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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
25 August 2016 (25.08.2016)

WIPO | PCT

(10) International Publication Number
WO 2016/134303 A1

(51) International Patent Classification:
G06F 3/06 (2006.01) *G06F 12/02* (2006.01)
G06F 11/10 (2006.01)

(21) International Application Number:
PCT/US2016/018751

(22) International Filing Date:
19 February 2016 (19.02.2016)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
14/627,825 20 February 2015 (20.02.2015) US

(71) Applicant: **NETAPP, INC.** [US/US]; 495 East Jave Drive, Sunnyvale, California 94089 (US).

(72) Inventors: **MCKEAN, Brian D.**; 495 East Java Street, Sunnyvale, California 94089 (US). **UMMADI, Sandeep Kumar R.**; 495 East Java Street, Sunnyvale, California 94089 (US).

(74) Agents: **KELTON, Thomas W.** et al.; Haynes and Boone, LLP, 2323 Victory Avenue, Suite 700, Dallas, Texas 75219 (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, BN, 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, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, 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, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, 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,

[Continued on next page]

(54) Title: SOLID STATE DEVICE PARITY CACHING IN A HYBRID STORAGE ARRAY

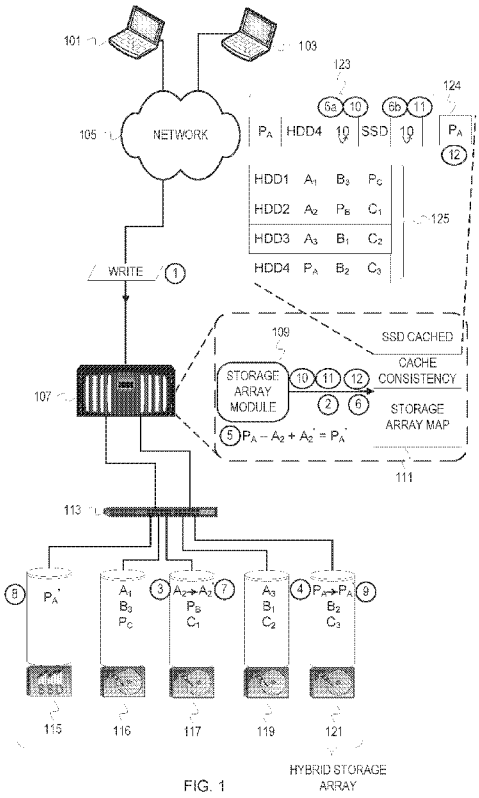


FIG. 1

(57) Abstract: A storage manager can reduce the overhead of parity based fault tolerance by leveraging the access performance of SSDs for the parities. Since reading a parity value can be considered a small read operation, the reading of parity from an SSD is an effectively "free" operation due to the substantially greater SSD read performance. With reading parity being an effectively free operation, placing parity on SSDs eliminates the parity read operations (in terms of time) from the parity based fault tolerance overhead. A storage manager can selectively place parity on SSDs from HDDs based on a criterion or criteria, which can relate to frequency of access to the data corresponding to the parity. The caching criterion can be defined to ensure the reduced overhead gained by reading parity values from a SSD outweighs any costs (e.g., SSD write endurance).

WO 2016/134303 A1

SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

— *with international search report (Art. 21(3))*

SOLID STATE DEVICE PARITY CACHING IN A HYBRID STORAGE ARRAY

BACKGROUND

Aspects of the disclosure generally relate to the field of storage systems, and more particularly to storing parity in storage systems.

Although solid state storage drives are available, storage solutions have been offered with hard disk drive arrays because of the greater cost of solid state drives. As the cost of solid state drives (SSDs) has decreased, storage solutions have increased use of the solid state drives due to the performance benefits of SSDs. However, SSDs are still more expensive than hard disk drives (HDDs) and numerous HDD based storage systems are already deployed. To gain some of the performance benefits of SSDs while controlling costs, some storage solutions incorporate both types of storage devices. These storage solutions are referred to as hybrid storage arrays or hybrid arrays.

SUMMARY

A storage manager can reduce the overhead of parity based fault tolerance by leveraging the access performance of SSDs for the parities. Since reading a parity value can be considered a small read operation, the reading of parity from an SSD is an effectively “free” operation due to the substantially greater SSD read performance. With reading parity being an effectively free operation, placing parity on SSDs eliminates the parity read operations (in terms of time) from the parity based fault tolerance overhead. A storage manager can selectively place parity on SSDs from HDDs based on a criterion or criteria, which can relate to frequency of access to the data corresponding to the parity. The caching criterion can be defined to ensure the reduced overhead gained by reading parity values from a SSD outweighs any costs (e.g., SSD write endurance).

BRIEF DESCRIPTION OF THE DRAWINGS

Aspects of the disclosure may be better understood by referencing the accompanying drawings.

Figures 1 and 2 illustrate example single parity caching in a solid state drive of a hybrid storage array. Figure 1 illustrates initial caching of a recalculated parity, and Figure 2 illustrates use of the cached parity.

Figure 3 and Figure 4 illustrate an example of double parity caching in a hybrid storage array configured with row diagonal parity and dedicated parity drives.

Figures 5-7 are flowcharts of example operations for single parity caching on a solid state drive of a hybrid storage array.

Figure 8 depicts a flowchart of example operations for evicting cached parity from a solid state storage device in a hybrid storage array.

Figure 9 is a flowchart of example operations for parity caching in a hybrid storage array that uses multiple parity protection.

Figure 10 is a flowchart of example operations for parity eviction from a solid state storage device cache in a hybrid storage array with multiple parity protection.

Figure 11 depicts an example computer system with a storage array controller that caches parity on a constituent solid state storage device.

DESCRIPTION

The description that follows includes example systems, methods, techniques, and program flows that embody aspects of the disclosure. However, it is understood that this disclosure may be practiced without these specific details. For instance, this disclosure refers to standard RAID 5 and RAID DP in illustrative examples. But aspects of this disclosure can be applied to other Storage Networking Industry Association (SNIA) defined standard parity based fault tolerance storage configurations (e.g., RAID 6) and other non-standard parity based fault tolerance storage configurations (e.g., RAID 5E). In other instances, well-known instruction instances, protocols, structures and techniques have not been shown in detail in order not to obfuscate the description.

Introduction

In a hybrid storage array, the SSDs and HDDs can be used regardless of device type. For instance, data blocks can be distributed across the drives without regard to the drive being a SSD or a HDD. In some hybrid storage array configurations, some or all of the SSDs can be used as if a cache for the HDDs within the storage array. This is in addition to any caching done at a storage manager for all of the storage arrays managed by the storage manager. When an SSD is used as cache for the hybrid storage array, the data blocks are written to the caching SSD and the HDDs of the hybrid storage array in accordance with a caching algorithm. Regardless of the particular caching scheme, the hybrid storage array can be configured in accordance with a fault tolerance configuration, such as a RAID configuration.

A RAID configuration provides fault tolerance for a storage array. The benefit of fault tolerance, however, incurs overhead for each data update. In a single parity configuration, this overhead can generally be quantified as four input/output (I/O) operations:

1) reading the data block impacted by the update,
2) reading the parity for the stripe that includes the impacted data block,
3) writing the updated data, and
4) writing the updated parity.

In a two parity configuration, such as RAID 6 or RAID DP, this overhead can generally be quantified as six I/O operations:

1) reading the data block impacted by the update,
2) reading the first parity for the stripe that includes the impacted data block,
3) reading the second parity for the stripe that includes the impacted data block,
4) writing the updated data,
5) writing the updated first parity, and
6) writing the updated second parity.

Overview

A storage operating system or storage manager can reduce the overhead of parity based fault tolerance by leveraging the access performance of SSDs for the parities. Although performance can vary across SSDs, small/random read operations have been measured in SSDs to be 100-1000x times faster than in HDDs. Since reading a parity value can be considered a small read operation, the reading of parity from an SSD is an effectively "free" operation due to the substantially greater SSD read performance. With reading parity being an effectively free operation, placing parity on SSDs eliminates the parity read operations (in terms of time) from the parity based fault tolerance overhead. A storage operating system or storage manager can selectively place parity on an SSD of a hybrid storage array based on a criterion or criteria, which can relate to frequency of access to the data corresponding to the parity. This "caching criterion" can be defined to ensure the reduced overhead gained by reading parity values from a SSD outweighs any costs (e.g., SSD write endurance).

Example Illustrations

Figures 1 and 2 illustrate example single parity caching in a solid state drive of a hybrid storage array. Figure 1 illustrates initial caching of a recalculated parity, and Figure 2 illustrates use of the cached parity. Both Figure 1 and Figure 2 are annotated with a series of numbers, 1-12. These numbers represent stages of operations. Although these stages are ordered for this example, the stages illustrate one example to aid in understanding this disclosure and should not be used to limit the claims. Claimed subject matter can vary with respect to the order and some of the operations.

Figure 1 depicts client devices 101, 103 in communication, via a network 105, with a storage manager 107. The storage manager 107 manages access to at least one hybrid storage array. In this example illustration, the storage manager 107 accesses the storage array via a network element 113 (e.g., a switch). The storage array includes a solid state drive 115 and four hard disk drives 116, 117, 119, 121. For this illustration, data caching in the solid state drive 115 is not depicted to avoid distracting from the description of the parity caching. But both parity and data can be cached in the solid state drive 115 for

the HDDs of the storage array. The storage array already hosts three stripes of data. These stripes (A, B, and C) are logically depicted in a horizontal arrangement across the storage array. Stripe A consists of data block A_1, data block A_2, and data block A_3. The parity for stripe A is indicated as P_A. Stripe B consists of data block B_1, data block B_2, and data block B_3. The parity for stripe B is indicated as P_B. Stripe C consists of data block C_1, data block C_2, and data block C_3. The parity for stripe C is indicated as parity P_C.

Figure 1 also depicts a storage array module 109 instantiated on the storage manager 107. The storage array module 109 can be an executing program that manages access to the storage array, a card or application specific integrated circuit designed for storage array management, etc. As examples, the storage array module 109 can be a storage operating system, a part of a storage operating system (e.g., a kernel process), a driver (e.g., RAID driver), an add-on to a driver, etc. The storage array module 109 maintains one or more data structures 111. The one or more data structures 111 have data 125, data 123, and data 124. The data 125 indicate locations of data and parity within the storage array ("storage array map"). The data 123 indicate parity consistency status with respect to the cache in the storage manager 107 ("cache consistency status"). The data 124 indicate which parities are cached in the caching SSD ("SSD cached"). Before any SSD caching, the data 124 will be empty or non-existent. Since this illustration stores a parity (when SSD cached) in both the caching SSD and an HDD, the data 123 indicates two consistency statuses for a single parity. The data 123 indicates a consistency status for a parity between the storage controller cache and the HDD and between the storage controller cache and the caching SSD. In the depicted data of the storage array map 125 and the parity status 123, the drives are identified as follows: the solid state drive 115 is 'SSD', the hard disk drive 116 is 'HDD1', the hard disk drive 117 is 'HDD2', the hard disk drive 119 is 'HDD3', and the hard disk drive 121 is 'HDD4'.

At a stage 1, one of the client devices 101, 103 communicate a write request to the storage manager 107 via the network 105. Data in the storage array may have been written by either of both of the client devices 101, 103. The communicated write request will update data already written into the storage array.

At stage 2, the storage array module 109 initially processes the write request. The storage array module 109 determines that the write request impacts the data block A_2. The write request can indicate a file handle and offset used by the storage array module 109 to determine the impacted data block. The storage array module 109 accesses the data structure 111 to determine location of the data block A_2. With the storage array map 125, the storage array module 109 determines that the data block A_2 is stored on the hard disk drive 117 and determines that the parity P_A (i.e., the parity value for stripe A) is stored on the hard disk drive 121. The storage array module 109 also accesses the SSD cached data 123 and determines that the parity P_A is not currently cached.

After determining their locations, the storage array module 109 obtains A_2 and P_A. At stage 3, the storage array module 109 reads A_2 from the hard disk drive 117. At stage 4, the storage array module 109 reads P_A from the hard disk drive 121.

After obtaining the impacted data block and the corresponding parity value, the storage array module 109 generates an updated data block and recalculates the corresponding parity. The updated data block is depicted as A_2'. At stage 5, the storage array module 109 recalculates the parity for stripe A to reflect the change to the data block A_2. For this illustration, parity is generated by XORing data blocks of a stripe. To calculate the new parity value, the data block A_2 is "subtracted" from the parity value P_A, and the updated data block A_2' is added to the result of the subtraction. With XORing, the addition and subtraction are XOR operations. The recalculated parity value is depicted as P_A'. Both the updated data block and the recalculated parity are cached in memory of the storage manager 107.

At stage 6, the storage array module 109 updates the cache consistency status 123 for the parity for stripe A. Stage 6 is decomposed into stages 6a and 6b and depicted adjacent to the cache consistency status 123. Again, the indication of "a" and "b" are used to illustrate the different changes and not to indicate any particular order. At stage 6a, the storage array module 109 indicates that the stripe A parity as stored in the storage manager 107 is inconsistent with the stripe A parity in the hard disk drive 121. At stage 6b, the storage array module 109 indicates that the stripe A parity as stored in the storage manager 107 is inconsistent with the stripe A parity in the solid state drive 115. In this case, the

inconsistency with the solid state drive 115 exists because there is no stripe A parity in the solid state drive 115 yet. The storage array module 109 could, instead, initially use a null value or other similar indication in the cache consistency status 123 to indicate that the parity is not yet stored in the solid state drive 115. As another example, the storage array module 109 may not maintain a consistency status for the parity until the parity is actually stored in the solid state drive 115. Parity is likely not cached until a criterion or criteria are satisfied. But this example assumes that a parity caching criterion is satisfied. In Figure 1, the cache consistency status 123 is structured to index by parity (i.e., each row corresponds to parity). For each parity of the hybrid storage array, the cache consistency status 123 indicates a consistency bit for each location beyond the storage controller cache and those locations. In this case, those locations include the caching SSD and the HDD. Initially, cache consistency status 123 can indicate null values for the locations beyond the storage controller cache.

After updating parity status, the storage array module 109 writes the updated data block and the recalculated parity to the storage array. At stage 7, the storage array module 109 writes A_2' to the hard disk drive 117. At stage 8, the storage array module 109 writes P_A' to the solid state drive 115. At stage 9, the storage array module 109 writes P_A' to the hard disk drive 121.

Each drive will return an acknowledgement to the storage manager 107 after successful completion of the requested write and cache consistency status will be updated accordingly. At stage 10, the storage array module 109 updates the cache consistency status 123 in accordance with the acknowledgement received from the hard disk drive 121 to indicate consistency with the storage controller cache. After the updated data block has been successfully written by the hard disk drive 117, the storage array module 109 will update cache consistency status for the data block (not depicted). After the solid state drive 115 indicates successful completion of the requested write, the storage array module 109 updates the cache consistency status for the SSD cached stripe A parity to indicate consistency between the SSD cache and the storage controller cache for the stripe A parity.

Since P_A is now being cached, the storage array module 109 also updates data to reflect the SSD caching of parity. At stage 12, the storage array module 109 updates the

SSD cached data 124 to indicate P_A. For this illustration, this update can be done concurrently with stage 11, after acknowledgement from the solid state drive 115, or after stage 11. If done prior to stage 11, the cache consistency status 123 could be used to prevent an attempt to read P_A from the caching SSD prematurely or the SSD cached data 124 can indicate that P_A is not yet ready to be read from the caching SSD.

Figure 2 continues the illustration from Figure 1 and depicts use of the SSD cached, recalculated parity. For Figure 2, one of the client devices 101, 103 sends another write request that impacts a data block protected by the cached stripe A parity at stage 1. Although some of the operations between Figures 1 and 2 are similar, stripe A parity is not read from a hard disk drive in Figure 2.

At stage 2, the storage array module 109 processes the write request. The storage array module 109 determines that the write request impacts the data block A_3. The write request can indicate a file handle and offset used by the storage array module 109 to determine the impacted data block. The storage array module 109 accesses the one or more data structures 111 to determine location of the data block A_3. With the storage array map 125, the storage array module 109 determines that the data block A_3 is stored on the hard disk drive 119 and determines that the stripe A parity is stored on the hard disk drive 121. The storage array module 109 also accesses the SSD cached data 124 and determines that parity for stripe A is currently cached in the solid state drive 115. Since the parity for stripe A is SSD cached, the storage array module 109 will read the stripe A parity from the solid state drive 115 instead of the hard disk drive 121.

After determining their locations, the storage array module 109 obtains A_3 and P_A'. At stage 3, the storage array module 109 reads A_3 from the hard disk drive 119. At stage 4, the storage array module 109 reads P_A' from the solid state drive 115. As mentioned earlier, this parity read from the solid state drive 115 instead of the hard disk drive 121 reduces the time overhead for updating parity.

After obtaining the impacted data block and the corresponding parity value, the storage array module 109 generates an updated data block and recalculates the corresponding parity. The updated data block is depicted as A_3'.At stage 5, the storage

array module 109 recalculates the parity for stripe A to reflect the change to the data block A_3. To calculate the new parity value, the data block A_3 is subtracted from the parity value P_A', and the updated data block A_3' is added to the result of the subtraction. The recalculated parity value is depicted as P_A''.

At stage 6, the storage array module 109 updates the cache consistency status 123 to indicate that the parity for stripe A in both the hard disk drive 121 and the solid state drive 115 are inconsistent with that stored in the storage controller cache. As in Figure 1, stage 6 is decomposed into stages 6a and 6b. At stage 6a, the storage array module 109 indicates that the stripe A parity as stored in the storage manager 107 is inconsistent with the stripe A parity in the hard disk drive 121. At stage 6b, the storage array module 109 indicates that the stripe A parity as stored in the storage manager 107 is inconsistent with the stripe A parity in the solid state drive 115.

After updating cache consistency status, the storage array module 109 writes the updated data block and the recalculated parity to the storage array. At stage 7, the storage array module 109 writes A_3' to the hard disk drive 119. At stage 8, the storage array module 109 writes P_A'' to the solid state drive 115. At stage 9, the storage array module 109 writes P_A'' to the hard disk drive 121.

Each drive will return an acknowledgement to the storage manager 107 after successful completion of the requested write and cache consistency will be updated accordingly. At stage 10, the storage array module 109 updates the cache consistency status 123 in accordance with the acknowledgement received from the hard disk drive 121. After the updated data block has been successfully written by the hard disk drive 117, the storage array module 109 will update cache consistency status accordingly (not depicted). After the solid state drive 115 indicates successful completion of the requested write, the storage array module 109 updates the cache consistency status 123 to indicate consistency between the solid state drive 115 and the storage controller cache. Since SSD cache data 124 already indicates SSD caching of the parity for stripe A, no updates are necessary to the data 124.

Figures 1 and 2 depicted SSD single parity caching for a hybrid storage array. A storage array can be protected with greater parity, though. A storage array can be protected with double parity, triple parity, etc. Calculation and arrangement of parity can also vary. For example, the first parity of a double parity configuration can be calculated with XOR operations as depicted in Figures 1 and 2. The second parity, however, can be calculated with a finite field or Galois field based encoding technique (e.g., Reed Solomon encoding). As another example, parity can be calculated with XOR operations and different data striping. The first parity can be calculated with horizontal striping and XOR operations, and the second parity can be calculated with diagonal striping and XOR operations (i.e., row diagonal parity).

Figure 3 and Figure 4 illustrate an example of double parity SSD caching in a hybrid storage array configured with row diagonal parity and dedicated parity drives. In Figures 1 and 2, cache consistency was maintained for the storage controller cache. The storage controller cache had 2 parity caching relationships: 1) an HDD relationship, and 2) an SSD relationship. The two relationships could be considered an implicit parity caching relationship between the SSD and the HDDs. If stripe A parity on the SSD is inconsistent with the storage controller cache and stripe A parity on the HDD is consistent with the storage controller cache, then the different consistencies indicate an inconsistency between the SSD and the HDD. Unlike Figures 1 and 2, parity caching relationships are maintained between a caching SSD and the HDDs as depicted in Figures 3 and 4. Figure 3 illustrates initial SSD caching of both recalculated parities, and Figure 4 illustrates use of the SSD cached parities. Both Figure 3 and Figure 4 are annotated with a series of numbers 1-13. These numbers represent stages of operations, similar to Figures 1 and 2. Although these stages are ordered for this example, the stages illustrate one example to aid in understanding this disclosure and should not be used to limit the claims. Claimed subject matter can vary with respect to the order and some of the operations.

Figure 3 depicts client devices 301, 303 in communication, via a network 305, with a storage manager 307. The storage manager 307 manages access to a hybrid storage array. In this example illustration, the storage manager 307 accesses the storage array via a network element 313 (e.g., a switch). The storage array includes a solid state drive 315 and six hard disk drives 316, 317, 319, 320, 321, 322. Similar to the example of Figures 1 and

2, the solid state drive 315 is configured as a cache for the HDDs of the storage array. The storage array already hosts four groups of data blocks and corresponding parity. The groups of data blocks include data blocks A_1- A_3, data blocks B_1-B_3, data blocks C_1-C_3, and data blocks D_1-D_3. The row parity stored in the hybrid storage array includes RP_1-RP_3. The diagonal parity stored in the hybrid storage array includes DP_1-DP_3.

The storage array module 309 maintains one or more data structures 311. The one or more data structures 311 have storage array map data 325, cache consistency data 323, and SSD cache consistency data 326. In this illustration, the cache consistency 323 indicates consistency of parity between cache memory of the storage manager 307 and the solid state drive 315. The SSD cache consistency data 326 indicates consistency of parity between the caching SSD and relevant HDDs. In the depicted data, the drives are identified as follows: the solid state drive 315 is SSD, the hard disk drive 316 is HDD1, the hard disk drive 317 is HDD2, the hard disk drive 319 is HDD3, the hard disk drive 320 is HDD4, the hard disk drive 321 is HDD5, and the hard disk drive 322 is HDD6.

At stage 1, one of the client devices 301, 303 communicates a write request to the storage manager 307 via the network 305. Data in the storage array may have been written by either or both of the client devices 301, 303. The communicated write request will update data already written into the storage array.

At stage 2, the storage array module 309 initially processes the write request. The storage array module 309 determines that the write request impacts the data block B_1. The storage array module 309 accesses the data structure 311 to determine location of the data block B_1 and corresponding parities. The cache consistency data 323 is arranged differently than in Figure 1 to show that consistency data may not be maintained for a consistency relationship between the storage manager cache and the HDDs, at least with respect to cached SSD parity. With the storage array map 325, the storage array module 309 determines that the data block B_1 is stored on the hard disk drive 317. The storage array module 309 also determines that the corresponding row parity RP_1 is stored on the hard disk drive 321 and that the corresponding diagonal parity DP_2 is stored on the hard disk drive 322. The storage array module 309 also determines that neither the row parity nor the diagonal parity is cached in the solid state drive 315 with the SSD cache

consistency data 326 since it will initially have no indication of either parity. Indeed, the SSD cache consistency data 326 may be non-existent or a corresponding structure not instantiated until parity is first cached (or attempted to be cached) in the solid state drive 315.

After determining locations, the storage array module 309 obtains B_1, RP_1, and DP_2. At stage 3, the storage array module 309 reads B_1 from the hard disk drive 317. At stage 4, the storage array module 309 reads RP_1 from the hard disk drive 321. At stage 5, the storage array module 309 reads DP_2 from the hard disk drive 322.

After obtaining the impacted data block and the corresponding parity values, the storage array module 309 generates an updated data block and recalculates the corresponding parity values. The updated data block is depicted as B_1'. At stage 6, the storage array module 309 recalculates the row parity value for B_1 to reflect the change to the data block B_1. For this illustration, row parity is generated by XORing data blocks of a row stripe. To calculate the new row parity value, the data block B_1 is subtracted from the parity value for RP_1, and the updated data block B_1' is added to the result of the subtraction. The recalculated row parity RP_1 is depicted as RP_1'. At stage 7, the storage array module 309 recalculates the diagonal parity value for B_1 to reflect the change to the data block B_1. The diagonal parity is generated by XORing data blocks of a diagonal stripe. To calculate the new diagonal parity value, the data block B_1 is subtracted from the parity value of DP_2, and the updated data block B_1' is added to the result of the subtraction. The recalculated diagonal parity DP_2 is depicted as DP_2'.

At stage 8, the storage array module 309 updates the cache consistency data 323 for the recalculated parities. Stage 8 is decomposed into stages 8a and 8b and depicted adjacent to the cache consistency data 323. Again, the indication of "a" and "b" are used to illustrate the different changes and not to indicate any particular order. At stage 8a, the storage array module 309 indicates that RP_1 as stored in the storage manager cache is inconsistent with RP_1 as stored (or not stored in this case) in the solid state drive 315. As described earlier, in Figure 1, various techniques can be used to account for the initial lack of any parity in the caching SSD. At stage 8b, the storage array module 309 indicates that DP_2 as stored in the storage manager cache is inconsistent with RP_1 as stored (or not stored

in this case) in the solid state drive 315. In both stages 8a and 8b, the storage array module 309 also updates the cache consistency data 323 to indicate "SSD" in order to explicit identify the consistency relationship as being between the storage manager cache and the caching SSD. Although this is not necessary in some cases, a data structure with the cache consistency data 323 may also indicate consistency between the storage manager cache and other drives (even other caching SSDs) of the storage array. The storage array module 309 also updates cache consistency data for the data block B_1 in the hard disk drive 317 to indicate inconsistent (not depicted).

After updating cache consistency data 323, the storage array module 309 writes the updated data block and the recalculated parity values to the storage array. At stage 9, the storage array module 309 writes B_1' to the hard disk drive 317. At stage 10, the storage array module 309 writes RP_1' to the solid state drive 315. At stage 11, the storage array module 309 writes DP_2' to the solid state drive 315.

Each drive will return an acknowledgement to the storage manager 307 after successful completion of the requested write and the storage array module 309 with update cache consistency data accordingly. The storage array module 309 updates the cache consistency data of the data block and parities in accordance with the acknowledgements received from the drives. After the updated data block B_1 has been successfully written by the hard disk drive 317, the storage array module 309 will update cache consistency data to indicate consistent (not depicted). After the solid state drive 315 indicates successful completion of both requested writes, the storage array module 309 updates the cache consistency data 323 to indicate consistent for the cached row parity and diagonal parity at stage 12. As with stage 8, stage 12 is decomposed into stages 12a and 12b. The storage array module 309 updates the cache consistency data 323 for RP_1 in stage 12a and for DP_2 in stage 12b. If the requested write for both parities was a single request, a single confirmation or acknowledgement from the solid state drive 315 can trigger stage 12.

Since RP_1 and DP_2 are now being SSD cached, the storage array module 309 also creates or updates the SSD cache consistency data 326 to reflect the SSD caching of the parities. At stage 13, the storage array module 309 updates the SSD cache consistency data 326 to indicate that RP_1 on the hard disk drive 321 is inconsistent with RP_1 on the

solid state drive 315. The storage array module 309 also updates the SSD cache consistency data 326 to indicate that DP_2 on the hard disk drive 322 is inconsistent with DP_2 on the solid state drive 315. For this illustration, this update of the SSD cache consistency data 326 can be done concurrently with stage 12, after stage 12, or after acknowledgement from the solid state drive 315. This update to the SSD cache consistency data 326 could also be done in connection with stage 8. In other words, the SSD cache consistency data 326 can be created/updated based upon an attempted SSD caching of the parity. Regardless of when parity on a HDD is marked as inconsistent with SSD cached parity, it is not updated to consistent or removed until the SSD cached parity is evicted or selected for eviction. If a change in designated parity drive occurs, then the SSD cache consistency data 326 is updated to reflect the change in parity drive.

Figure 4 continues the illustration from Figure 3 and depicts use of the cached, recalculated parity values. For Figure 4, one of the client devices 301, 303 sends another write request that impacts a data block protected by the cached diagonal parity DP_2' at stage 1. Although some of the operations between Figures 3 and 4 are similar, the diagonal parity DP_2' is not stored on the hard disk drive 322.

At stage 2, the storage array module 309 processes the write request. The storage array module 309 determines that the write request impacts the data block C_2. The storage array module 309 accesses the data structure 311 to determine location of the data block C_2. With the storage array map 325, the storage array module 309 determines that the data block C_2 is stored on the hard disk drive 319. The storage array module 309 also determines the locations of the parity values for the data block C_2. The storage array module 309 determines that the row parity for the data block C_2 (RP_2) is on the hard disk drive 321 with the storage array map 325 and is not cached (the SSD cache consistency data 326 does not indicate RP_2). The storage array module 309 determines that the corresponding diagonal parity for the data block C_2 is cached on the solid state drive 315 according to the SSD cached consistency data 326. Thus, the storage array module 309 disregards the inconsistent diagonal parity DP_2 on the hard disk drive 322.

After determining locations, the storage array module 309 obtains C_2 and the corresponding parities, RP_2 and DP_2'. At stage 3, the storage array module 309 reads C_2

from the hard disk drive 319. At stage 4, the storage array module 309 reads RP_2 from the hard disk drive 321. At stage 5, the storage array module 309 reads the diagonal parity value DP_2' from the solid state drive 315 in accordance with the SSD cache consistency data 326.

After obtaining the impacted data block and the corresponding parity values, the storage array module 309 generates an updated data block and recalculates the corresponding parity values. The updated data block is depicted as C_2'. At stage 6, the storage array module 309 recalculates the row parity value for C_2 to reflect the change to the data block C_2. To calculate the new row parity value, the data block C_2 is subtracted from the parity value RP_2, and the updated data block C_2' is added to the result of the subtraction. The recalculated row parity value is depicted as RP_2'. At stage 7, the storage array module 309 recalculates the diagonal parity value for C_2 to reflect the change to the data block C_2. To calculate the new diagonal parity value, the data block C_2 is subtracted from the parity value DP_2', and the updated data block C_2' is added to the result of the subtraction. The recalculated diagonal parity value is depicted as DP_2''.

At stage 8, the storage array module 309 updates the cache consistency data 323 to indicate inconsistency between the storage manager cache and the solid state drive 315 for the recalculated parities. Stage 8 is decomposed into stages 8a and 8b again. At stage 8a, the storage array module 309 indicates that DP_2 as stored in the storage manager cache is inconsistent with DP_2 as stored (or not stored in this case) in the solid state drive 315. At stage 8b, the storage array module 309 indicates that RP_2 as stored in the storage manager cache is inconsistent with RP_2 as stored (or not stored in this case) in the solid state drive 315. As described earlier, various techniques can be used to account for the initial lack of a parity in the caching SSD. In both stages 8a and 8b, the storage array module 309 also updates the cache consistency data 323 to indicate "SSD" in order to explicit identify the consistency relationship as being between the storage manager cache and the caching SSD, unless already indicated therein. The storage array module 309 also updates status of the data block C_2 to indicate that the hard disk drive 319 is inconsistent with the storage manager cache for the data block C_2.

After updating consistency data, the storage array module 309 writes the updated data block and the recalculated parity values to the hybrid storage array. At stage 9, the storage array module 309 writes C_2' to the hard disk drive 319. At stage 10, the storage array module 309 writes RP_2' to the solid state drive 315. At stage 11, the storage array module 309 writes DP_2" to the solid state drive 315.

Each drive will return an acknowledgement to the storage manager 307 after successful completion of the requested write and the storage manager will update cache consistency accordingly. After the updated data block C_2 has been successfully written by the hard disk drive 319, the storage array module 309 will update cache consistency data accordingly (not depicted). After the solid state drive 315 indicates successful completion of requested writes of the recalculated parities, the storage array module 309 updates the cache consistency data 323 to indicate consistency between the storage manager cache and the solid state drive 315 with respect to the row parity RP_2 and the diagonal parity DP_2.

Since RP_2 is now being SSD cached, the storage array module 309 updates the SSD cache consistency data 326 to reflect the SSD caching of RP_2. At stage 13, the storage array module 309 updates the SSD cache consistency data 326 to indicate that RP_2 on the hard disk drive 321 is inconsistent with RP_2 on the solid state drive 315.

Figures 5-7 are flowcharts of example operations for single parity caching on a solid state drive of a hybrid storage array. In the flowchart, dashed lines are used to depict asynchronous operations (e.g., waiting on a response or acknowledgement). For continuity of this description, the example operations are described as if performed by a storage manager. In addition, the parity caching operations can operate regardless of caching/buffering done internally by the storage manager (e.g., in non-volatile random access memory) prior to writes being committed to a storage array.

At block 501, a storage manager detects a data update for a hybrid storage array. The storage manager may receive a write request that impacts at least one data block stored on a storage device in the hybrid storage array, and determines the identity of the impacted data block. The data update may be generated internally with respect to the storage manager. For instance, the data update may be a data synchronization operation.

At block 503, the storage manager determines locations of the data block and corresponding parity value in the hybrid storage array affected by the data update. The storage manager determines locations with mapping data. The mapping data indicates logical addresses used by clients and/or the storage manager and corresponding addresses used by the storage devices. The addresses used by the storage devices may also be logical addresses or may be physical addresses. For example, the storage manager may access mapping data that maps logical block addresses to physical block addresses.

At block 505, the storage manager determines whether the parity is already cached on a solid state storage device of the hybrid storage array. The storage manager can maintain data that indicates whether a parity is cached in a solid state storage device of the hybrid storage array. For example, a data structure can be maintained (e.g., list, array, table, etc.) that indicates cached parities by device identifier (e.g., network address). The device identifier identifies the solid state storage device being used as the caching SSD. As another example, a storage manager can maintain a flag or bit in the storage array map to indicate whether a parity is cached in the caching SSD. If the parity is cached, then control flows to block 507. If the parity is not cached, then control flows to block 701 of Figure 7.

At block 507, the storage manager updates the impacted data block and recalculates the corresponding parity in accordance with the updated data block. After generating the updated data block and recalculated parity, the storage manager also updates the cache consistency status of the impacted data block.

At block 509, the storage manager indicates two consistency statuses as inconsistent for the corresponding parity that has been recalculated. The storage manager indicates consistency statuses for the hard disk storage device the caching solid state storage device as inconsistent with the storage manager cache.

At block 511, the storage manager writes the updated data block to the hybrid storage array. For instance, the storage manager communicates a write request or write command to the storage device that hosts the impacted data block.

After confirmation that the updated data block has been written, the storage manager updates consistency status for the data block to clean at block 515.

After (or concurrently with) writing the updated data block, the storage manager writes the recalculated parity to the solid state drive being used for parity caching and to the hard disk storage device at block 513. The storage manager communicates different write commands or write requests to the different storage devices. The storage manager can be designed to presume that the write to the solid state storage device will complete more quickly than the write to the hard disk storage device. Thus, the storage manager can prioritize writes to the hard disk storage devices over writes to the solid state storage device.

After confirmation that the parity has been updated at each storage device, the storage manager updates each status accordingly at block 517. After confirmation from the caching SSD, the storage manager updates the consistency status to indicate that the caching SSD is consistent with the storage manager cache with respect to the cached parity. Likewise, the storage manager updates the consistency status for the hard disk storage device with respect to the parity to indicate consistency with the storage manager cache after confirmation from the hard disk storage device.

Figure 6 depicts example operations that can be performed when the parity corresponding to the impacted data block is already cached on the solid state storage device. The example operations depicted in Figure 6 can be used to implement block 507.

At block 601, the storage manager reads the impacted data block from the storage array based on the determined location of the impacted data block.

At block 603, the storage manager reads the parity from the solid state storage device.

At block 605, the storage manager updates the obtained data block in accordance with the data update.

At block 607, the storage manager recalculates the parity based on the updated data block.

At block 609, the storage manager indicates an inconsistent status for the impacted data block as stored on the hard disk storage device prior to requesting the updated data block be written to the hybrid storage array.

Figure 7 depicts example operations when the parity corresponding to the impacted data block has not yet been cached on a solid state storage device of the hybrid storage array.

At block 701, the storage manager reads the impacted data block from the hybrid storage array based on the determined location of the impacted data block.

At block 703, the storage manager reads the parity from the previously determined location, which is a hard disk storage device of the hybrid storage array.

At block 705, the storage manager updates the obtained data block in accordance with the data update.

At block 707, the storage manager recalculates the parity based on the updated data block.

At block 709, the storage manager indicates an inconsistent status for the impacted data block as stored on the determined hard disk storage device with respect to the storage manager cache.

At block 711, the storage manager writes the updated data block to the data block location in the hybrid storage array. After confirmation that the updated data block has been written, the storage manager updates status for the data block to consistent at block 718. Control flows from block 711 to block 713.

At block 713, the storage manager determines whether a parity caching criterion is satisfied. The parity caching criterion can be directly related to the parity itself and/or to a corresponding group of data blocks. As an example of a direct relationship, the caching criterion can set a threshold number of parity updates. Upon reaching or exceeding that threshold, the parity caching criterion is satisfied for that parity. As another example, a most recently updated or most frequently updated caching algorithm can be applied to the

parities. As an example of an indirect relationship, a parity caching criterion can include "hotness" of data. In other words, mechanisms can be employed to determine how frequently data blocks in the hybrid storage array are being updated within a given window of time. The storage manager (or some other monitoring component) can then group the frequency of updates by corresponding parity. In terms of stripes, the storage manager determines hot data stripes. If the frequency of updates grouped by parity is beyond a threshold, then the parity caching criterion is satisfied. The frequency of access can be measured in terms of number of accesses. The storage manager can then determine the parities corresponding to the hot data blocks and indicate those parities as ready for caching. As example illustrations, the storage manager can maintain a list of the parities ready for caching or set data fields (e.g., flags) associated with the parities in the storage array map to indicate activation of parity caching. A caching criterion is not necessarily reactive, as in the examples above. The storage manager can determine that an incoming (or ongoing) workload or job will access a set of data blocks beyond a threshold. The storage manager can determine the parities corresponding to those data blocks and activate parity caching for those data blocks that are expected to be or will be updated beyond the threshold. As mentioned later, a caching criterion is not necessarily employed. If the parity caching criterion is satisfied, then control flows to block 715. Otherwise, control flows to block 722.

At block 715, the storage manager determines a solid state storage device in which to cache the parity. A solid state storage device can be configured in advance to be the parity cache for a hybrid storage array. If the hybrid storage array includes multiple solid state storage devices and no particular one is specified to be a parity cache, a solid state storage device of the array can be selected. Examples of a device selection criterion include type of solid state storage device, manufacturer of solid state storage device, capacity of the solid state storage device, age of the solid state storage device, current use of the solid state storage device, etc. For instance, the oldest solid state storage device with the least capacity may be selected for parity caching to allow a newer solid state storage device with greater capacity to be used for data caching. The reverse may also be implemented for device selection (i.e., the newest SSD with most capacity selected).

At block 717, the storage manager indicates the selected solid state storage device to cache the parity.

At block 719, the storage manager indicates an inconsistent status for the parity at both the hard disk storage device location and the solid state storage device cache location. After indication of the selected solid state storage device and update of status for both locations, the operations at blocks 721 are similar to those in blocks 513 and 517 of Figure 5.

At block 721, the storage manager writes the recalculated parity to the solid state storage device being used for parity caching and to the hard disk storage device location.

After confirmation that the parity has been updated at each location, the storage manager updates each status accordingly at block 723.

If the storage manager determined at block 713 that the parity caching criterion is not satisfied, then control flows to block 722.

At block 722, the storage manager indicates inconsistent status for the parity as stored in the hard disk storage device with respect to the storage manager cache. The storage manager indicates a status for the primary location, which is a hard disk storage device in this case, as dirty.

At block 727, the storage manager writes the recalculated parity to the hard disk storage device.

After confirmation that the parity has been updated at the hard disk storage device, the storage manager updates status of parity at the hard disk storage device to consistent with the storage manager cache at block 729.

Although the space on a solid state storage device that can be used for parity caching is substantial, an eviction mechanism may still be implemented. It may be desirable to evict cached parity from a solid state storage device to regulate the parity caching based writes to the solid state storage device. Figure 8 depicts a flowchart of

example operations for evicting cached parity from a solid state storage device in a hybrid storage array. As with the above figures, Figure 8 is described as if the operations are performed by an eviction process of a storage manager.

At block 801, the eviction process begins an eviction evaluation for each parity of the hybrid storage array that is cached in a solid state storage device of the array. Although these operations are presented as iterating over each cached parity, the claims are not constrained to iterating over each cached parity. An eviction mechanism can iterate over sets of cached parity that are less than the total cached parity. The eviction mechanism can be bound by time, can end early (e.g., after evicting *x* parities), and can be interrupted. In addition, the eviction mechanism may first collect information about the cached parity in an iterative manner, and then evaluate the collected information to determine which (if any) cached parity to evict. The eviction mechanism can be triggered based on time (e.g., periodically), an event or activity (e.g., number of updates to an array, transition in a workload, etc.), attempted caching of a new parity, consumption of a threshold amount of space on the solid state storage device allocated for parity caching, etc.

At block 803, the eviction process determines whether a parity eviction criterion is satisfied. The parity eviction criterion can depend upon the parity caching criterion. For instance, the parity eviction criterion can be in accordance with a least recently updated algorithm or a least frequently updated algorithm. The parity eviction criterion can be based on the data protected by the parity. For instance, the parity corresponding to the "coldest" data (i.e., data with the least updates within a sliding window of time or window of operations) may be evicted. If the parity eviction criterion is satisfied, then control flows to block 805. If it is not, then control flows to block 811.

At block 805, the eviction process ensures that the status of the parity at the hard disk storage device location is consistent with cache of the storage manager, if relevant. If the status is inconsistent, then the storage manager waits until the status is changed to consistent. After determining that the status is consistent or if the parity is not indicated in the storage manager cache, the storage manager can obtain a lock on data elements corresponding to the parity.

At block 807, the eviction process notifies the solid state storage device caching the parity to mark the parity as invalid. For example, the storage manager can send a TRIM command to the solid state storage device.

At block 809, the eviction process indicates that the parity is no longer being cached. For instance, the storage manager can clear or remove indication of the solid state storage device that was caching the parity. The storage manager can change a bit or flag that indicates whether the associated parity is SSD cached.

At block 811, the eviction process determines whether there is additional cached parity to evaluate for possible eviction. If not, then control flows to block 813. If there is an additional cached parity to evaluate for possible eviction, control flows back to block 801.

At block 813, the eviction process waits for the next parity eviction evaluation trigger.

The above flowcharts depict example operations for the case of single parity protection. Figure 9 is a flowchart of example operations for parity caching in a hybrid storage array that uses multiple parity protection. As with the single parity Figures, the example operations of Figure 9 are described as if performed by a storage manager. Operations in Figure 9 are similar to some of the operations in Figures 5-7, but a cached parity is not written to the corresponding hard disk storage device while being cached. To reduce unnecessary repetition of operations and aid in understanding the disclosure, Figure 9 does not depict certain operations. For instance, Figure 9 does not depict determining locations, reading an impacted data block, or maintaining status of the impacted data block.

At block 901, a storage manager detects an update for a data block.

At block 903, the storage manager begins parity caching determination operations for each of m parity (m=1..n, with n>1) that corresponds to the data block. The iterative process suggested by block 903 should not be used to constrain the claims. The process can be terminated early, can exit upon satisfaction of a condition, can be interrupted, etc.

At block 905, the storage manager determines whether parity *m* is cached in a solid state storage device. If so, then control flows to block 907. If the parity *m* is not cached, then control flows to block 917.

At block 917, the storage manager reads the parity *m* from its hard disk storage device location. Since a storage manager may handle write requests (from a same or different clients) targeting different data blocks but impacting a same parity, the storage manager can ensure the parity is either not cached in the storage manager or that the parity at the hard disk storage device location is consistent with the storage manager cache. The storage manager ensures that it is reading a consistent version of the parity m, and will wait if the parity *m* is currently inconsistent. Or the storage manager can skip to the next parity and return to the currently inconsistent parity later.

At block 919, the storage manager recalculates the parity *m* based on the updated data block.

At block 921, the storage manager determines whether a parity caching criterion is satisfied for the parity *m*. If the parity caching criterion is satisfied, then control flows to block 927. Otherwise, control flows to block 923.

At block 927, the storage manager determines a solid state storage device in which to cache the parity and indicates this device for the recalculated parity cache location. A solid state storage device can be configured in advance to be the parity cache for a hybrid storage array. If the hybrid storage array includes multiple solid state storage devices and no particular one is specified to be a parity cache, a solid state storage device of the array can be selected. Examples of device selection criteria are previously given with respect to block 715.

At block 929, the storage manager indicates status of the parity *m* for the solid state storage device location as inconsistent with the storage manager cache.

At block 930, the storage manager indicates status of the parity *m* as stored in the hard disk storage device location as inconsistent with the caching solid state storage device. Control flows from block 930 to block 913.

At block 913, the storage manager writes the recalculated parity to the solid state storage device being used for parity caching. Control flows from block 913 to block 915, as well as asynchronously to block 935.

At block 915, the storage manager determines whether there is an additional parity corresponding to the updated data block. If there is an additional parity corresponding to the updated data block, then control returns to block 903. Otherwise, control flows to block 931. At block 931, the storage manager indicates that *m* parities have been updated for the data block.

After confirmation that the parity *m* has been written to the SSD cache location, the storage manager updates parity status for the SSD cache location to indicate consistency with the storage manager cache at block 935.

If the storage manager determines at block 921 that the parity caching criterion is not satisfied, then control flows to block 923.

At block 923, the storage manager indicates status of the parity as stored in the hard disk storage device as inconsistent with the storage manager cache. This status is indicated for the hard disk storage device location alone, since the parity is not being cached.

At block 925, the storage manager writes the recalculated parity to the hard disk storage device. Control flows from block 925 to block 915, as well as asynchronously to block 933.

After confirmation that the parity has been updated at the hard disk storage device location, the storage manager updates parity status for the hard disk storage device location to consistent at block 933.

If the storage manager determines that the parity *m* is already cached at block 905, then control flows to block 907. At block 907, the storage manager reads the parity *m* from the caching SSD.

At block 909, the storage manager recalculates the parity based on the updated data block.

At block 911, the storage manager indicates parity status for the SSD location as inconsistent with the storage manager cache. Control flows to block 913 from block 911.

Figure 10 is a flowchart of example operations for parity eviction from a caching solid state storage device in a hybrid storage array with multiple parity protection. Figure 10 is similar to Figure 8 with a deviation to accommodate updating a hard disk storage device parity that is inconsistent with the SSD cached parity.

At block 1001, an eviction process begins an eviction evaluation for each parity of the hybrid storage array that is cached in a solid state storage device of the array. Although these operations are presented as iterating over each cached parity, the claims are not constrained to iterating over each cached parity as discussed above with reference to Figure 8. Similarly, the eviction process can be triggered with the various triggering criteria also discussed with respect to Figure 8.

At block 1003, the eviction process determines whether a parity eviction criterion is satisfied. Various parity eviction criteria are discussed above with respect to block 803 of Figure 8. If the parity eviction criterion is satisfied, then control flows to block 1005. If it is not, then control flows to block 1011.

At block 1006, the eviction process updates the hard disk storage device location for the parity in accordance with the SSD cached parity. The eviction process reads the parity value in the SSD cached location, determines the hard disk storage device location, and requests that the parity value be written to the hard disk storage device location.

At block 1007, the eviction process notifies the solid state storage device caching the parity to mark the parity as invalid after receiving confirmation that the hard disk storage device location has been updated. For example, the storage manager can send a TRIM command to the solid state storage device.

At block 1009, the eviction process indicates that the parity is no longer being SSD cached. For instance, the storage manager can clear or remove indication of the solid state storage device that was caching the parity. The storage manager can change a bit or flag that indicates whether the associated parity is SSD cached.

At block 1011, the eviction process determines whether there is additional cached parity to evaluate for possible eviction. If not, then control flows to block 1013. If there is an additional cached parity to evaluate for possible eviction, control flows back to block 1001.

At block 1013, the eviction process waits for the next parity eviction evaluation trigger.

Variations

Although the examples refer to maintenance of dirty bits, other techniques can be used to ensure validity and preserve consistency of parity (and data). As examples, consistency and validity of parity can be preserved with locking, atomic transactions, etc.

The examples often refer to a "storage manager." The storage manager is a construct used to refer to implementation of functionality for parity caching in a solid state storage device of a hybrid storage array. This construct is utilized since numerous implementations are possible. A storage manager may be a server, filer, controller, a particular component or components of machine (e.g., a particular circuit card enclosed in a housing with other circuit cards/boards), machine-executable program or programs (e.g., a storage operating system), firmware, a circuit card with circuitry configured and programmed with firmware for managing storage arrays, etc. The term is used to efficiently explain content of the disclosure. The storage manager can also be referred to as a managing storage node. Although the examples refer to operations being performed by a storage manager, different entities can perform different operations. For instance, a dedicated co-processor or application specific integrated circuit can calculate parity.

The examples in Figures 1-4 depict a hybrid storage array that is external to the storage manager. However, at least part of the array may be housed within the storage

manager itself. The storage manager can have circuitry to manage a storage array that includes storage devices inserted into bays of the storage manager, storage devices connected directly to ports of the storage manager, and storage devices separated from the storage manager by at least one network element.

The flowcharts are provided to aid in understanding the illustrations and are not to be used to limit scope of the claims. The flowcharts depict example operations that can vary within the scope of the claims. Additional operations may be performed; fewer operations may be performed; the operations may be performed in parallel; and the operations may be performed in a different order. For example, the operations depicted in blocks 511 and 513 can be performed in parallel or concurrently. With respect to Figures 7 and 9, a caching criterion is not necessary. Parity can be cached upon update and management of the cached parity rely on a clean-up or eviction criterion or criteria. In addition, the update to indicate inconsistency between a caching SSD and an HDD for a parity can occur after confirmation of a successful write to the caching SSD instead of at block 930 as depicted in Figure 9.

As will be appreciated, aspects of the disclosure may be embodied as a system, method or program code/instructions stored in one or more machine-readable media. Accordingly, aspects may take the form of hardware, software (including firmware, resident software, micro-code, etc.), or a combination of software and hardware aspects that may all generally be referred to herein as a “circuit,” “module” or “system.” The functionality presented as individual modules/units in the example illustrations can be organized differently in accordance with any one of platform (operating system and/or hardware), application ecosystem, interfaces, programmer preferences, programming language, administrator preferences, etc.

Any combination of one or more machine readable medium(s) may be utilized. The machine readable medium may be a machine readable signal medium or a machine readable storage medium. A machine readable storage medium may be, for example, but not limited to, a system, apparatus, or device, that employs any one of or combination of electronic, magnetic, optical, electromagnetic, infrared, or semiconductor technology to store program code. More specific examples (a non-exhaustive list) of the machine

readable storage medium would include the following: a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), a portable compact disc read-only memory (CD-ROM), an optical storage device, a magnetic storage device, or any suitable combination of the foregoing. In the context of this document, a machine readable storage medium may be any tangible medium that can contain, or store a program for use by or in connection with an instruction execution system, apparatus, or device. A machine readable storage medium is not a machine readable signal medium.

A machine readable signal medium may include a propagated data signal with machine readable program code embodied therein, for example, in baseband or as part of a carrier wave. Such a propagated signal may take any of a variety of forms, including, but not limited to, electro-magnetic, optical, or any suitable combination thereof. A machine readable signal medium may be any machine readable medium that is not a machine readable storage medium and that can communicate, propagate, or transport a program for use by or in connection with an instruction execution system, apparatus, or device.

Program code embodied on a machine readable medium may be transmitted using any appropriate medium, including but not limited to wireless, wireline, optical fiber cable, RF, etc., or any suitable combination of the foregoing.

Computer program code for carrying out operations for aspects of the disclosure may be written in any combination of one or more programming languages, including an object oriented programming language such as the Java® programming language, C++ or the like; a dynamic programming language such as Python; a scripting language such as Perl programming language or PowerShell script language; and conventional procedural programming languages, such as the "C" programming language or similar programming languages. The program code may execute entirely on a stand-alone machine, may execute in a distributed manner across multiple machines, and may execute on one machine while providing results and or accepting input on another machine.

Aspects of this disclosure are described with reference to flowchart illustrations and/or block diagrams. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by program code. The program code may be provided to a processor of a general purpose computer, special purpose computer, or other programmable machine or apparatus.

The program code/instructions may also be stored in a machine readable medium that can direct a machine to function in a particular manner, such that the instructions stored in the machine readable medium produce an article of manufacture including instructions which implement the function/act specified in the flowchart and/or block diagram block or blocks.

Figure 11 depicts an example computer system with a storage array controller that caches parity on a solid state storage device of a hybrid storage array. The computer system includes a processor unit 1101 (possibly including multiple processors, multiple cores, multiple nodes, and/or implementing multi-threading, etc.). The computer system includes memory 1107. The memory 1107 may be system memory (e.g., one or more of cache, SRAM, DRAM, zero capacitor RAM, Twin Transistor RAM, eDRAM, EDO RAM, DDR RAM, EEPROM, NRAM, RRAM, SONOS, PRAM, etc.) or any one or more of the above already described possible realizations of machine-readable media. The computer system also includes a bus 1103 (e.g., PCI, ISA, PCI-Express, HyperTransport® bus, InfiniBand® bus, NuBus, etc.) and a network interface 1105 (e.g., a Fiber Channel interface, an Ethernet interface, an internet small computer system interface, SONET interface, wireless interface, etc. The system also includes a storage array controller 1111. The storage array controller 1111 caches parities of a hybrid storage array in a constituent solid state storage device. The storage array controller also evicts cached parity when a parity eviction criterion is met. Any one of the previously described functionalities may be partially (or entirely) implemented in hardware and/or on the processing unit 1101. For example, the functionality may be implemented with an application specific integrated circuit, in logic implemented in the processing unit 1101, in a co-processor on a peripheral device or card, etc. Further, realizations may include fewer or additional components not illustrated in Figure 11 (e.g., video cards, audio cards, additional network interfaces,

peripheral devices, etc.). The processor unit 1101 and the network interface 1105 are coupled to the bus 1103. Although illustrated as being coupled to the bus 1103, the memory 1107 may be coupled to the processor unit 1101.

While the aspects of the disclosure are described with reference to various implementations and exploitations, it will be understood that these aspects are illustrative and that the scope of the claims is not limited to them. In general, techniques for caching parity and evicting parity in a hybrid storage array as described herein may be implemented with facilities consistent with any hardware system or hardware systems. Many variations, modifications, additions, and improvements are possible.

Plural instances may be provided for components, operations or structures described herein as a single instance. Finally, boundaries between various components, operations and data stores are somewhat arbitrary, and particular operations are illustrated in the context of specific illustrative configurations. Other allocations of functionality are envisioned and may fall within the scope of the disclosure. In general, structures and functionality presented as separate components in the example configurations may be implemented as a combined structure or component. Similarly, structures and functionality presented as a single component may be implemented as separate components. These and other variations, modifications, additions, and improvements may fall within the scope of the disclosure.

Terminology

This description uses the term "parity." Since this disclosure relates to storage arrays, it should be clear that parity refers to fault tolerance data that can be used to reconstruct data (e.g., user data or application data) after a storage device failure. However, the possibility exists that interpretations will be asserted that disregard this disclosure since parity can have other meanings. For at least this reason, the claims will also refer to reconstruction information since parity is used to reconstruct data on a failed storage device. In addition, the description refers to parity when parity value would be more appropriate. This is done for ease of explanation. Parity for a set/group/plurality of data

units can refer to any parity value that protects the data units. A parity value is a particular instance of that parity.

This description also uses the term "data blocks." This is a familiar term within the industry, but it is susceptible to variations in meaning. For this disclosure, a data block is a unit of data that is read from or written to a storage array. The claims will refer to data units to avoid any unintended interpretations or unnecessary definitions of data blocks based on a particular technology.

In relation to the above terms, the claims will not refer to stripes. Although this is a common term used when data striping is employed, the claims will simply refer to a groups or set of data units protected or corresponding to a parity.

WHAT IS CLAIMED IS:

1. A method comprising:

determining that a parity for a plurality of data units is stored on a first of a plurality of hard disk storage devices after detecting an update that impacts a first data unit of the plurality of data units, wherein a storage array at least includes the plurality of hard disk storage devices and a solid state storage device;

after reading a first value for the parity from the first hard disk storage device in the storage array, calculating a second value for the parity based, at least in part, on the first value and an updated first data unit;

indicating that the parity is stored in the solid state storage device; and

requesting that the second value be written to the solid state storage device.

2. The method of claim 1 further comprising:

determining that the parity is stored in the solid state storage device in response to detecting a second update that impacts a second data unit of the plurality of data units;

reading the second value from solid state storage device; and

requesting that a third value for the parity be written to the solid state storage device after calculating the third value based, at least in part, on the second value and the second update.

3. The method of claim 1 further comprising reading the first value from the first hard disk storage device after determining that the parity for the plurality of data units is stored in the first hard disk storage device.

4. The method of claim 1 further comprising maintaining a first indication of consistency for the parity in the first hard disk storage device with respect to a storage array manager cache and a second indication of consistency for the parity in the solid state storage device with respect to the storage array manager cache.

5. The method of claim 1 further comprising requesting that the second value also be written to the first hard disk storage device.

6. The method of claim 1 further comprising maintaining a first indication of consistency for the parity in the first hard disk storage device with respect to the parity in the solid state storage device.

7. The method of claim 6 further comprising maintaining a second indication of consistency for the parity in the solid state storage device with respect to a storage array manager cache.

8. The method of claim 6 further comprising:

determining that the parity has been selected for eviction from the solid state storage device; and

requesting that the second value be written to the first hard disk storage device for the parity in response to determining that the parity has been selected for eviction from the solid state storage device.

9. One or more machine-readable storage media having program code stored therein, the program code to:

determine that reconstruction information for a plurality of data units is stored on a first of a plurality of hard disk storage devices of a storage array after detection of an update that impacts a first data unit of the plurality of data units;

read a first value for the reconstruction information from the first hard disk storage device;

after calculation of a second value for the reconstruction information based, at least in part, on the first value and an updated first data unit, indicate that the reconstruction information is cached in a solid state storage device of the storage array; and

request that the second value be written to the solid state storage device.

10. The machine-readable storage media of claim 9, further comprising program code to:

determine that the reconstruction information is cached in the solid state storage device in response to detection of a second update that impacts a second data unit of the plurality of data units;

read the second value from solid state storage device; and

request that a third value for the reconstruction information be cached in the solid state storage device after calculation of the third value based, at least in part, on the second value and the second update.

11. The machine-readable storage media of claim 9 further comprising program code to read the first value from the first hard disk storage device after a determination that the reconstruction information for the plurality of data units is stored in the first hard disk storage device.

12. The machine-readable storage media of claim 9 further comprising program code to maintain a first indication of consistency for the reconstruction information in the first hard disk storage device with respect to a storage array manager cache and a second indication of consistency for the reconstruction information in the solid state storage device with respect to the storage array manager cache.

13. The machine-readable storage media of claim 9 further comprising program code to request that the second value also be written to the first hard disk storage device.

14. The machine-readable storage media of claim 9 further comprising program code to maintain a first indication of consistency for the reconstruction information in the first hard disk storage device with respect to the reconstruction information in the solid state storage device.

15. The machine-readable storage media of claim 14 further comprising program code to maintain a second indication of consistency for the reconstruction information in the solid state storage device with respect to a storage array manager cache.

16. The machine-readable storage media of claim 14 further comprising program code to:

determine that the reconstruction information has been selected for eviction from the solid state storage device; and

request that the second value be written to the first hard disk storage device for the reconstruction information in response to a determination that the reconstruction information has been selected for eviction from the solid state storage device.

17. An apparatus comprising:

a processor; and

a machine-readable medium having program code executable by the processor to cause the apparatus to,

determine that reconstruction information for a plurality of data units is stored on a first of a plurality of hard disk storage devices of a storage array after detection of an update that impacts a first data unit of the plurality of data units;

read a first value for the reconstruction information from the first hard disk storage device;

after calculation of a second value for the reconstruction information based, at least in part, on the first value and an updated first data unit, indicate that the reconstruction information is cached in a solid state storage device of the storage array; and

request that the second value be written to the solid state storage device.

18. The apparatus of claim 17, further comprising program code to cause the apparatus to:

determine that the reconstruction information is cached in the solid state storage device in response to detection of a second update that impacts a second data unit of the plurality of data units;

read the second value from solid state storage device; and

request that a third value for the reconstruction information be cached in the solid state storage device after calculation of the third value based, at least in part, on the second value and the second update.

19. The apparatus of claim 17 further comprising program code to cause the apparatus to read the first value from the first hard disk storage device after a determination that the reconstruction information for the plurality of data units is stored in the first hard disk storage device.

20. The apparatus of claim 17 further comprising program code to cause the apparatus to maintain a first indication of consistency for the reconstruction information in the first hard disk storage device with respect to a cache memory of the apparatus and a second indication of consistency for the reconstruction information in the solid state storage device with respect to the cache memory.

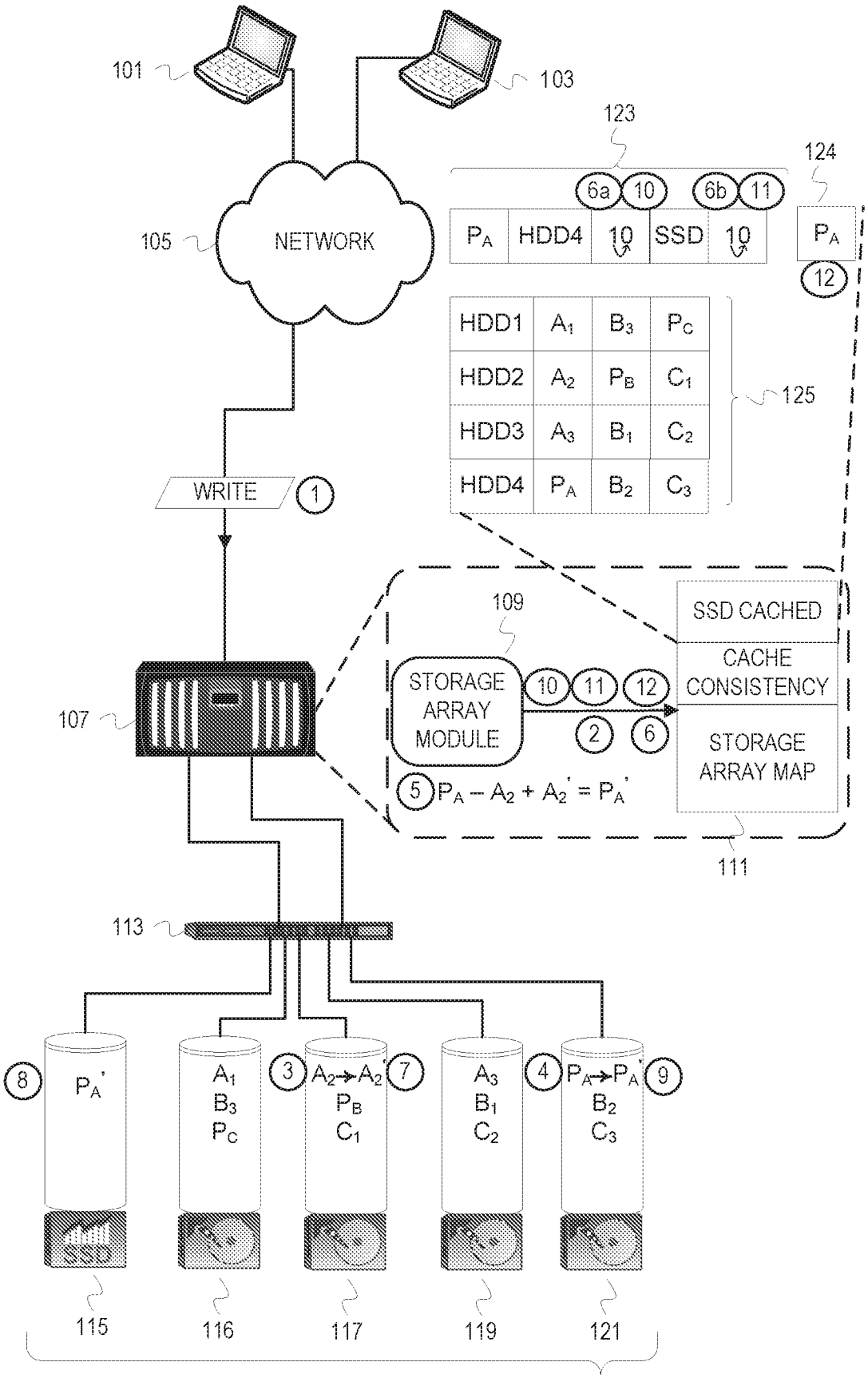

101
103
123
124
6a
10
6b
11
105
NETWORK
PA
HDD4
10
SSD
10
PA
12
HDD1
A1
B3
PC
HDD2
A2
PB
C1
HDD3
A3
B1
C2
HDD4
PA
B2
C3
125
WRITE
1
109
SSD CACHED
CACHE CONSISTENCY
STORAGE ARRAY MODULE
10
11
12
2
6
STORAGE ARRAY MAP
5 PA – A2 + A2' = PA'
107
111
113
8
PA'
A1
B3
PC
3
A2→A2'
7
PB
C1
A3
B1
C2
4
PA→PA'
9
B2
C3
SSD
115
116
117
119
121
HYBRID STORAGE ARRAY

FIG. 1

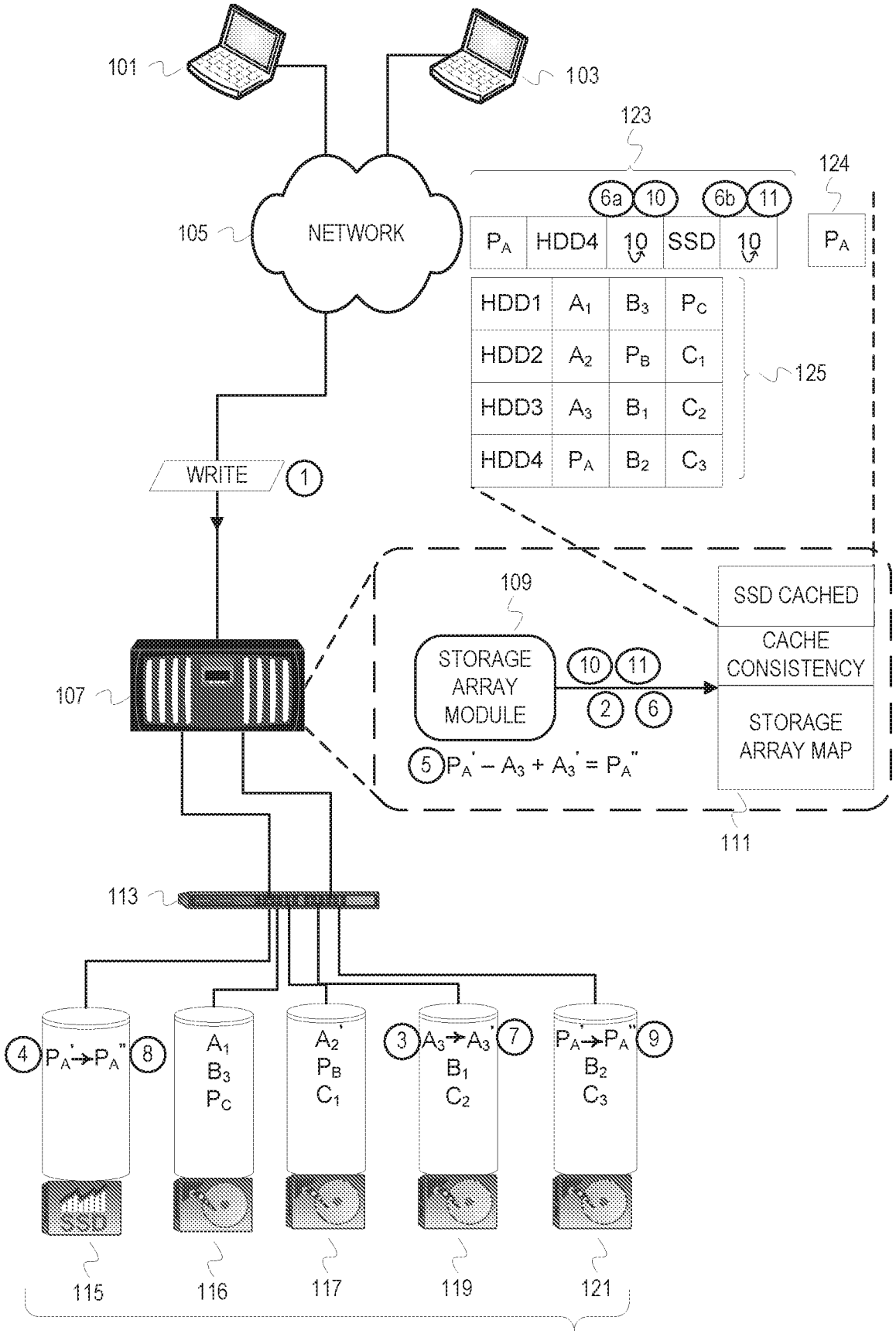

101
103
105
NETWORK
123
124
6a
10
6b
11
P_A HDD4 10 SSD 10
P_A
HDD1 A_1 B_3 P_C
HDD2 A_2 P_B C_1
HDD3 A_3 B_1 C_2
HDD4 P_A B_2 C_3
125
WRITE
1
109
STORAGE ARRAY MODULE
10
11
2
6
5 $P_A' - A_3 + A_3' = P_A''$
SSD CACHED
CACHE CONSISTENCY
STORAGE ARRAY MAP
111
107
113
4 $P_A' \rightarrow P_A''$ 8
A_1 B_3 P_C
A_2' P_B C_1
3 $A_3 \rightarrow A_3'$ 7
B_1 C_2
$P_A \rightarrow P_A''$ 9
B_2 C_3
SSD
115
116
117
119
121
HYBRID STORAGE ARRAY

FIG. 2

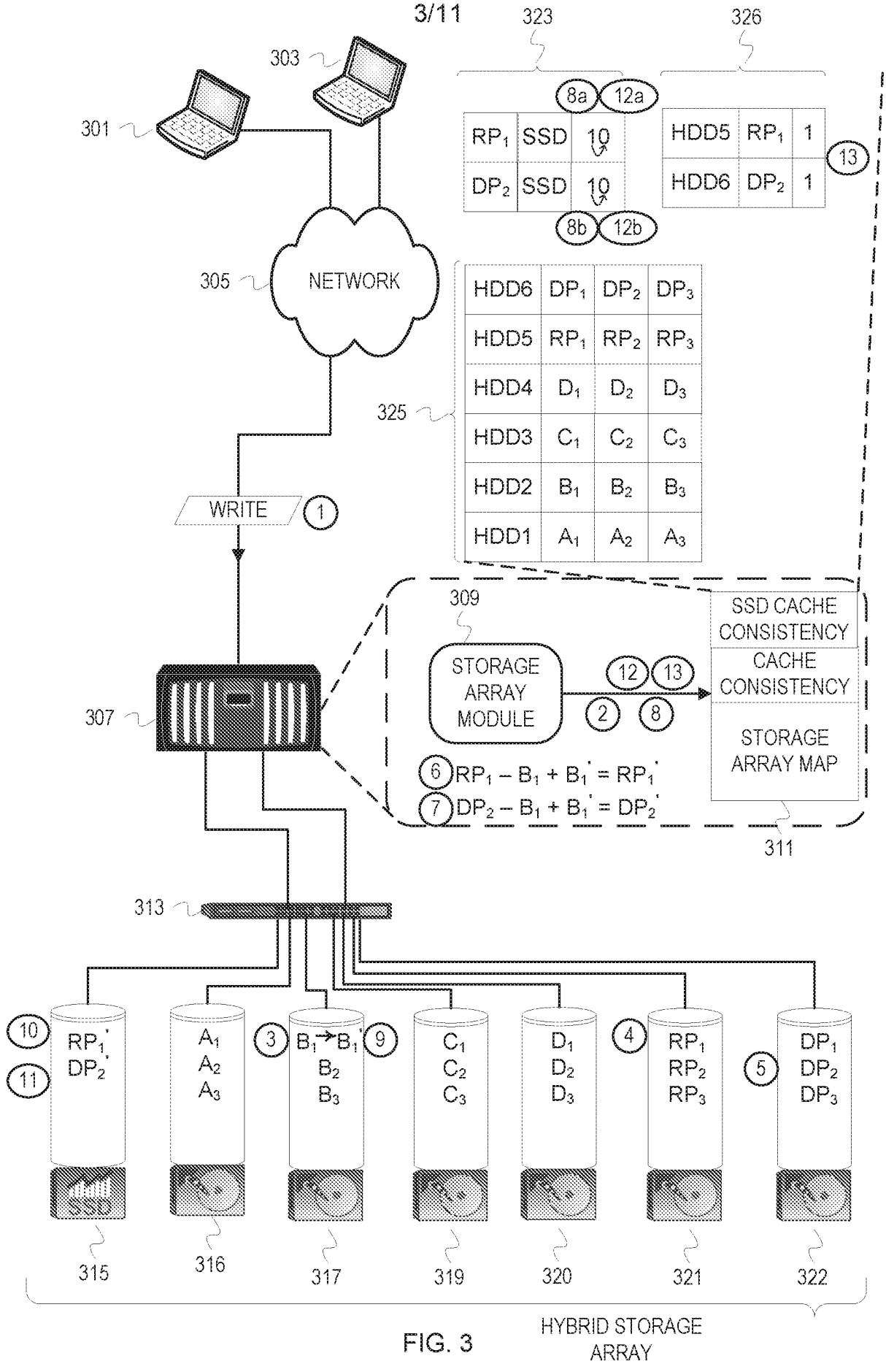

323
326
303
301
8a
12a
RP1 SSD 10
DP2 SSD 10
HDD5 RP1 1
HDD6 DP2 1
13
8b
12b
305
NETWORK
HDD6 DP1 DP2 DP3
HDD5 RP1 RP2 RP3
HDD4 D1 D2 D3
HDD3 C1 C2 C3
HDD2 B1 B2 B3
HDD1 A1 A2 A3
325
WRITE
1
309
SSD CACHE CONSISTENCY
STORAGE ARRAY MODULE
12
13
2
8
CACHE CONSISTENCY
STORAGE ARRAY MAP
307
6 RP1 – B1 + B1' = RP1'
7 DP2 – B1 + B1' = DP2'
311
313
10
11
RP1'
DP2'
A1
A2
A3
3
B1→B1'
B2
B3
9
C1
C2
C3
D1
D2
D3
4
RP1
RP2
RP3
5
DP1
DP2
DP3
SSD
315
316
317
319
320
321
322
HYBRID STORAGE ARRAY

FIG. 3

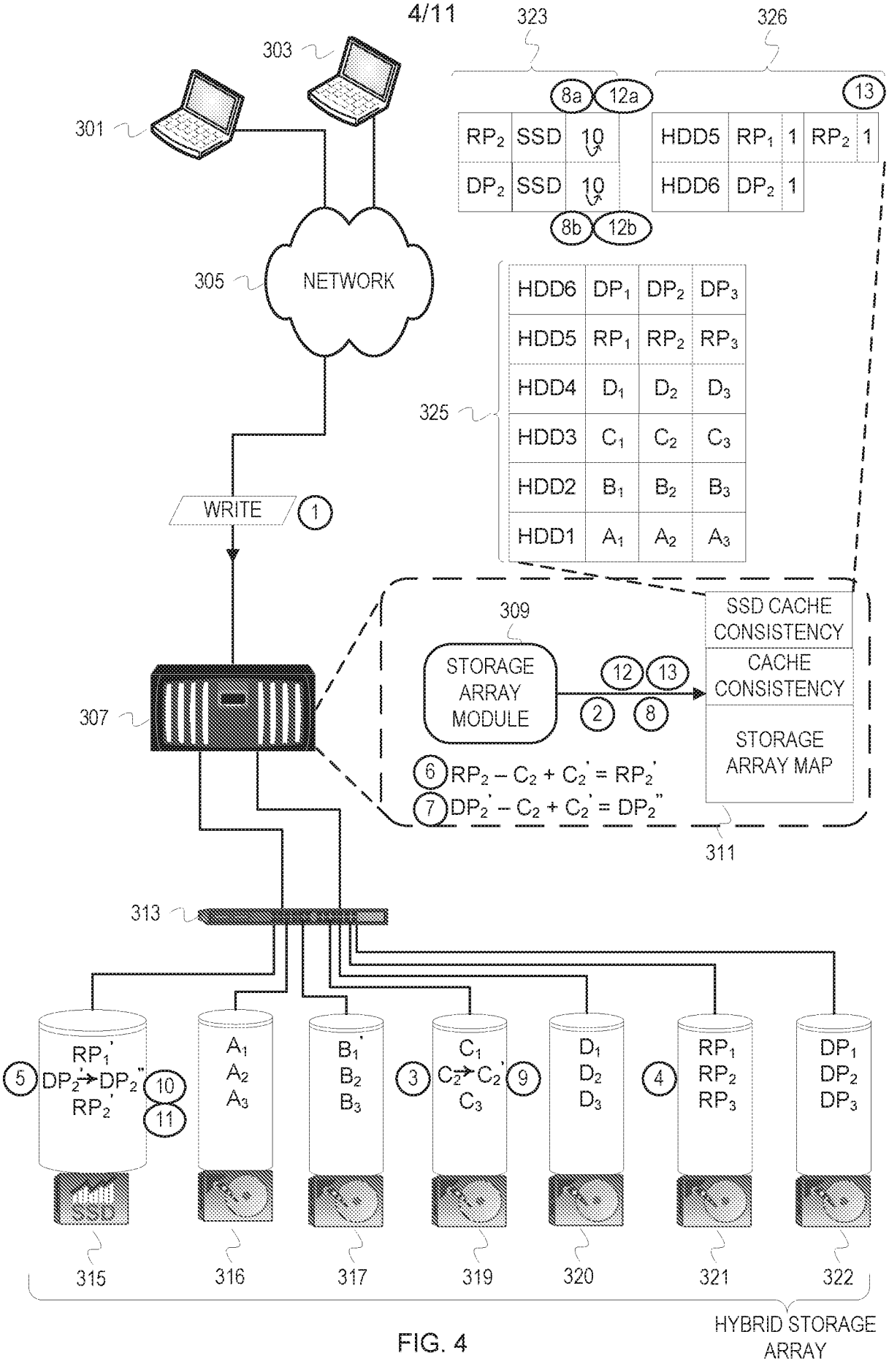

323
326
303
301
8a
12a
13
RP2 SSD 10
DP2 SSD 10
HDD5 RP1 1 RP2 1
HDD6 DP2 1
8b
12b
305
NETWORK
HDD6 DP1 DP2 DP3
HDD5 RP1 RP2 RP3
HDD4 D1 D2 D3
HDD3 C1 C2 C3
HDD2 B1 B2 B3
HDD1 A1 A2 A3
325
WRITE
1
309
SSD CACHE CONSISTENCY
STORAGE ARRAY MODULE
12
13
2
8
CACHE CONSISTENCY
STORAGE ARRAY MAP
307
6 RP2 – C2 + C2' = RP2'
7 DP2' – C2 + C2' = DP2"
311
313
RP1'
5 DP2'→DP2"
RP2'
10
11
A1
A2
A3
B1'
B2
B3
3
C1
C2→C2'
C3
9
D1
D2
D3
4
RP1
RP2
RP3
DP1
DP2
DP3
SSD
315
316
317
319
320
321
322
HYBRID STORAGE ARRAY

FIG. 4

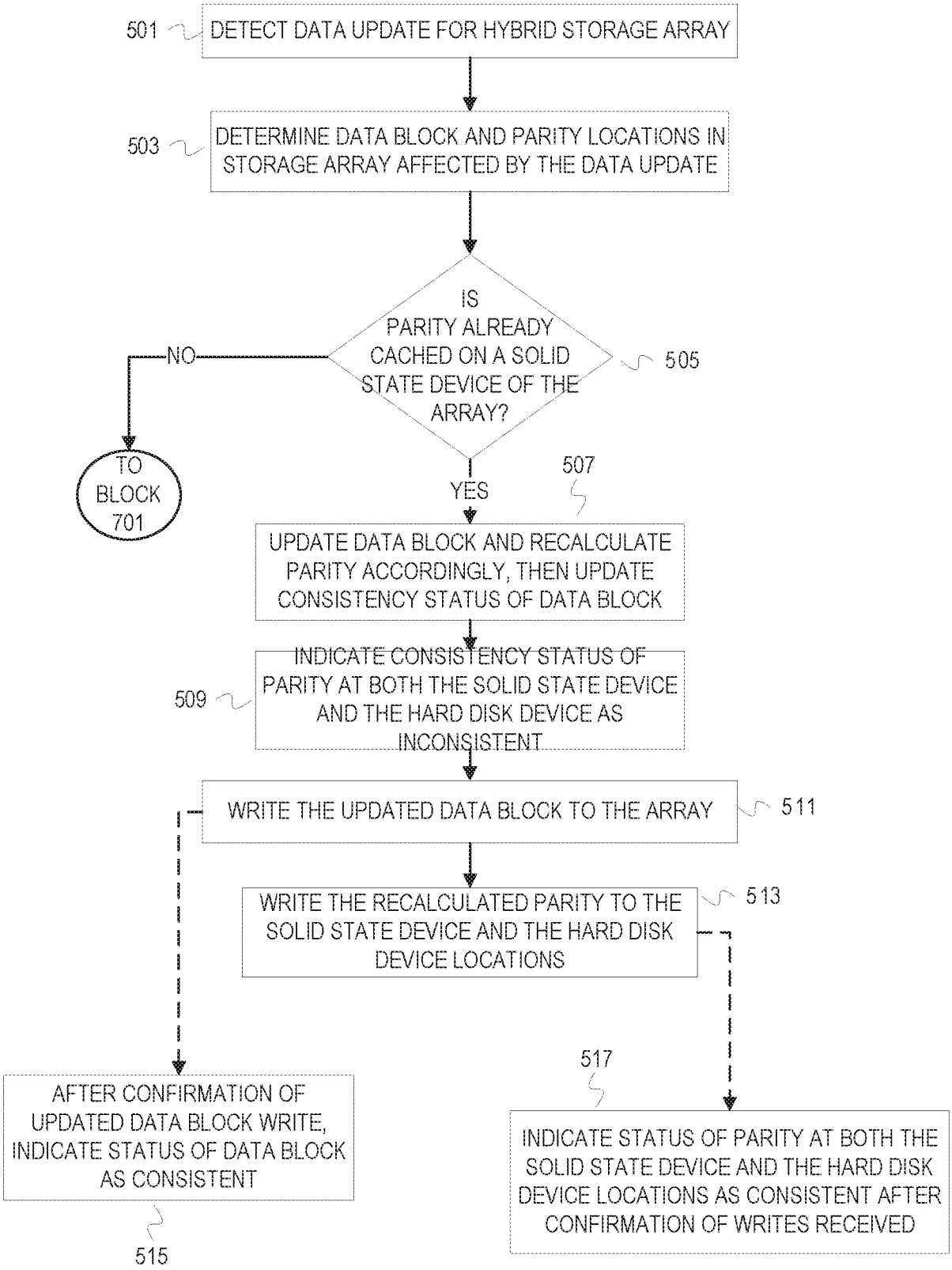

501
DETECT DATA UPDATE FOR HYBRID STORAGE ARRAY
503
DETERMINE DATA BLOCK AND PARITY LOCATIONS IN STORAGE ARRAY AFFECTED BY THE DATA UPDATE
IS PARITY ALREADY CACHED ON A SOLID STATE DEVICE OF THE ARRAY?
505
NO
TO BLOCK 701
YES
507
UPDATE DATA BLOCK AND RECALCULATE PARITY ACCORDINGLY, THEN UPDATE CONSISTENCY STATUS OF DATA BLOCK
509
INDICATE CONSISTENCY STATUS OF PARITY AT BOTH THE SOLID STATE DEVICE AND THE HARD DISK DEVICE AS INCONSISTENT
WRITE THE UPDATED DATA BLOCK TO THE ARRAY
511
WRITE THE RECALCULATED PARITY TO THE SOLID STATE DEVICE AND THE HARD DISK DEVICE LOCATIONS
513
AFTER CONFIRMATION OF UPDATED DATA BLOCK WRITE, INDICATE STATUS OF DATA BLOCK AS CONSISTENT
515
517
INDICATE STATUS OF PARITY AT BOTH THE SOLID STATE DEVICE AND THE HARD DISK DEVICE LOCATIONS AS CONSISTENT AFTER CONFIRMATION OF WRITES RECEIVED

FIG. 5

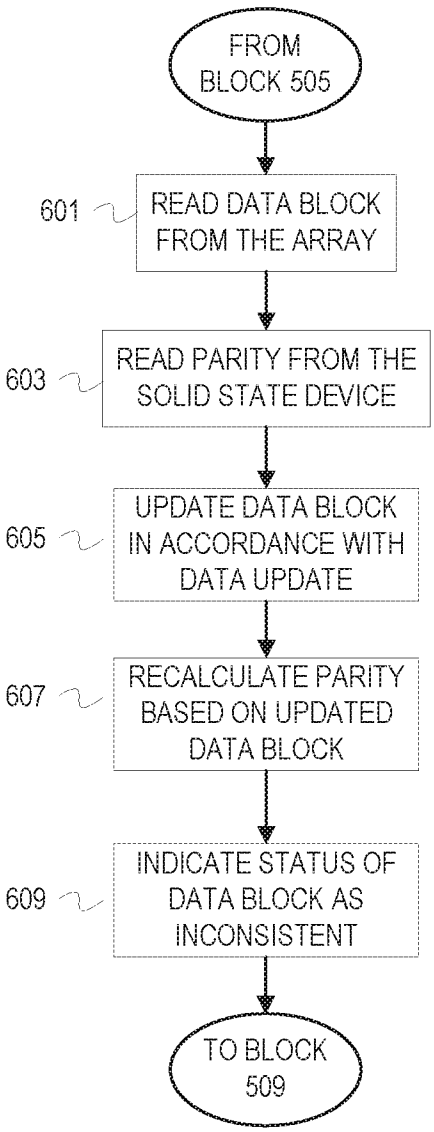

FROM BLOCK 505
601
READ DATA BLOCK FROM THE ARRAY
603
READ PARITY FROM THE SOLID STATE DEVICE
605
UPDATE DATA BLOCK IN ACCORDANCE WITH DATA UPDATE
607
RECALCULATE PARITY BASED ON UPDATED DATA BLOCK
609
INDICATE STATUS OF DATA BLOCK AS INCONSISTENT
TO BLOCK 509

FIG. 6

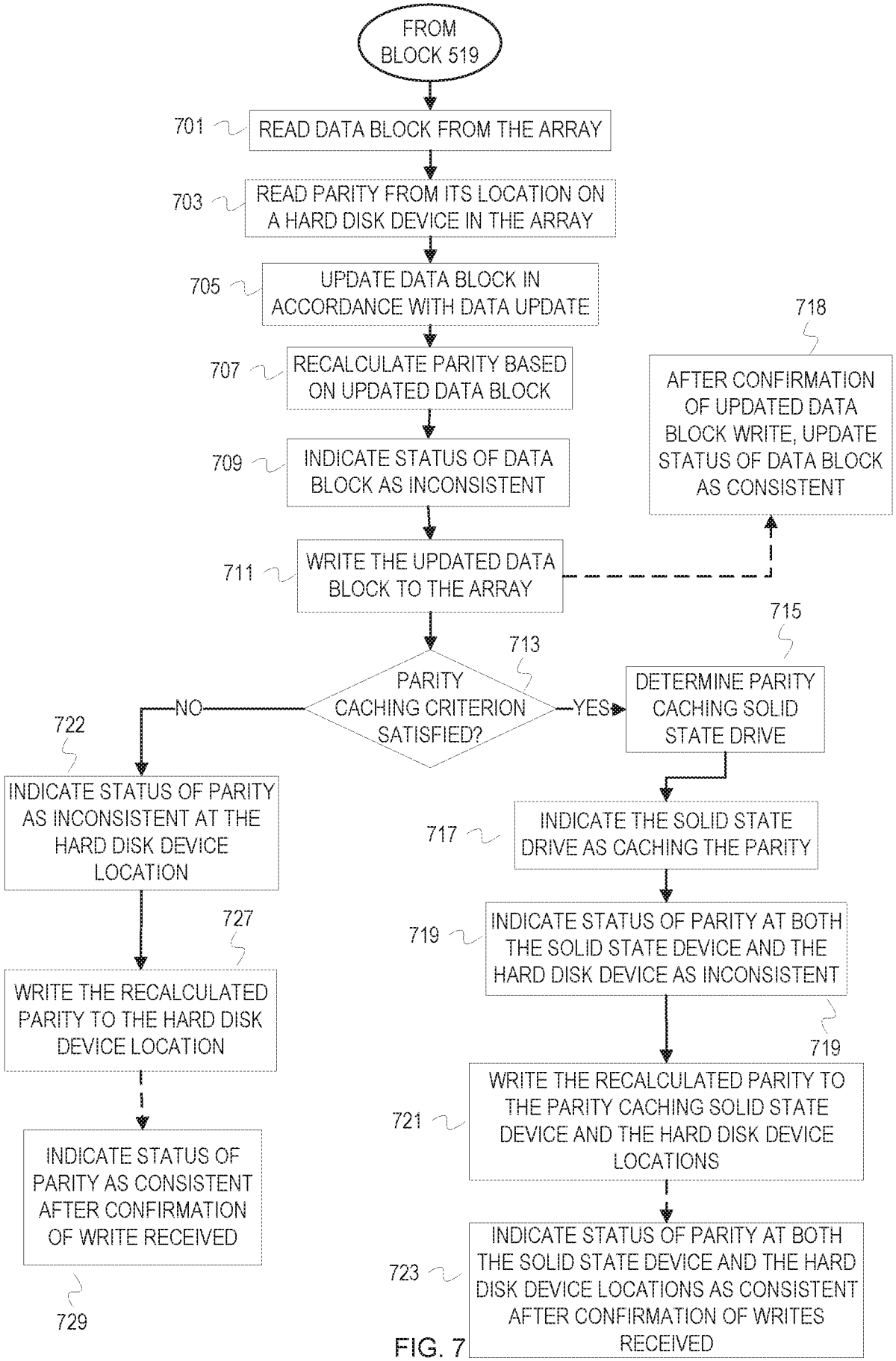

FROM BLOCK 519
701
READ DATA BLOCK FROM THE ARRAY
703
READ PARITY FROM ITS LOCATION ON A HARD DISK DEVICE IN THE ARRAY
705
UPDATE DATA BLOCK IN ACCORDANCE WITH DATA UPDATE
707
RECALCULATE PARITY BASED ON UPDATED DATA BLOCK
709
INDICATE STATUS OF DATA BLOCK AS INCONSISTENT
711
WRITE THE UPDATED DATA BLOCK TO THE ARRAY
718
AFTER CONFIRMATION OF UPDATED DATA BLOCK WRITE, UPDATE STATUS OF DATA BLOCK AS CONSISTENT
713
PARITY CACHING CRITERION SATISFIED?
NO
YES
715
DETERMINE PARITY CACHING SOLID STATE DRIVE
722
INDICATE STATUS OF PARITY AS INCONSISTENT AT THE HARD DISK DEVICE LOCATION
717
INDICATE THE SOLID STATE DRIVE AS CACHING THE PARITY
719
INDICATE STATUS OF PARITY AT BOTH THE SOLID STATE DEVICE AND THE HARD DISK DEVICE AS INCONSISTENT
727
WRITE THE RECALCULATED PARITY TO THE HARD DISK DEVICE LOCATION
719
721
WRITE THE RECALCULATED PARITY TO THE PARITY CACHING SOLID STATE DEVICE AND THE HARD DISK DEVICE LOCATIONS
INDICATE STATUS OF PARITY AS CONSISTENT AFTER CONFIRMATION OF WRITE RECEIVED
723
INDICATE STATUS OF PARITY AT BOTH THE SOLID STATE DEVICE AND THE HARD DISK DEVICE LOCATIONS AS CONSISTENT AFTER CONFIRMATION OF WRITES RECEIVED
729

FIG. 7

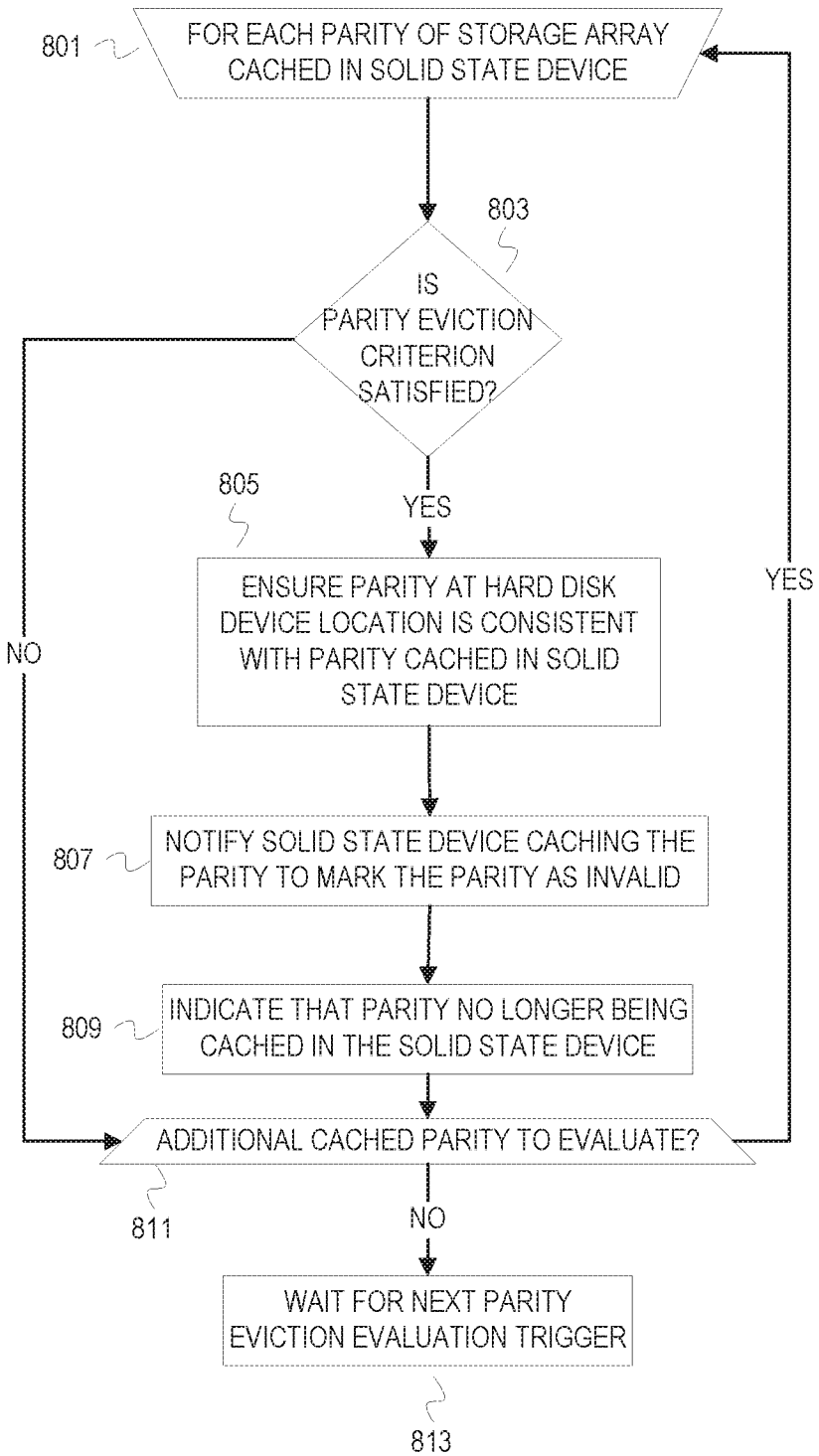

801
FOR EACH PARITY OF STORAGE ARRAY CACHED IN SOLID STATE DEVICE
803
IS PARITY EVICTION CRITERION SATISFIED?
NO
YES
805
ENSURE PARITY AT HARD DISK DEVICE LOCATION IS CONSISTENT WITH PARITY CACHED IN SOLID STATE DEVICE
807
NOTIFY SOLID STATE DEVICE CACHING THE PARITY TO MARK THE PARITY AS INVALID
809
INDICATE THAT PARITY NO LONGER BEING CACHED IN THE SOLID STATE DEVICE
ADDITIONAL CACHED PARITY TO EVALUATE?
YES
811
NO
WAIT FOR NEXT PARITY EVICTION EVALUATION TRIGGER
813

FIG. 8

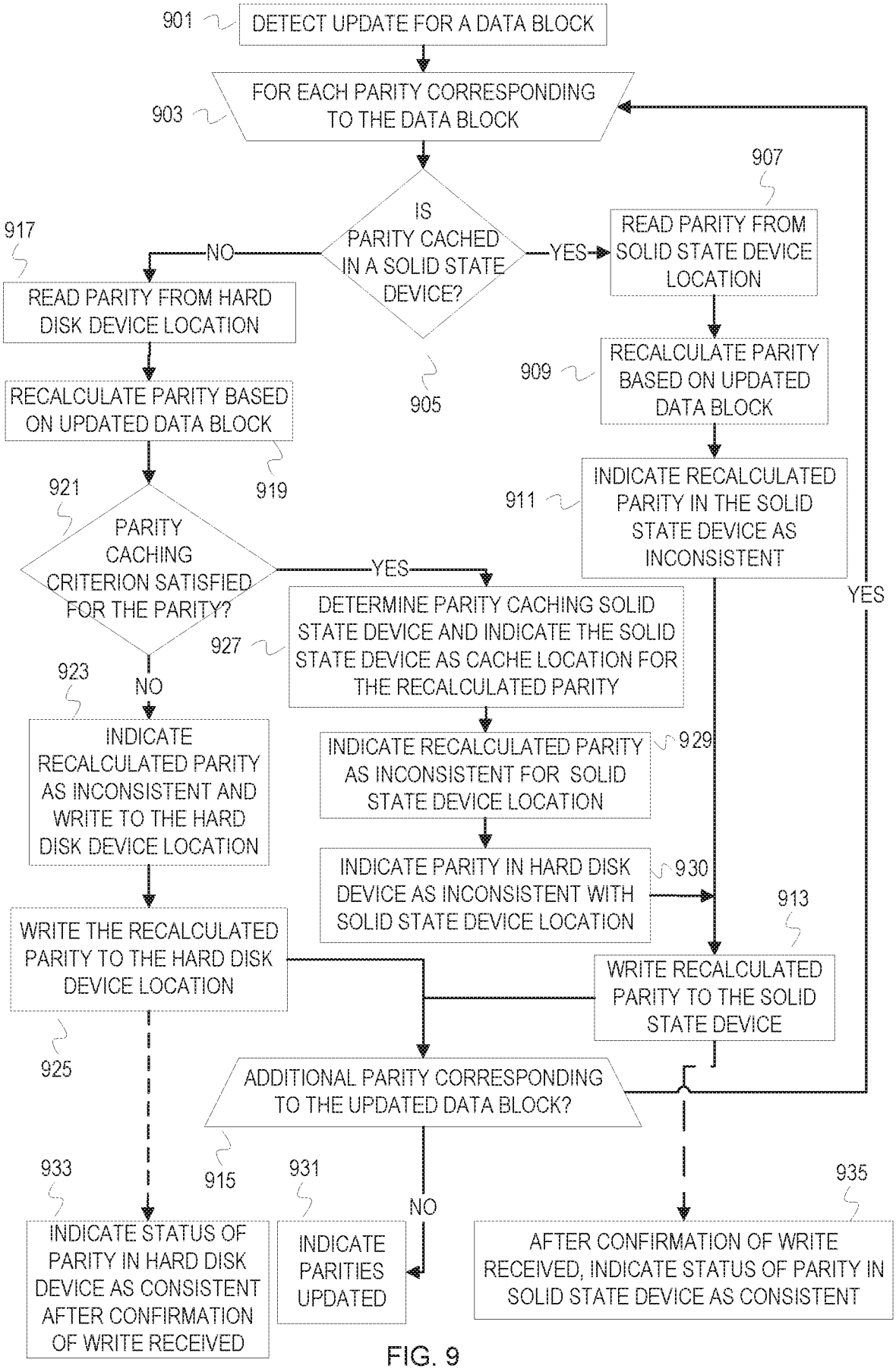

901
DETECT UPDATE FOR A DATA BLOCK
903
FOR EACH PARITY CORRESPONDING TO THE DATA BLOCK
IS PARITY CACHED IN A SOLID STATE DEVICE?
905
NO
YES
907
READ PARITY FROM SOLID STATE DEVICE LOCATION
909
RECALCULATE PARITY BASED ON UPDATED DATA BLOCK
911
INDICATE RECALCULATED PARITY IN THE SOLID STATE DEVICE AS INCONSISTENT
913
WRITE RECALCULATED PARITY TO THE SOLID STATE DEVICE
917
READ PARITY FROM HARD DISK DEVICE LOCATION
919
RECALCULATE PARITY BASED ON UPDATED DATA BLOCK
921
PARITY CACHING CRITERION SATISFIED FOR THE PARITY?
YES
NO
923
INDICATE RECALCULATED PARITY AS INCONSISTENT AND WRITE TO THE HARD DISK DEVICE LOCATION
925
WRITE THE RECALCULATED PARITY TO THE HARD DISK DEVICE LOCATION
927
DETERMINE PARITY CACHING SOLID STATE DEVICE AND INDICATE THE SOLID STATE DEVICE AS CACHE LOCATION FOR THE RECALCULATED PARITY
929
INDICATE RECALCULATED PARITY AS INCONSISTENT FOR SOLID STATE DEVICE LOCATION
930
INDICATE PARITY IN HARD DISK DEVICE AS INCONSISTENT WITH SOLID STATE DEVICE LOCATION
915
ADDITIONAL PARITY CORRESPONDING TO THE UPDATED DATA BLOCK?
YES
NO
931
INDICATE PARITIES UPDATED
933
INDICATE STATUS OF PARITY IN HARD DISK DEVICE AS CONSISTENT AFTER CONFIRMATION OF WRITE RECEIVED
935
AFTER CONFIRMATION OF WRITE RECEIVED, INDICATE STATUS OF PARITY IN SOLID STATE DEVICE AS CONSISTENT

FIG. 9

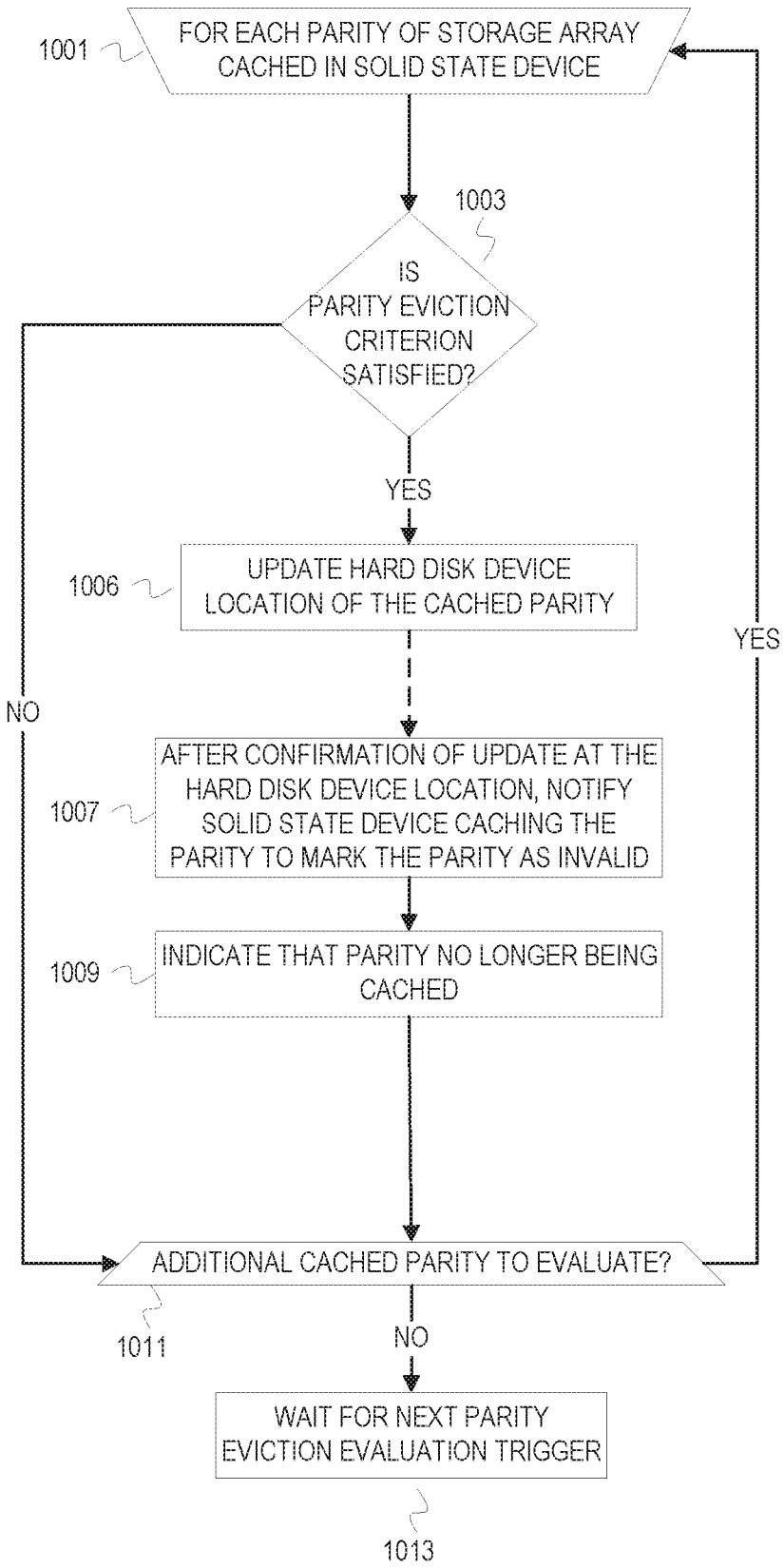

1001
FOR EACH PARITY OF STORAGE ARRAY CACHED IN SOLID STATE DEVICE
1003
IS PARITY EVICTION CRITERION SATISFIED?
YES
NO
1006
UPDATE HARD DISK DEVICE LOCATION OF THE CACHED PARITY
1007
AFTER CONFIRMATION OF UPDATE AT THE HARD DISK DEVICE LOCATION, NOTIFY SOLID STATE DEVICE CACHING THE PARITY TO MARK THE PARITY AS INVALID
1009
INDICATE THAT PARITY NO LONGER BEING CACHED
ADDITIONAL CACHED PARITY TO EVALUATE?
YES
1011
NO
WAIT FOR NEXT PARITY EVICTION EVALUATION TRIGGER
1013

FIG. 10

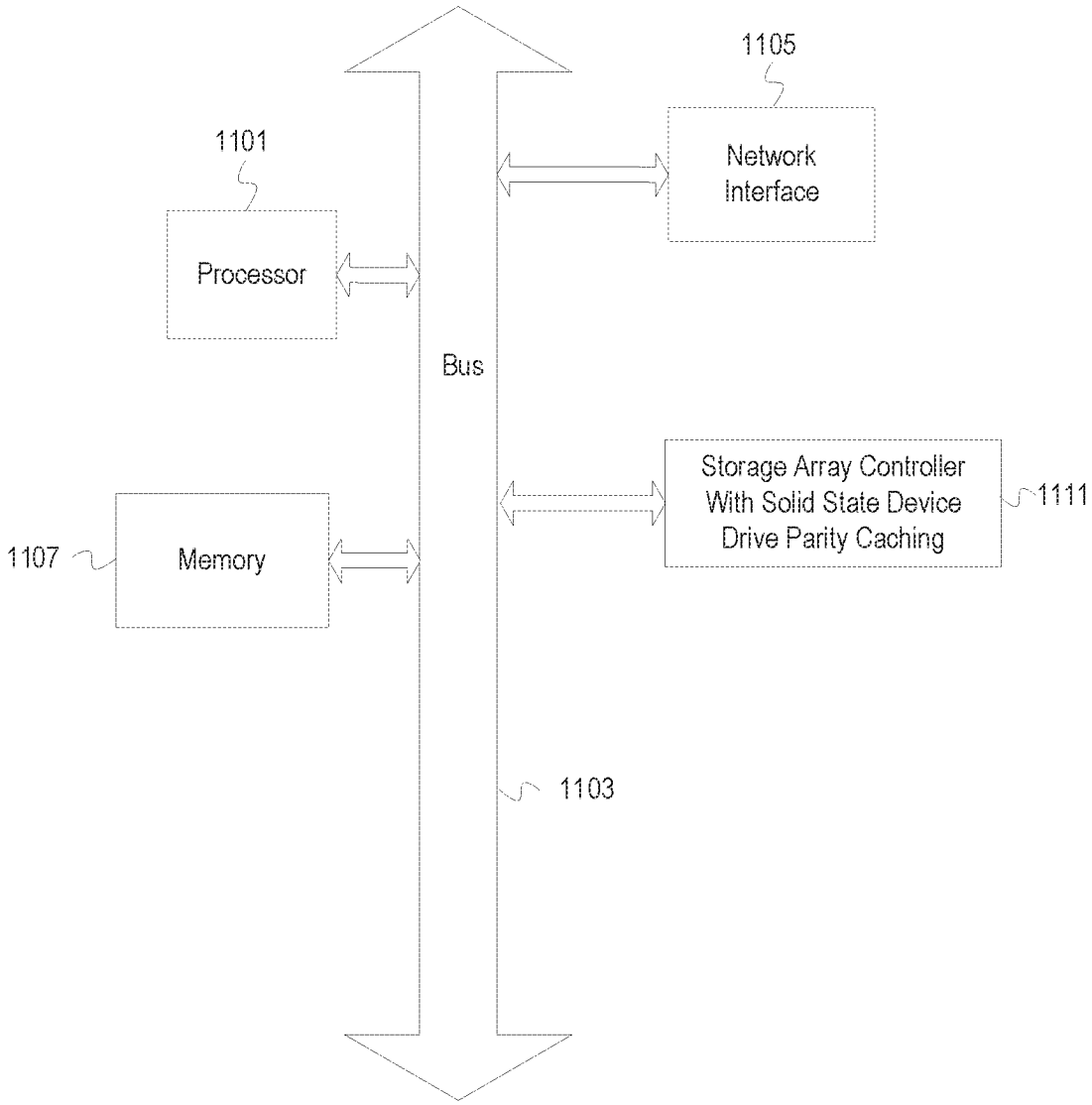

1105
Network Interface
1101
Processor
Bus
Storage Array Controller With Solid State Device Drive Parity Caching
1111
1107
Memory
1103

FIG. 11

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US16/18751

A. CLASSIFICATION OF SUBJECT MATTER
IPC(8) - G06F 3/06, 11/10, 12/02 (2016.01)
CPC - G06F 3/0685, 3/0689, 11/108
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC(8) - G06F 3/06, 11/10, 12/02 (2016.01)
CPC - G06F 3/0685, 3/061, 3/0658, 3/0659, 3/0688, 3/0689, 11/108, 12/0246, 12/0866

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

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
PatSeer (US, EP, WO, JP, DE, GB, CN, FR, KR, ES, AU, IN, CA, Other Countries (INPADOC), RU, AT, CH, TH, BR, PH); Google/Google Scholar; IEEE/IEEEXplore; EBSCO Non-Patent Prior Art Source
KEYWORDS: parity, hybrid, storage array, SSD-HDD, evict

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X --- Y	US 2011/0238885 A1 (KITAHARA, J., et al.) 29 September 2011; abstract; figures 8B, 9-11; paragraphs [0028], [0062], [0064], [0080]-[0081]	1, 3-7, 9, 11-15, 17, 19, 20 --- 2, 8, 10, 16, 18
Y	US 8,839,028 B1 (POLIA, M., et al/) 16 September 2014; abstract; claim 13; figures 1A, 2; column 5, lines 42-45, column 8, lines 29-35	2, 10, 18
Y	US 8,549,222 B1 (KLEIMAN, S., et al.) 01 October 2013; abstract; claim 13; figure 3; paragraphs [0005], [0037], [0040], [0047]	8, 16
A	US 2015/0032921 A1 (SILICON GRAPHICS INTERNATIONAL CORP.) 29 January 2015; entire document	1-20
A	MAO, B., et al. "HPDA: A hybrid parity-based disk array for enhanced performance and reliability"; ACM Transactions on Storage (TOS), Vol. 8, No. 1; Publication [online]. February 2012 [retrieved 04 April 2016]. Retrieved from the Internet: <URL: http://or.nsfc.gov.cn/bitstream/00001903-5/90177/1/1000003549834.pdf >; pp. 4.1-4.20	1-20

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

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

Date of the actual completion of the international search	Date of mailing of the international search report
13 April 2016 (13.4.2016)	09 JUN 2016
Name and mailing address of the ISA/ Mail Stop PCT, Attn: ISA/US, Commissioner for Patents P.O. Box 1450, Alexandria, Virginia 22313-1450 Facsimile No. 571-273-8300	Authorized officer Shane Thomas PCT Helpdesk: 571-272-4300 PCT OSP: 571-272-7774

Form PCT/ISA/210 (second sheet) (January 2015)