocol module 714 uses another type of deterministic protocol, such as a hash function or the like, to predictably select one of the first cache 102 and the second cache 112.

In another embodiment, the deterministic protocol module 714 assigns logical addresses that correspond to addresses of the storage device 118 to the first cache 102 and assigns logical addresses that are outside the physical capacity of the storage device 118 to the second cache 112. The deterministic protocol module 714, in one embodiment, maps the logical addresses that correspond to addresses of the storage device 118, which are assigned to the first cache 102, to the logical addresses that are outside the physical capacity of the storage device 118. The deterministic protocol module 714, in various embodiments, may perform the mapping by adding an offset value to a logical address, maintaining a mapping data structure, shifting the bits of a logical address, adding a prefix to a logical address, performing another deterministic binary operation on a logical address, or the like.

In one embodiment, the range of logical addresses that correspond to addresses of the storage device 118 are at a lower end of the logical address space, with logical addresses that are outside the physical capacity of the storage device 118 at the higher end of the logical address space. For example, in one embodiment, the first cache 102 and the second cache 112 share a logical address space that is 64 bit addressable and the storage device 118 has a physical capacity, such as 2 terabytes or the like, that only needs 32 bit logical block addresses. In the example embodiment, the deterministic protocol module 714 may assign logical addresses 0 through 232 to the first cache 102 and assign the remaining logical addresses, 232 through 264 to the second cache 112. Because the higher range of logical addresses is much larger than the lower range of logical addresses, in the example embodiment, the deterministic protocol module 714 may map the lower range of logical addresses to any set of logical addresses in the higher range.

In one embodiment, the cache selection module 702 uses the cache attribute module 716 to select one of the first cache 102 and the second cache 112 based on one or more cache attributes. Cache attributes, in various embodiments, include the type of storage media, such as SLC or MLC type solid-state storage media 110, erase cycle counts, error rates, age, fault history, total storage capacity, remaining storage capacity, access times, and the like. The cache attribute module 716, in one embodiment, weights or otherwise combines several cache attributes to select one of the first cache 102 and the second cache 112. The cache attributes, in one embodiment, may include user preferences, user settings, user prioritization, and/or user rankings for the first cache 102 and the second cache 112. The cache attributes, in a further embodiment, are user selectable. For example, in one embodiment, the cache selection module 702 may receive user input from a user of the host device 114, or the like, indicating which cache attributes to use, indicating weights for various cache attributes, selecting threshold values for

various cache attributes, and the like.

In one embodiment, the cache selection module 702 includes one of the deterministic protocol module 714 and the cache attribute module 716 and not the other. In a further embodiment, the cache selection module 702 includes both the deterministic protocol module 714 and the cache attribute module 716. In another embodiment, the cache selection module 702 does not use the deterministic protocol module 714 or the cache attribute module 716, but selects one of the caches 102, 112 in another manner, such as making a random or pseudorandom selection, or the like.

In an embodiment where the cache selection module 702 does not use the deterministic protocol module 714, for example using the cache attribute module 716 or selecting a cache 102, 112 in another manner, the cache selection module 702 may maintain a mapping indicating which of the first cache 102 and the second cache 112 stores which data according to logical addresses or ranges of logical addresses. The mapping, in various embodiments, may include an index, an array, a linked-list, a look-up table, a b tree, a CAM, a binary tree, a hash table, or another mapping data structure. In one embodiment, the mapping maps a logical address of data to the first cache 102, the second cache 112, or both. In a further embodiment, the cache selection module 702 does not maintain a mapping or use the deterministic protocol module 714, and instead queries one or both of the first cache 102 and the second cache 112 to determine which cache 102, 112 stores data corresponding to a specific logical address.

In one embodiment, the read request module 704 services read requests for data stored in the first cache 102, the second cache 112, and/or the storage device 118. The read request module 704, in one embodiment, detects a read request to retrieve requested data from the storage device 118. In a further embodiment, the read request module 704 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.

The read request module 704, in one embodiment, directs a read request to one of the first cache 102 and the second cache 112 based on a selection by the cache selection module 702. For example, in an embodiment where the cache selection module 702 selects one of the first cache 102 and the second cache 112 for clearing of data based on a deterministic protocol the read request module 704 directs a read request for the data to the other of the first cache 102 and the second cache 112 that was not selected by the cache selection module 702 for clearing. In a further embodiment, the cache selection module 702 may maintain a mapping of which cache 102, 112 stores which data, or may query the first cache 102 and/or the second cache 112 to determine which cache 102, 112 stores specific data, and the read request module 704 may direct

a read request to the appropriate cache 102, 112.

In the depicted embodiment, the read request module 704 includes a read miss module 718 and a read retrieve module 720. The read miss module 718, in one embodiment, determines whether or not requested data is stored in either the first cache 102 or the second cache 112. The read miss module 718 may query the first cache 102 and/or the second cache 112 directly, query one of the first cache 102 and the second cache 112 based on a deterministic protocol used by the cache selection module 702, query a mapping of which cache 102, 112 stores which data, or the like to determine whether or not requested data is stored in either the first cache 102 or the second cache 112.

The read retrieve module 720, in one embodiment, returns requested data to the requesting entity, such as the host device 114. If the read miss module 718 determines that either the first cache 102 or the second cache 112 stores the requested data, in one embodiment, the read retrieve module 720 reads the requested data from the determined cache 102, 112 and returns the data to the requesting entity. If the read miss module 718 determines that neither the first cache 102 or the second cache 112 stores the requested data, in one embodiment, the read retrieve module 720 reads the requested data from the storage device 118, writes the requested data to at least one of the first cache 102 and the second cache 112, and returns the requested data to the requesting entity. In one embodiment, the cache selection module 702 selects one of the first cache 102 and the second cache 112 for the read retrieve module 720 to write the requested data to, using the deterministic protocol module 714, the cache attribute module 716, or the like.

In one embodiment, the read retrieve module 720 caches data of a read request in one of the first cache 102 and the second cache 112 based on a deterministic protocol, using the deterministic protocol module 714, or the like. In a further embodiment, the read retrieve module 720 alternates between selection of the first cache 102 and the second cache 112 for consecutive logical addresses so that a portion of the logical address space is assigned to the first cache 102 and a portion of the logical address space is assigned to the second cache 112. By logically striping read request data across the first cache 102 and the second cache 112 or otherwise distributing read caching operations between the first cache 102 and the second cache 112, in one embodiment, the first cache 102 and the second cache 112 form a single logical read cache for the storage device 118.

In one embodiment, the write acknowledgement module 706 acknowledges, to a requesting entity such as the host device 114, a write request that the write request module 602 receives. The write acknowledgement module 706, in a further embodiment, acknowledges

persistence of the write request. In one embodiment, the write acknowledgement module 706 implements a particular data integrity policy. Advantageously, embodiments of the present invention permit variations in the data integrity policy that is implemented.

The write acknowledgement module 706, in one embodiment, acknowledges the write request in response to the first cache write module 608 writing data of the write request to the first cache 102, regardless of whether or not the second cache write module 610 has written the data to the second cache 112. Acknowledging a write request regardless of whether the second cache write module 610 has written data of the write request to the second cache 112, in one embodiment, decreases a write time that a client, such as the host device 114, perceives balanced against the cost of some amount of reliability due to redundancy.

In a further embodiment, the write acknowledgement module 706 acknowledges the write request in response to both the first cache write module 608 writing the data to the first cache 102 and the second cache write module 610 writing the data to the second cache 112. Waiting to acknowledge a write request until both the first cache 102 and the second cache 112 store the data of the write request, in one embodiment, increases reliability and redundancy of the data at the possible cost of a slightly increased write time that a client, such as the host device 114, perceives. The data integrity policy, in one embodiment, is user defined. For example, a user may select a write acknowledgement setting for a data integrity policy, select a performance setting that prioritizes performance relative to data integrity, or the like.

In one embodiment, the cleaner module 708 writes data from the first cache 102 and/or the second cache 112 to the storage device 118. Data that is stored in the first cache 102 and/or the second cache 112 that is not yet stored in the storage device 118 is referred to as "dirty" data. Once the storage device 118 stores data, the data is referred to as "clean." The cleaner module 708 cleans data in the first cache 102 and the second cache 112 by writing the data to the storage device 118. The cleaner module 708, in one embodiment, writes data to the storage device 118 based on a write policy.

In one embodiment, the cleaner module 708 uses a write-back write policy, and does not immediately write data of a write request to the storage device 118 upon receiving the write request. Instead, the cleaner module 708, in one embodiment, performs an opportunistic or "lazy" write, writing data to the storage device 118 when the data is ejected from the caches 102, 112, when the caches 102, 112 or the direct cache module 116 has a light load, when available storage capacity in the first cache 102 and/or the second cache 112 falls below a threshold, or the like. In a write-back embodiment, the cleaner module 708 reads data from one of the first cache 102 and the second cache 112, writes the data to the storage device 118, and sets an indicator that

the storage device 118 stores the data in response to successfully writing the data to the storage device 118. Setting the indicator that the storage device 118 stores the data alerts the trim module 606 that the data may be cleared from one of the first cache 102 and the second cache 112.

In one embodiment, the cleaner module 708 sets an indicator that the storage device 118 stores data by marking the data as clean in at least one of the first cache 102 and the second cache 112. In a further embodiment, the cleaner module 708 may set an indicator that the storage device 118 stores data by communicating an address of the data to the trim module 606, sending a command to the trim module 606 to clear the data, updating an indicator in a logical to physical mapping, or the like.

In one embodiment, the cleaner module 708 maintains a data structure indicating which data in the first cache 102 and the second cache 112 is clean and which data is dirty. In another embodiment, the cleaner module 708 references indicators in a mapping of logical addresses to physical media addresses to determine which data in the first cache 102 and the second cache 112 is clean and which data is dirty. In a further embodiment, the cleaner module 708 determines that data is dirty by determining whether the data is stored in both the first cache 102 and the second cache 112. Because dirty data is mirrored in both the first cache 102 and the second cache 112 and clean data is cached in a single one of the first cache 102 and the second cache 112, in one embodiment, the cleaner module 708 can determine whether data is clean or dirty by determining whether the data resides in both the first cache 102 and/or the second cache 112, without tracking clean and dirty data. The cleaner module 708, in this embodiment, writes data to the storage device 118 in response to determining that both the first cache 102 and the second cache 112 store the data.

In another embodiment, instead of cleaning data according to a write-back write policy, the cleaner module 708 uses a write-through policy, performing a synchronous write to the storage device 118 for each write request. The cleaner module 708, 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 one of the first cache 102 and the second cache 112, or the like.

In one embodiment, the garbage collection module 710 recovers storage capacity of physical storage media corresponding to data that is marked as invalid, such as data cleared by the trim module 606. The garbage collection module 710, in one embodiment, is integrated with the cleaner module 708 and/or the trim module 606 and recovers storage capacity of the physical storage media substantially immediately in response to the trim module 606 clearing the data,

simultaneously with the trim module 606 clearing the data, or the like.

In a further embodiment, the garbage collection module 710 recovers storage capacity of physical storage media corresponding to invalid data opportunistically. The garbage collection module 710, in another embodiment, recovers data substantially independently of the timing of the cleaner module 708 and/or the trim module 606. For example, the garbage collection module 710 may recover storage capacity in response to a lack of available storage capacity, a percentage of data marked as invalid reaching a predefined threshold level, a consolidation of valid data, an error detection rate for a section of physical storage media reaching a threshold value, performance crossing a threshold value, a scheduled garbage collection cycle, identifying a section of physical storage media with a high amount of invalid data, identifying a section of physical storage media with a low amount of wear, or the like.

In one embodiment, the garbage collection module 710 relocates valid data in a section of physical storage media in the first cache 102 or the second cache 112 that the garbage collection module 710 is recovering. In one embodiment, the garbage collection module 710 is part of an autonomous garbage collector system that operates within the first cache 102 and/or the second cache 112. This allows each cache 102, 112 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 garbage collection module 710, upon recovering a section of physical storage media, allows the first cache 102 or the second cache 112 to re-use the section of physical storage media to store different data. In one embodiment, the garbage collection module 710 adds the recovered section of physical storage media to an available storage pool, or the like. The garbage collection module 710, in one embodiment, erases existing data in a recovered section. In a further embodiment, the garbage collection module 710 allows the first cache 102 or the second cache 112 to overwrite existing data in a recovered section. Whether or not the garbage collection module 710, in one embodiment, erases existing data in a recovered section may depend on the nature of the physical storage media. 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 garbage collection module 710 does not erase data in a recovered section, but allows the first cache 102 or the second cache 112 to overwrite data in the recovered section, the garbage collection module 710, in certain embodiments, may mark the data in the recovered section as unavailable to service read requests so that subsequent requests for data in the recovered section return a null result or an empty set of data until the first cache

102 or the second cache 112 overwrites the data.

In one embodiment, the garbage collection module 710 recovers storage capacity of the first cache 102 and/or the second cache 112 one or more storage divisions at a time. A storage division, in one embodiment, is an 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 trim module 606 may erase the data of a storage division instead of the garbage collection module 710.

In one embodiment, allowing the trim module 606 to mark data as invalid rather than actually erasing the data and allowing the garbage collection module 710 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 garbage collection module 710 to operate autonomously and opportunistically within the first cache 102 and the second cache 112 provides a way to separate erase operations from reads, writes, and other faster operations so that the caches 102, 112 operate very efficiently.

In one embodiment, the cache replace module 712 provides for the physical replacement of one of the first cache 102 and the second cache 112 due to device failure, a device upgrade, or the like. The cache replace module 712, in one embodiment, detects that one of the first cache 102 and the second cache 112 has been replaced with a replacement cache. The cache replace module 712 may detect a replacement cache automatically, in response to a user command, as part of a startup routine, or the like.

The cache replace module 712, in a further embodiment, mirrors dirty data from the remaining one of the first cache 102 and the second cache 112 to the detected replacement cache in response to detecting the replacement cache. In another embodiment, the cleaner module 708 writes dirty data from the remaining one of the first cache 102 and the second cache 112 to the storage device 118 in response to the cache replace module 712 detecting the replacement cache. Once the cache replace module 712 detects the replacement cache, in one embodiment, the replacement cache becomes either the first cache 102 or the second cache 112, and is treated as such by the direct cache module 116. In one embodiment, the replacement cache shares the logical address space with the remaining one of the first cache 102 and the second cache 112, etc.

In one embodiment, the storage device interface module 714 provides an interface between the direct cache module 116 and the first cache 102, the second cache 112, and/or the storage device 118. As described above with regard to Figure 2, in various embodiments, the direct cache module 116 may interact with the first cache 102, the second cache 112, and/or the storage device 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 storage device interface module 714 provides the direct cache module 116 with access to one or more of these interfaces. For example, the storage device interface module 714 may receive read commands, write commands, and clear (or TRIM) commands from one or more of the first cache write module 608, the second cache write module 610, the trim module 606, the read request module 704, the cleaner module 708, the garbage collection module 710, the cache replace module 712, and the like and relay the commands to the first cache 102, the second cache 112, and/or the storage device 118. In a further embodiment, the storage device interface module 714 may translate or format a command into a format compatible with an interface for the first cache 102, the second cache 112, and/or the storage device 118.

In one embodiment, the storage device interface module 714 has exclusive ownership over the storage device 118 and the direct cache module 116 is an exclusive gateway to accessing the storage device 118. Providing the storage device interface module 714 with exclusive ownership over the storage device 118 and preventing access to the storage device 118 by other routes obviates stale data issues and cache coherency requirements, because all changes to data in the storage device 114 are processed by the direct cache module 116.

In a further embodiment, the storage device interface module 714 does not have exclusive ownership of the storage device 118, and the storage device interface module 714 manages cache coherency for the first cache 102 and the second cache 112. For example, in various embodiments, the storage device interface module 714 may access a common directory with other users of the storage device 118 to maintain coherency, may monitor write operations from other users of the storage device 118, participate in a predefined coherency protocol with other users of the storage device 118, or the like.

Figure 6A depicts one embodiment 800 of the first cache 102 and the second cache 112. The depicted embodiment 800 illustrates a portion of a logical address space shared by both the first cache 102 and the second cache 112. As used herein a logical address space shared by two caches means that a given logical address may be mapped to a physical location in both caches. This means that data for a given logical address can exist in one cache or the other cache, or both caches.

The depicted embodiment 800 includes logical addresses 804a-e for the first cache 102 and the second cache 112 and data 810 associated with the logical addresses 804a-e. In the depicted embodiment 800, the logical addresses 804a-e each mirror data 810 between the first cache 102 and the second cache 112.

In the depicted embodiment 800, the first cache 102 and the second cache 112 share a logical address space that includes the set of logical addresses 804a-e. In one embodiment, physical storage capacity of the first cache 102 and the second cache 112 is allocated to a logical address 804 only when the logical address 804 stores data 810, and physical storage capacity of the first cache 102 and the second cache 112 is deallocated when an associated logical address 804 is deallocated, cleared, trimmed, or the like. In a further embodiment, physical storage capacity of the first cache 102 and the second cache 112 is allocated to logical addresses 804 regardless of whether the logical addresses 804 store data 810.

In the depicted embodiment 800, the first cache 102 and the second cache 112 store data 810 from write requests corresponding to the logical addresses 804a-e. The data 810, in the depicted embodiment 800, is mirrored between the first cache 102 and the second cache 112. In one embodiment, the first cache 102 and the second cache 112 store the data 810 corresponding to logical addresses 804a-e redundantly because the storage device 118 does not yet store the data 810 and/or because the trim module 606 has not yet cleared the data 810 from the first cache 102 and/or the second cache 112.

In the depicted embodiment 800, because each of the logical addresses 804a-e stores data 810 that is mirrored between the first cache 102 and the second cache 112, each of the logical addresses 804a-e corresponds to physical storage media 110 of the first cache 102 and the second cache 112 and use storage capacity of both the first cache 102 and the second cache 112. A client, such as the host device 114, in the depicted embodiment 800, can access the data 810 from either the first cache 102 or the second cache 112. If an error occurs in data 810 corresponding to logical addresses 804a-e in either the first cache 102 or the second cache 112, a client, the direct cache module 116, or the like, can retrieve a redundant copy of the data 810 from the other cache 102, 112.

Figure 6B depicts another embodiment 820 of the first cache 102 and the second cache 112. In the depicted embodiment 820, the storage device 118 stores the data 810 associated with logical addresses 804a-d (e.g. all entries except the last logical address "2187" 804e) and the trim module 606 has cleared the data 810 associated with logical addresses 804a-d from one of the first cache 102 and the second cache 112. Data 810 that the trim module 606 has cleared from a cache 102, 112, in the depicted embodiment 820, is illustrated with diagonal hashing. In the

depicted embodiment 820, the data 810 corresponding to logical addresses 804a-d is logically striped between the first cache 102 and the second cache 112 and the data 810 corresponding to logical address 804e is mirrored between the first cache 102 and the second cache 112 (the caches 102, 112 still hold the most current version of the data).

In the depicted embodiment 820, the cache selection module 702 selects odd logical addresses 804 for the trim module 606 to clear data 810 from the first cache 102 and selects even logical addresses 804 for the trim module 606 to clear data 810 from the second cache 112. Once the trim module 606 has cleared data 810 corresponding to one or more odd logical addresses 804 from the first cache 102 and has cleared data 810 corresponding to one or more even logical addresses 804 from the second cache 112, at least a portion of the remaining data 810 is logically striped between the first cache 102 and the second cache 112, and data 810 corresponding to consecutive logical addresses 804 is stored by different caches 102, 112. It should be noted that while the shared address space is logically striped in the depicted embodiment, those of skill in the art recognize that the media of the caches 102, 112 may not be logically striped.

Clearing the data 810 corresponding to logical addresses 804a-d from either the first cache 102 or the second cache 112, in one embodiment, frees storage capacity of the first cache 102 and the second cache 112, as the first cache 102 and the second cache 112 recover physical storage media 110 corresponding to the cleared data 810. In certain embodiments, the caches 102, 112 manage the physical layout of the data on the media and may use a log-based, append protocol that permits for cleared physical media to be reused. Because the trim module 606 clears the data 810 according to a deterministic protocol, the read request module 704, in one embodiment, directs read requests from a client, such as the host device 114, to either the first cache 102 or the second cache 112 based on the deterministic protocol, without first determining whether the first cache 102 or the second cache 112 store data 810 of the read request.

Figure 6C depicts one embodiment 830 of the first cache 102, the second cache 112, and the storage device 118. The embodiment 830 depicts a snapshot of a physical layout of data in the first cache 102 and the second cache 112 at a specific moment in time. In the depicted embodiment 830, the first cache 102, the second cache 112, and the storage device 118 are each illustrated with associated data and state information for the data. The state information for the data, including a physical address 802, a logical address 804, a clean/dirty indicator 806, a valid/invalid indicator 808, and a storage device address 812 are depicted for illustration purposes, and may or may not be stored on the first cache 102, the second cache 112, and/or the storage device 118. In various embodiments, the physical addresses 802, the logical addresses

804, the clean/dirty indicators 806, the valid/invalid indicators 808, and the storage device addresses 812 may be stored as metadata by the direct cache module 116 (i.e. in a mapping data structure), determined by the direct cache module 116, or not used by the direct cache module 116.

The depicted embodiment 830 illustrates five example physical addresses 802a-e for the first cache 102 and five example physical addresses 802f-j for the second cache 112. In the depicted embodiment 830, the logical addresses 804 of the first cache 102 and the second cache 112 are part of a shared logical address space used by both the first cache 102 and the second cache 112. The logical addresses 804 of the first cache 102 and the second cache 112, in the depicted embodiment 830, also correspond to the storage device addresses 812 of the storage device 118. The storage device addresses 812 of the storage device 118 may be logical addresses, physical addresses, or the like.

In the depicted embodiment 830, physical address "0" 802a and physical address "4" 802e of the first cache 102 store data 810 that is dirty and valid. The data 810 stored in physical address "0" 802a and physical address "4" 802e of the first cache 102 is mirrored in the second cache 112 at physical address "3" 802i and physical address "2" 802h. While the mirrored data 810 stored in both the first cache 102 and the second cache 112 has different physical addresses 802, the logical addresses 804 of mirrored data 810 are identical in the depicted embodiment 830. Because the data stored in the first cache 102 at physical address "0" 802a and physical address "4" 802e and in the second cache 112 at physical address "3" 802i and physical address "2" 802h is dirty and has not yet been stored in the storage device 118 by the cleaner module 702, the data 810 in corresponding storage device address "2183" 812b and storage device address "5613" 812e is different than the data 810 in the first and second caches 102, 112. The data 810 at storage device address "2183" 812b is null, or empty, and the data 810 at storage device address "5613" 812e is different than the data in the first cache 102 and the second cache 112.

The remaining data 810 in the first cache 102 and the second cache 112, in the depicted embodiment 830, is clean because the storage device 118 stores the remaining data 810. In the depicted embodiment 830, the cache selection module 702 has assigned the first cache 102 to store data 810 for even logical addresses 804 and assigned the second cache 112 to store data 810 for odd logical addresses 804. Because, in the depicted embodiment 830, the cache selection module 702 uses an even/odd deterministic protocol, the trim module 605 clears data 810 corresponding to odd logical addresses 804 from the first cache 102 and clears data 810 corresponding to even logical addresses 804 from the second cache 112.

Physical address "1" 802b of the first cache 102 stores data 810 that is clean and valid, and that is not mirrored to the second cache 112. The data 810 at physical address "1" 802b of the first cache 102 may be cached read data, written to the first cache 102 by the read retrieve module 720, or may be cached write data that the cleaner module 708 has previously written to the storage device 118 and that the trim module 606 has already cleared from the second cache 112. The data 810 at physical address "1" 802b of the first cache 102 matches corresponding data 810 stored at storage device address "3902" 812c because the data 810 is clean. The logical address 804 of "3902" corresponding to physical address "1" 802b of the first cache 102 is even.

In the depicted embodiment 830, the logical address 804 of "4327" associated with the data 810 of physical address "3" 802d of the first cache 102 is odd. Because of the odd logical address 804, the trim module 606, in the depicted embodiment 830, marked the data 810 stored at physical address "3" 802d of the first cache 102 as invalid. Although invalid, the physical address "3" 802d of the first cache 102 still stores the data 810 because the cleaner module 708 has not yet recovered the physical storage media of physical address "3" 802d in the depicted embodiment 830. The logical address 804 of "4327" associated with the data 810 of physical address "0" 802f of the second cache 102 is both clean and valid and available to service any read requests for logical address "4327."

Similarly, the data 810 stored at physical address "1" 802g of the second cache 112 is clean and invalid because the trim module 606 cleared or trimmed the data 810 from physical address "1" 802g of the second cache 112 because the storage device 118 stores corresponding data 810 at storage device address "6298" 812f, and the logical address 804 of "6298" that corresponds to physical address "1" 802g is even. The data 810 that has been marked invalid at physical address "1" 802g of the second cache 112 is cached in physical address "2" 802c of the first cache 102 to service read requests. Physical address "0" 802f and physical address "4" 802j of the second cache 112 store clean and valid data 810 with odd logical addresses 804 that the storage device 118 also stores at storage device address "4327" 812d and storage device address "1069" 812a to service read requests. As described above, data 810 of physical address "0" 802f of the second cache 112 is still mirrored in physical address "3" 802d of the first cache 102 because the cleaner module 708 has not yet recovered the physical storage media of physical address "3" 802d.

In one embodiment, the cleaner module 708 does not track the clean/dirty indicator 806, but determines the clean or dirty state of data 810 based on whether the data 810 is stored in both the first cache 102 and the second cache 112 or just in one of the caches 102, 112. If both the first cache 102 and the second cache 112 store data 810 corresponding to a logical address 804,

in one embodiment, the data 810 is known to be dirty. If only one of the first cache 102 and the second cache 112 stores data 810 corresponding to a logical address 804, in one embodiment, the data 810 is known to be clean, in part due to the way redundancy and capacity are balanced between the first cache 102 and the second cache 112.

The cleaner module 708, in one embodiment, may query both the first cache 102 and the second cache 112 with a logical address 804 to determine whether data 810 corresponding to the logical address 804 is clean or dirty (i.e. whether or not both caches 102, 112 store the data 810). In a further embodiment, the cleaner module 708 may query a single cache 102, 112 based on a deterministic protocol used by the deterministic protocol module 714, and if the selected cache 102, 112 does not store the data 810 (or the data 810 is invalid), then the data 810 is clean. If the selected cache 102, 112 does store the data 810 (and the data is valid), in one embodiment, the data 810 is dirty because the trim module 606 has not yet cleared or trimmed the data 810 from the selected cache 102, 112 in response to the storage device 118 storing the data. In a further embodiment, the cleaner module 708 may refer to a mapping data structure, such as a logical to physical mapping, a mapping maintained by the cache selection module 702, or the like to determine whether data 810 corresponding to a logical address 804 is clean or dirty.

FLOW CHARTS

Figure 7 depicts one embodiment of a method 1100 for redundant write caching. The method 1100 begins, and the write request module 602 detects 1102 (or receives) a write request from a requesting entity, such as the host device 114, to store data on the storage device 118. If the write request module 602 does not detect 1102 a write request, in the depicted embodiment, the write request module 602 continues to monitor for subsequent write requests.

In response to the write request module 602 detecting 1102 a write request, the cache write module 604 writes 1104 the data of the write request to a first cache 102 and the cache write module 604 mirrors 1106 the data to the second cache 112. The write acknowledgement module 706, in the depicted embodiment, acknowledges 1108 persistence of the write request 1108 to the requesting entity, such as the host device 114.

The trim module 606 determines 1110 whether or not the storage device 118 stores the data of the write request. The trim module 606, in various embodiments, may determine 1110 whether the storage device 118 stores the data based on an indicator from the cleaner module 708, by scanning the storage device 118 for the data, by querying the storage device 118 for the data, by checking a status of an indicator in a clean/dirty data structure that the cleaner module 706 maintains, or the like. In response to an indicator that the storage device 118 stores the data of the write request, the trim module 606 clears 1112 the data of the write request from one of

the first cache 102 and the second cache 112 based on a selection by the cache selection module 702 and the method 1100 ends. If the trim module 606 determines 1110 that the storage device 118 does not store the data of the write request, the trim module 606 continues to check to determine 1110 whether or not the storage device 118 stores the data. Subsequent to each determination 1110 the trim module 606 may pause for a period of time.

Figure 8 depicts one embodiment of a method 1200 for read caching. The method 1200 begins, and the read request module 704 detects 1202 (or receives) a read request from a requesting entity, such as the host device 114, to retrieve requested data from the storage device 118. If the read request module 704 does not detect 1202 a read request, in the depicted embodiment, the read request module 704 continues to detect 1202 subsequent read requests.

In response to the read request module 704 detecting 1202 a read request, the read miss module 704 determines 1204 whether one or more of the first cache 102 and the second cache 112 store the data of the read request. The read miss module 704 may direct the read request to both caches 102, 112, to one of the caches 102, 104 based on a deterministic protocol, may query one or both of the caches 102, 112, or the like to determine 1204 whether one or more of the caches 102, 112 store the data of the read request.

If the read miss module 718 determines 1204 that one of the caches 102, 112 stores the data of the read request, the read request module 704 directs 1206 the read request to the determined cache 102, 112, and the read request module 704 reads 1204 the data of the read request from the determined cache 102, 112. The read request module 704 (or the determined cache 102, 112 itself) returns 1216 the data of the read request to the requesting entity and the method 1200 ends.

If the read miss module 718 determines 1204 that neither of the caches 102, 112 store the data of the read request (i.e. that there is a cache miss), the read request module 704 reads 1210 the data of the read request from the storage device 118. The cache selection module 702 selects 1212 one of the first cache 102 and the second cache 112, using the deterministic protocol module 714, the cache attribute module 716, or the like. The read request module 704 writes 1214 the data of the read request to the selected cache 102, 112. The read request module 704 returns 1216 the data of the read request to the requesting entity and the method 1000 ends.

Figure 9 depicts one embodiment of a method 1300 for cleaning a cache 102, 112. The cleaner module 708 determines 1302 whether data in a cache 102, 112 is dirty and not yet stored in the storage device 118. The data, in various embodiments, may correspond to a specific logical address, to a write request, or the like. The cleaner module 708, in one embodiment, determines 1302 that the data is dirty based on a clean/dirty indicator 806 corresponding to the

data. In a further embodiment, the cleaner module 708 determines 1302 that the data is dirty based on whether both the first cache 102 and the second cache 112 store the data. If the cleaner module 708 determines 1302 that data is clean and not dirty, the cleaner module 708, in the depicted embodiment, continues to determine 1302 whether different data is dirty, such as data corresponding to a different logical address, to a different write request, or the like.

If the cleaner module 708 determines 1302 that the data is dirty, the cleaner module 708 writes 1304 the data to the storage device 118. The cleaner module 708 sets 1306 an indicator that the storage device 118 stores the data. The cleaner module 708 may set 1306 the indicator by marking the data as clean in one or both of the caches 102, 112, by communicating that the data is clean to the trim module 606, or the like. The cache selection module 702 selects 1308 one of the first cache 102 and the second cache 112 using the deterministic protocol module 714, the cache attribute module 716, or the like. The trim module 606 marks 1310 the data as invalid in the selected cache 102, 112, or otherwise clears or trims the data from the selected cache 102, 112. The garbage collection module 710 recovers 1312 storage capacity of the physical storage media corresponding to the data in the selected 1308 cache 102, 112 and the method 1300 ends.

[0161] 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 write caching, the method comprising:

detecting a write request to store data on a storage device;

writing data of the write request to a first cache;

mirroring the data to a second cache; and

clearing the data from one of the first cache and the second cache in response to an indicator that the storage device stores the data, such that the data remains available in the other of the first cache and the second cache to service read requests.

2. The method of Claim 1, further comprising selecting one of the first cache and the second cache according to a deterministic protocol, wherein the data is cleared in the selected one of the first cache and the second cache.

3. The method of Claim 2, wherein the first cache and the second cache share a logical address space that is directly mapped to distinct physical addresses on the first cache and the second cache.

4. The method of Claim 3, wherein the deterministic protocol is based on a logical address associated with the write request within the logical address space and further wherein the deterministic protocol logically stripes data between the first cache and the second cache by alternating selection of the first cache and the second cache for consecutive logical addresses such that a portion of the logical address space is assigned to the first cache and a portion of the logical address space is assigned to the second cache.

5. The method of Claim 2, further comprising directing, based on the deterministic protocol, a read request for the data to the other of the first cache and the second cache that is not selected.

6. The method of Claim 2, further comprising:

detecting a read request to retrieve requested data from the storage device;

determining that the requested data is not stored in either of the first cache and the second cache; and

writing the requested data to one of the first cache and the second cache according to the deterministic protocol in response to detecting that the requested data is not stored in either of the first cache and the second cache.

7. The method of Claim 1, further comprising acknowledging the write request in response to writing the data to the first cache.

8. The method of Claim 1, further comprising selecting one of the first cache and the second cache based on one or more cache attributes, wherein the data is cleared in the selected one of the first cache and the second cache.

9. The method of Claim 1, further comprising:

reading the data from one of the first cache and the second cache;

writing the data to the storage device; and

setting the indicator that the storage device stores the data in response to successfully writing the data to the storage device.

10. The method of Claim 1, wherein clearing the data in one of the first cache and the second cache is in response to a predefined event occurring after setting the indicator that the storage device stores the data.

11. The method of Claim 1, further comprising:

detecting a replacement of one of the first cache and the second cache with a replacement cache; and

mirroring dirty data from a remaining one of the first cache and the second cache to the replacement cache, the dirty data comprising data that is not stored on the storage device.

12. An apparatus for write caching, the apparatus comprising:

a write request module that detects a write request to store data on a storage device;

a first cache write module that writes data of the write request to a first non-volatile solid-state storage cache;

a second cache write module that writes the data to a second non-volatile solid-state storage cache; and

a trim module that trims the data from one of the first non-volatile solid-state storage cache and the second non-volatile solid-state storage cache in response to an indicator that the storage device stores the data, such that the data remains available in the other of the first non-volatile solid-state storage cache and the second non-volatile solid-state storage cache to service read requests.

13. The apparatus of Claim 12, further comprising:

a cache selection module that selects the first non-volatile solid-state storage cache, wherein the trim module trims the data from the first non-volatile solid-state storage cache in response to the cache selection module

selecting the first non-volatile solid-state storage cache; and

a read request module that directs a subsequent read request for the data to the second non-volatile solid-state storage cache.

14. The apparatus of Claim 12, further comprising a write acknowledgement module that acknowledges the write request in response to both the first cache write module writing the data to the first non-volatile solid-state storage cache and the second cache write module writing the data to the second non-volatile solid-state storage cache.

15. The apparatus of Claim 12, wherein the storage device, the first cache, and the second cache each comprise a block storage interface.

16. The apparatus of Claim 12, wherein the storage device and at least one of the first non-volatile solid-state storage cache and the second non-volatile solid-state storage cache comprise a direct attached storage ("DAS") device associated with a host device.

17. The apparatus of Claim 16, wherein the first non-volatile solid-state storage cache comprises a DAS device associated with the host device and the second non-volatile solid-state storage cache comprises an external device that is not directly associated with the host device.

18. A computer program product comprising a computer readable storage medium storing computer usable program code executed to perform operations for write caching, the operations of the computer program product comprising:

receiving a write request to store data on a storage device, the write request comprising data not stored on the storage device;

writing data of the write request to a first non-volatile solid-state storage cache;

writing the data of the write request to a second non-volatile solid-state storage cache;

selecting one of the first non-volatile solid-state storage cache and the second non-volatile solid-state storage cache according to a deterministic protocol; and

trimming the data on the selected one of the first non-volatile solid-state storage cache and the second non-volatile solid-state storage cache in response to an indicator that the storage device stores the data, such that the data of the write request remains available in the unselected one of the first non-volatile solid-state storage cache and the second non-volatile solid-state storage cache to service read requests.

19. The computer program product of Claim 18, further comprising:

determining that both the first non-volatile solid-state storage cache and the second non-volatile solid-state storage cache store the data;

writing the data to the storage device in response to determining that both the first non-volatile solid-state storage cache and the second non-volatile solid-state storage cache store the data; and

setting the indicator that the storage device stores the data in response to writing the data to the storage device.

20. The computer program product of Claim 18, wherein trimming the data comprises sending a TRIM command to the selected one of the first non-volatile solid-state storage cache and the second non-volatile solid-state storage cache, the TRIM command comprising an address for the data, wherein the first non-volatile solid-state storage cache is selected in response to a value for the address for the data being odd and the second non-volatile solid-state storage cache is selected in response to a value for the address for the data being even.

21. The computer program product of Claim 18, wherein trimming the data comprises marking the data as invalid on the selected one of the first non-volatile solid-state storage cache and the second non-volatile solid-state storage cache, wherein the selected one of the first non-volatile solid-state storage cache and the second non-volatile solid-state storage cache recovers storage capacity of physical storage media corresponding to the data in response to marking the data as invalid.

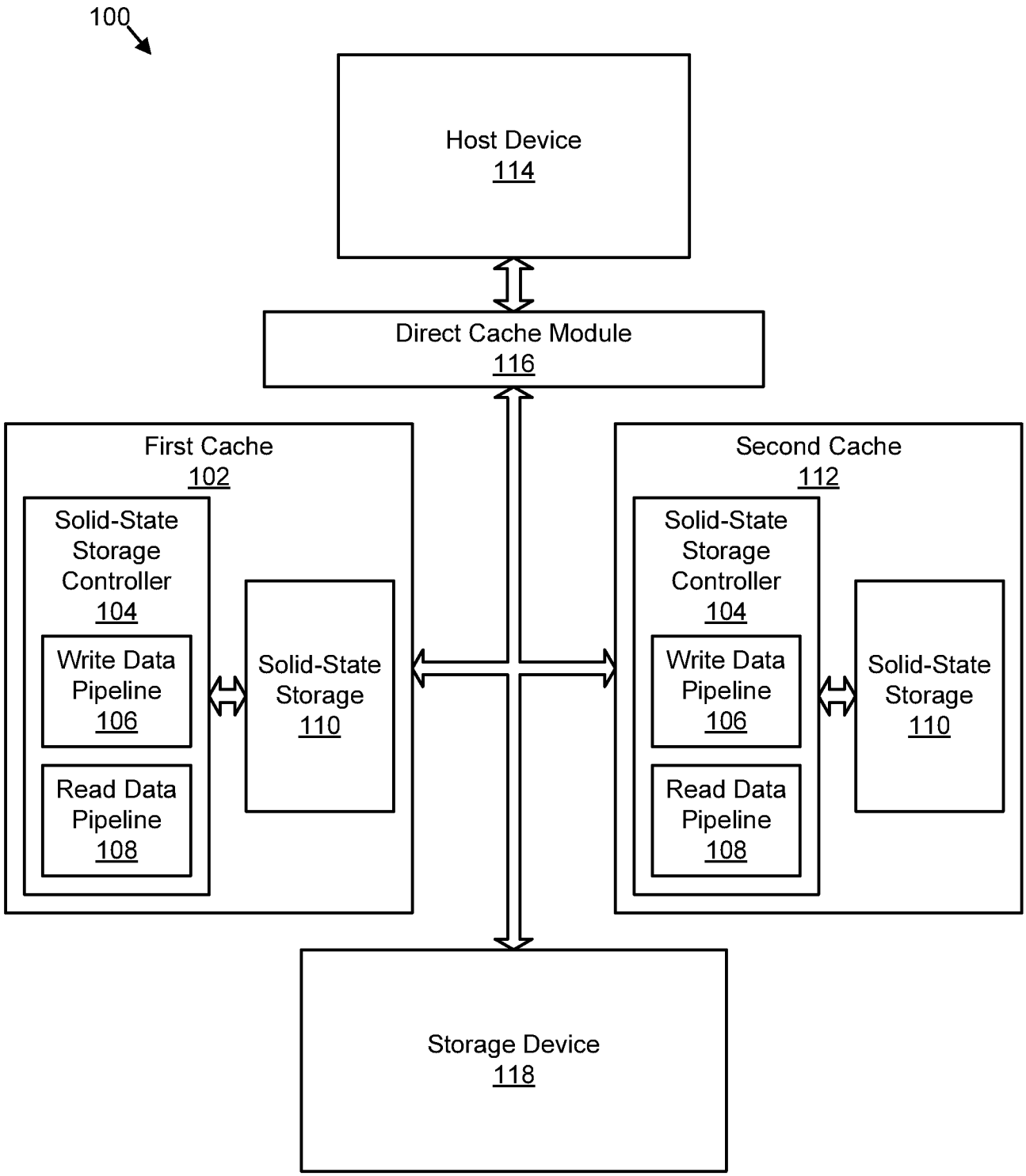
100
Host Device
114
Direct Cache Module
116
First Cache
102
Solid-State
Storage
Controller
104
Write Data
Pipeline
106
Read Data
Pipeline
108
Solid-State
Storage
110
Second Cache
112
Solid-State
Storage
Controller
104
Write Data
Pipeline
106
Read Data
Pipeline
108
Solid-State
Storage
110
Storage Device
118

FIG. 1

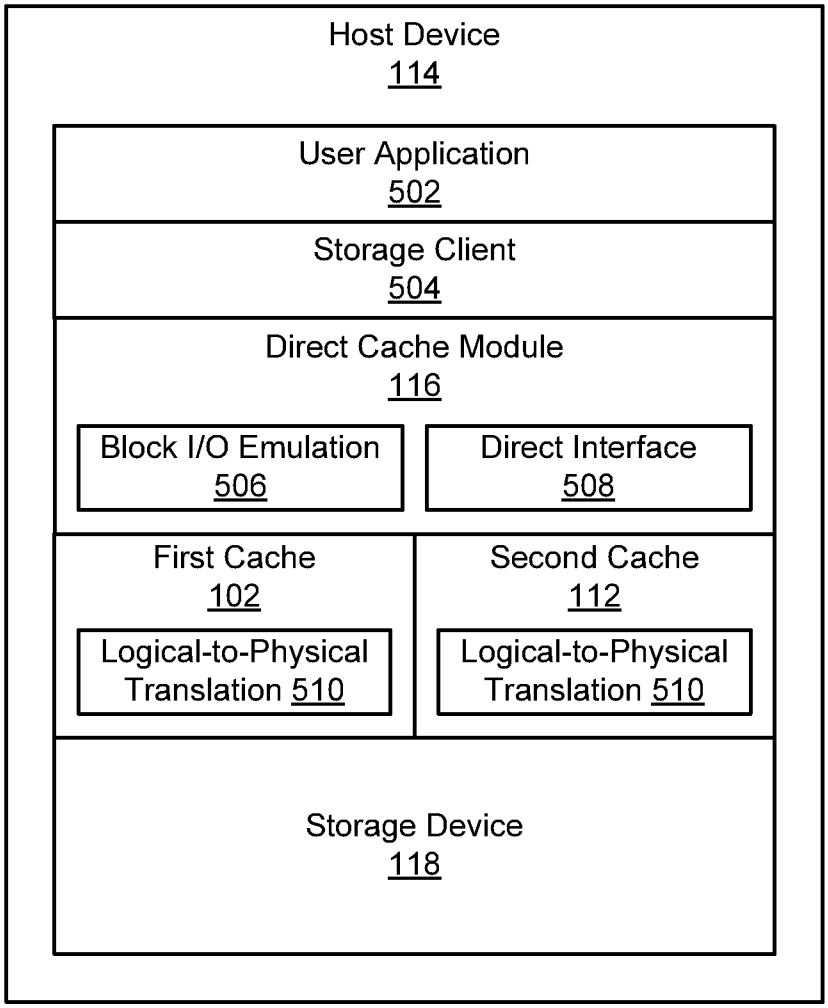
Host Device
114
User Application
502
Storage Client
504
Direct Cache Module
116
Block I/O Emulation
506
Direct Interface
508
First Cache
102
Logical-to-Physical
Translation 510
Second Cache
112
Logical-to-Physical
Translation 510
Storage Device
118

FIG. 2

PAGE 3/10

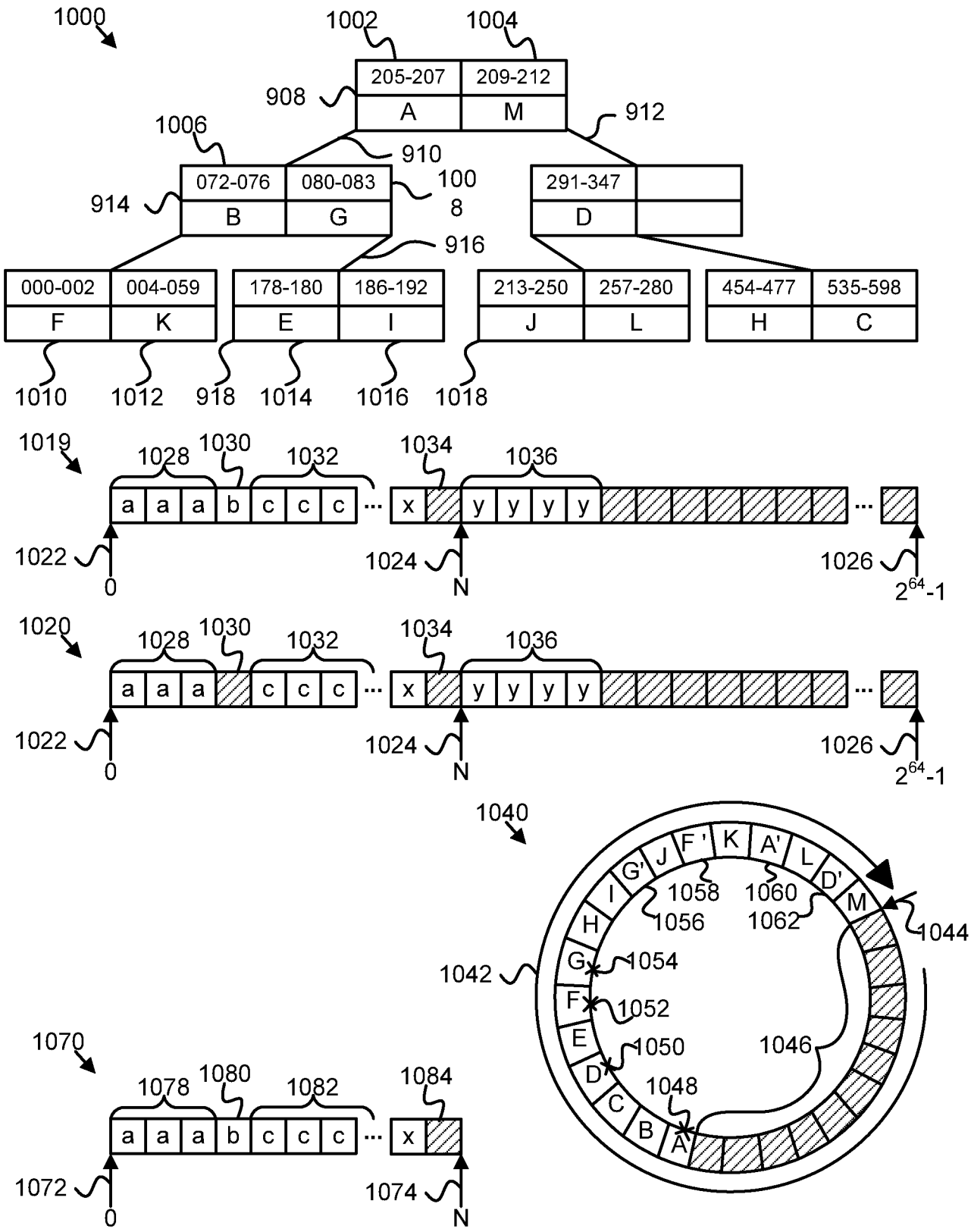

FIG. 3

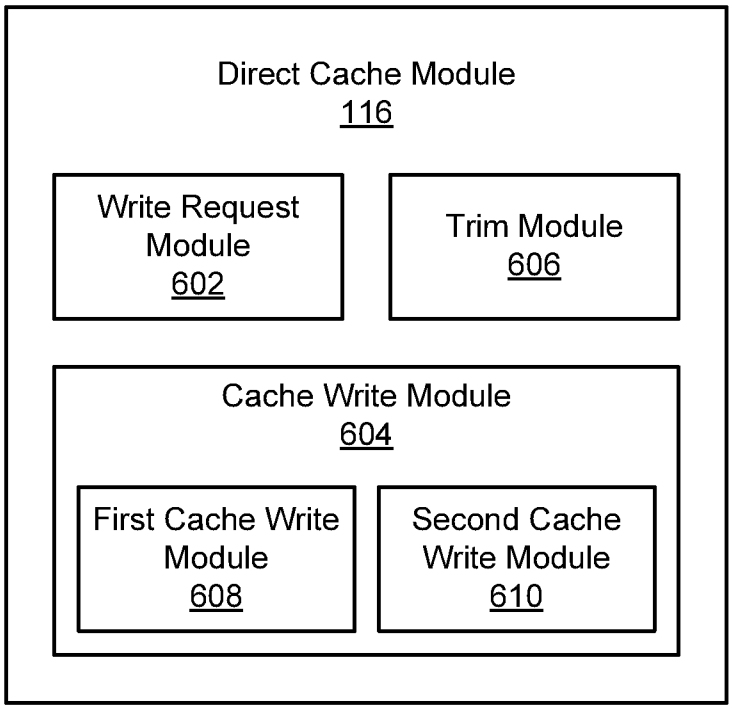
Direct Cache Module
116
Write Request Module
602
Trim Module
606
Cache Write Module
604
First Cache Write Module
608
Second Cache Write Module
610

FIG. 4

PAGE 5/10

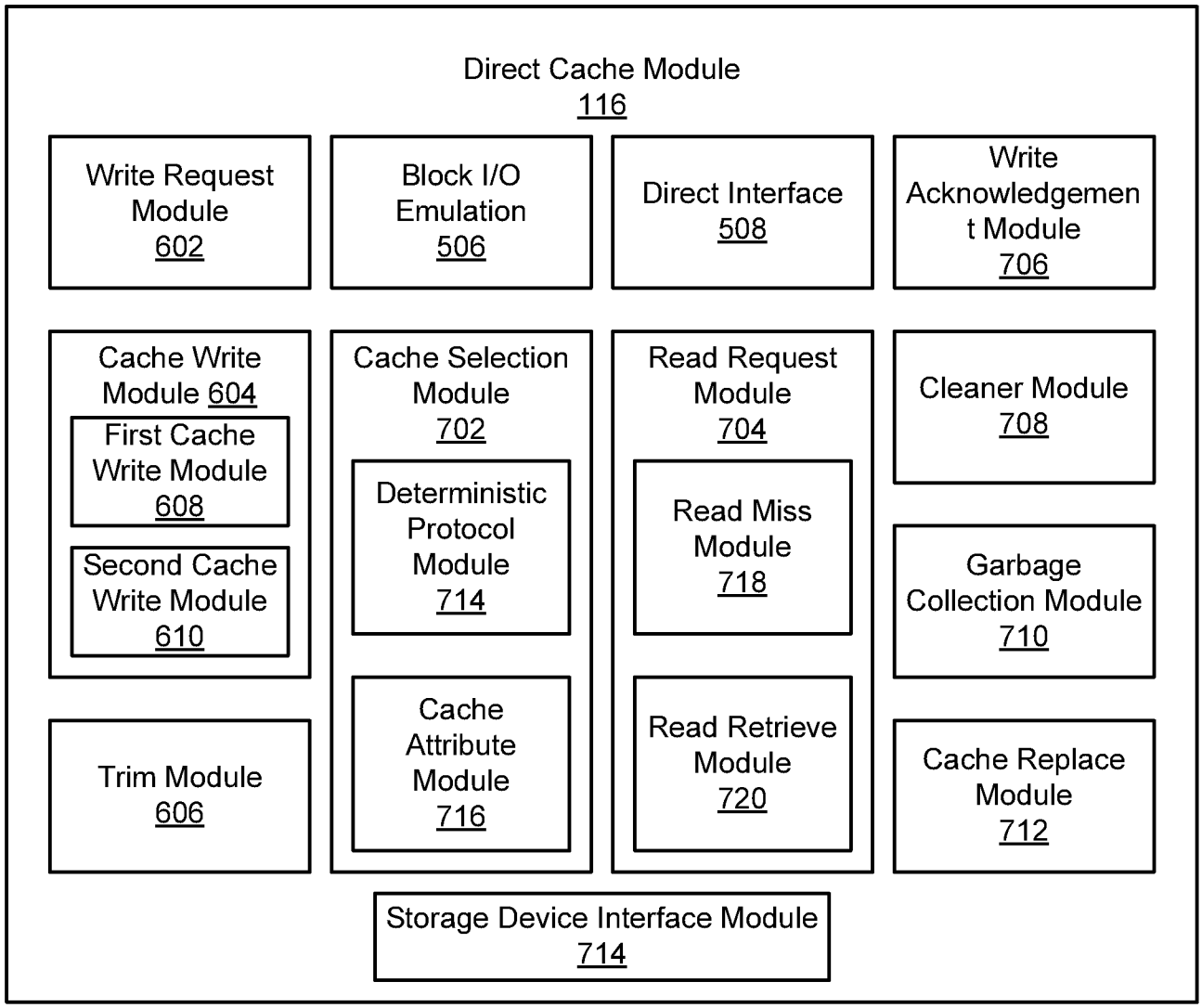

FIG. 5

PAGE 6/10

800

First Cache 102	
Log. Add. 804	Data 810
. . .	
2183 804a	1011 0001
2184 804b	0110 1010
2185 804c	1011 0011
2186 804d	1100 0110
2187 804e	0101 0100
. . .	

Second Cache 112	
Log. Add. 804	Data 810
. . .	
2183 804a	1011 0001
2184 804b	0110 1010
2185 804c	1011 0011
2186 804d	1100 0110
2187 804e	0101 0100
. . .	

FIG. 6A

820

First Cache 102	
Log. Add. 804	Data 810
. . .	
2183 804a	
2184 804b	0110 1010
2185 804c	
2186 804d	1100 0110
2187 804e	0101 0100
. . .	

Second Cache 112	
Log. Add. 804	Data 810
. . .	
2183 804a	1011 0001
2184 804b	
2185 804c	1011 0011
2186 804d	
2187 804e	0101 0100
. . .	

FIG. 6B

PAGE 7/10

830

First Cache
102

Phys. Add. 802	Log. Add. 804	Clean/ Dirty 806	Valid/ Invalid 808	Data 810
0 802a	2183	Dirty	Valid	1011 0001
1 802b	3902	Clean	Valid	0100 1110
2 802c	6298	Clean	Valid	1001 0010
3 802d	4327	Clean	Invalid	0110 0011
4 802e	5613	Dirty	Valid	0101 1011

Second Cache
112

Phys. Add. 802	Log. Add. 804	Clean/ Dirty 806	Valid/ Invalid 808	Data 810
0 802f	4327	Clean	Valid	0110 0011
1 802g	6298	Clean	Invalid	1001 0010
2 802h	5613	Dirty	Valid	0101 1011
3 802i	2183	Dirty	Valid	1011 0001
4 802j	1069	Clean	Valid	1100 1010

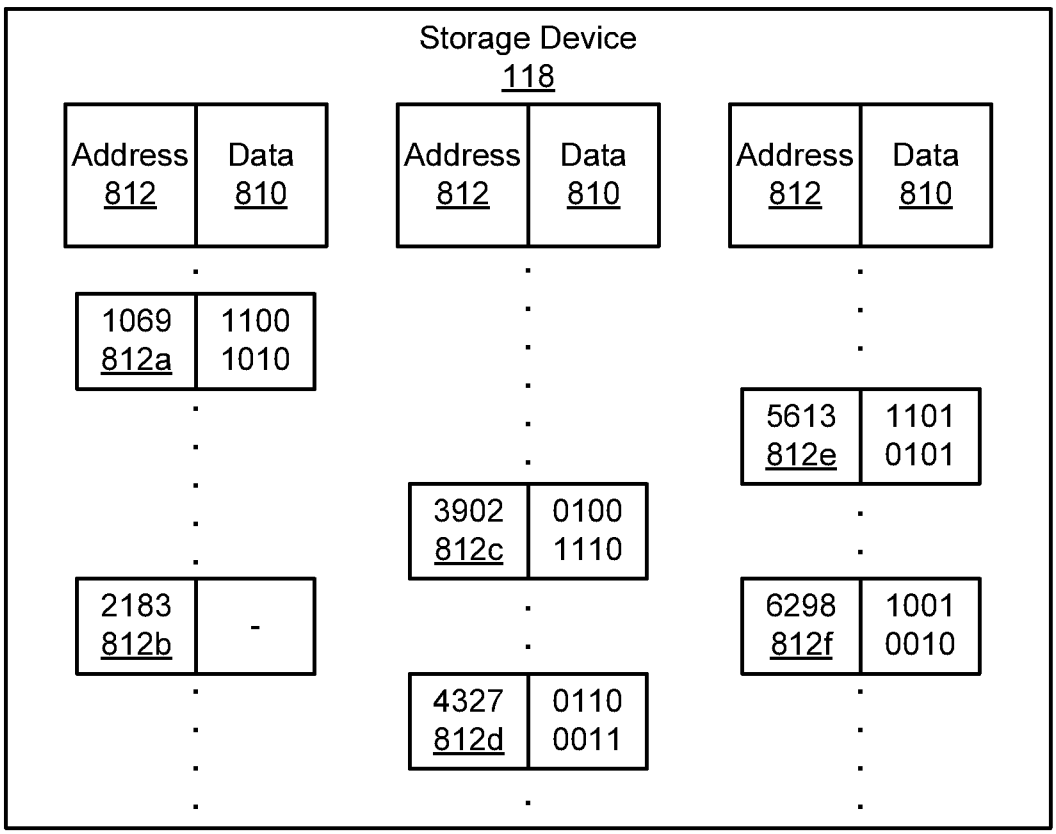

FIG. 6C

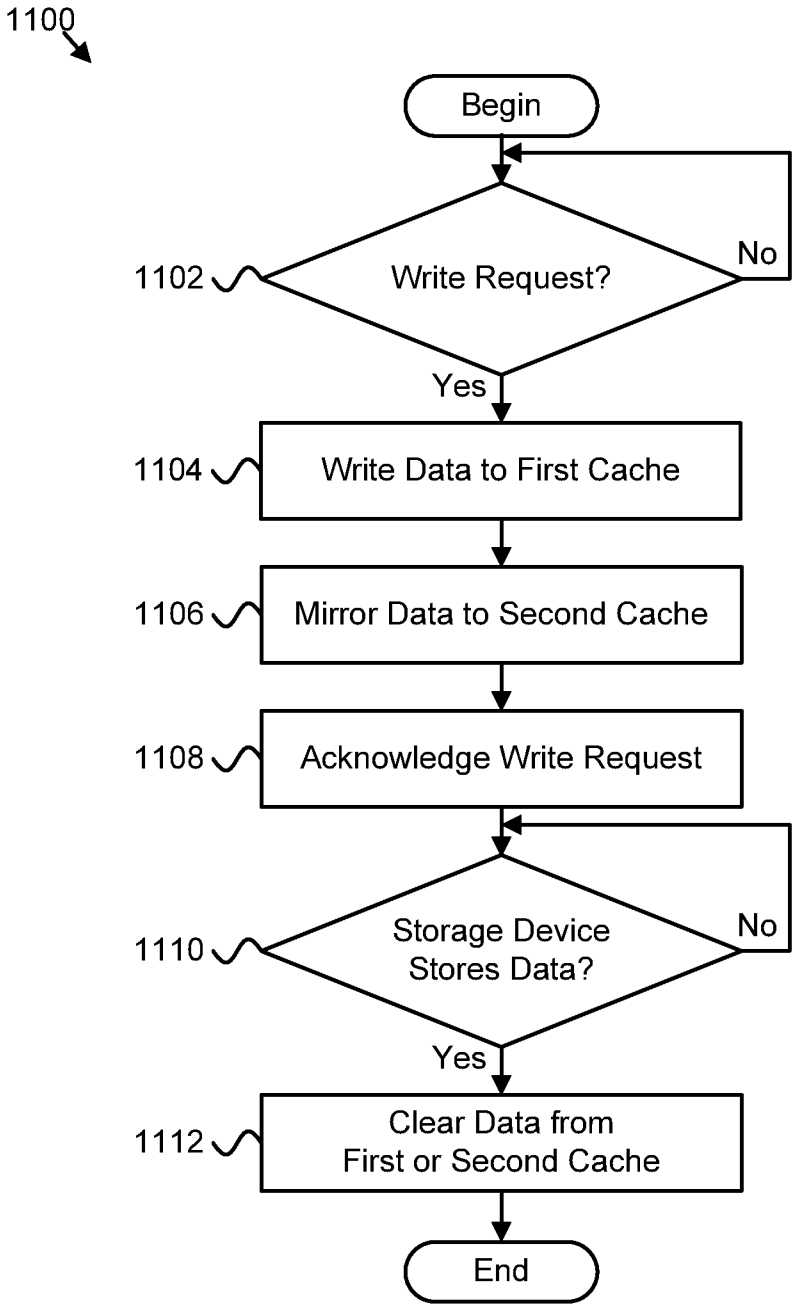
1100
Begin
1102
Write Request?
No
Yes
1104
Write Data to First Cache
1106
Mirror Data to Second Cache
1108
Acknowledge Write Request
1110
Storage Device Stores Data?
No
Yes
1112
Clear Data from First or Second Cache
End

FIG. 7

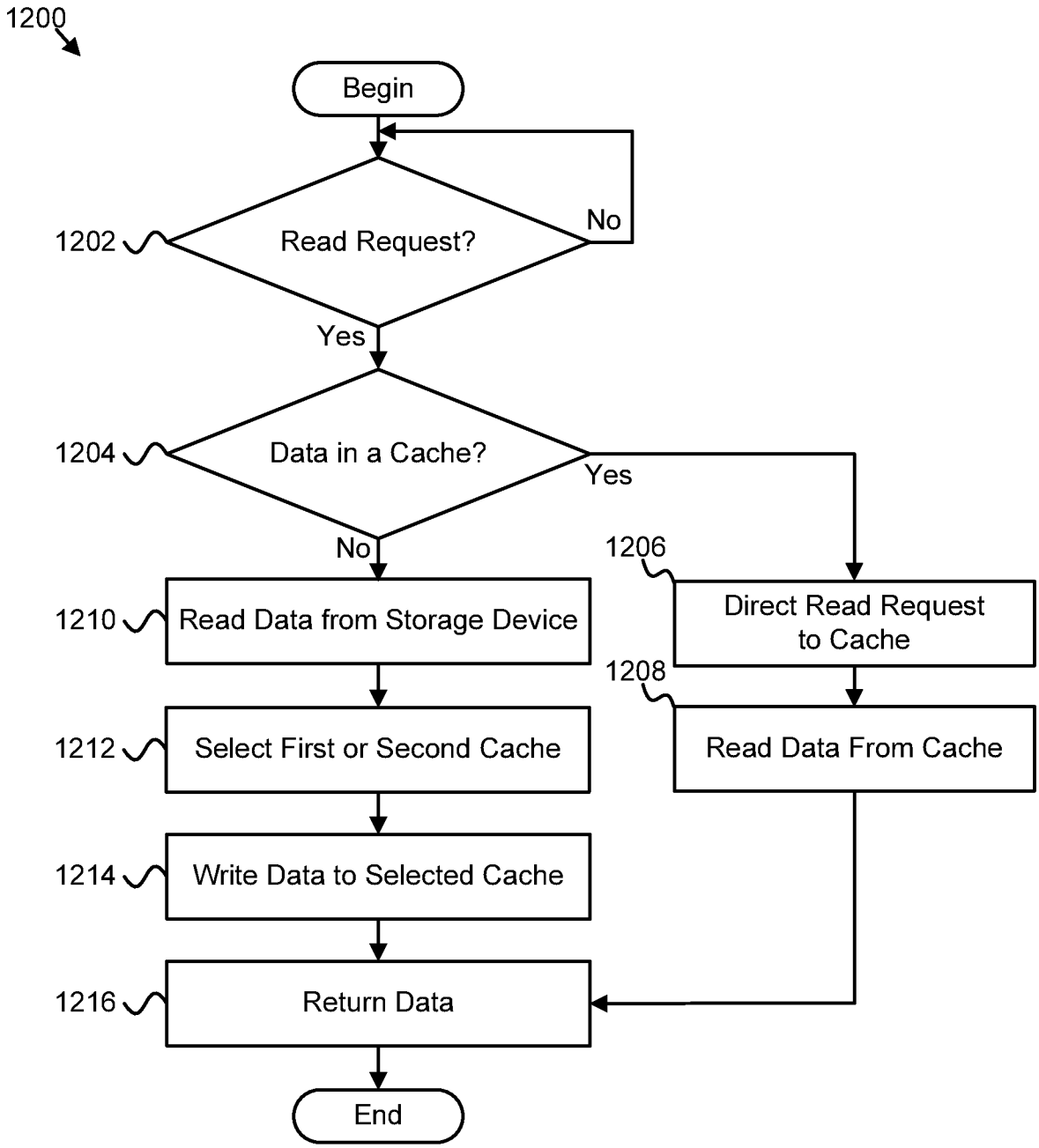
1200
Begin
1202
Read Request?
No
Yes
1204
Data in a Cache?
Yes
No
1206
Direct Read Request to Cache
1208
Read Data From Cache
1210
Read Data from Storage Device
1212
Select First or Second Cache
1214
Write Data to Selected Cache
1216
Return Data
End

FIG. 8

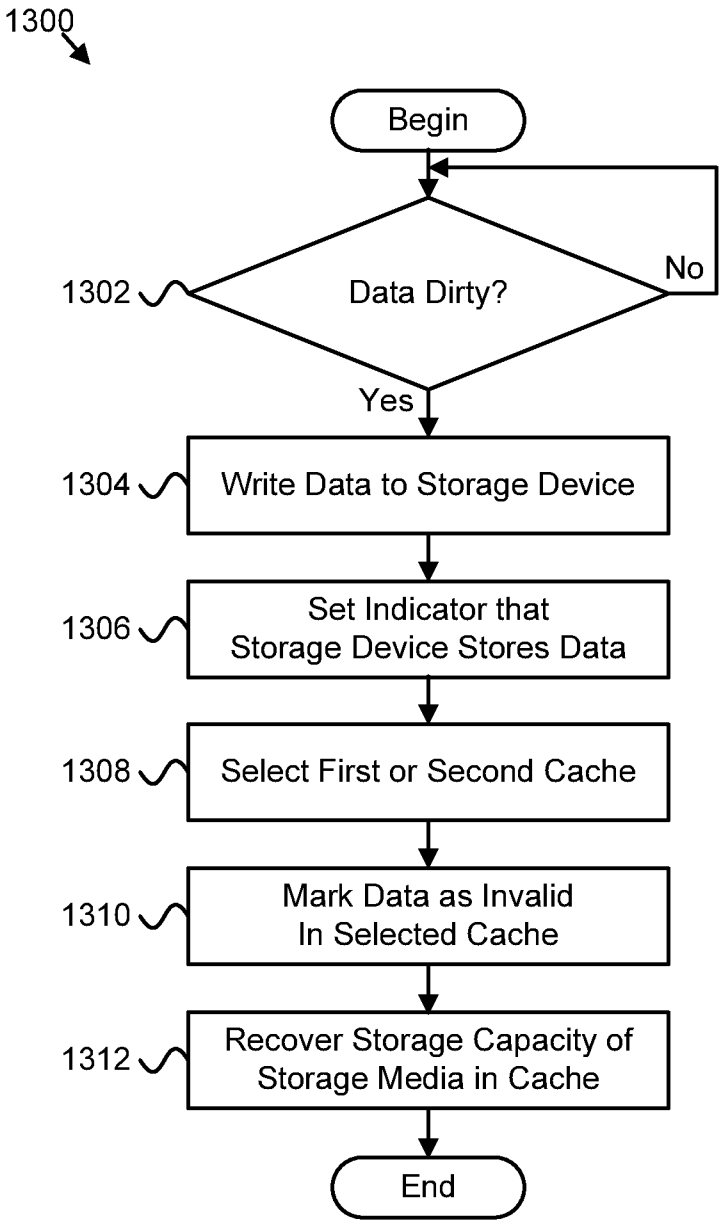
1300
Begin
1302
Data Dirty?
No
Yes
1304
Write Data to Storage Device
1306
Set Indicator that Storage Device Stores Data
1308
Select First or Second Cache
1310
Mark Data as Invalid In Selected Cache
1312
Recover Storage Capacity of Storage Media in Cache
End

FIG. 9