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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 February 2016 (04.02.2016)

WIPO | PCT

(10) International Publication Number
WO 2016/018383 A1

(51) International Patent Classification:
G06F 11/10 (2006.01)

(21) International Application Number:
PCT/US2014/049193

(22) International Filing Date:
31 July 2014 (31.07.2014)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **HEWLETT-PACKARD DEVELOPMENT COMPANY** [US/US]; 11445 Compaq Center Drive W., Houston, Texas 77070 (US).

(72) Inventors: **RAY, Harvey**; 3404 E Harmony Rd., Ft. Collins, Colorado 80528-9544 (US). **SHERLOCK, Derek Alan**; 3404 E Harmony Rd., Ft. Collins, Colorado 80528-9544 (US). **LESARTRE, Gregg B.**; 3404 E Harmony Rd., Ft. Collins, Colorado 80528-9544 (US).

(74) Agents: **SHOOKMAN, Jeb A.** et al.; Hewlett-Packard Company, Intellectual Property Administration, 3404 E. Harmony Road, Mail Stop 35, Fort Collins, Colorado 80528 (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, LT, 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, 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, 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 the identity of the inventor (Rule 4.17(i))*

[Continued on next page]

(54) Title: LIVE MIGRATION OF DATA

(57) Abstract: According to an example, hierarchal stripe locks may be obtained for a source stripe and a destination stripe. In response to receiving data for the source stripe, the data is written from the source stripe to the destination stripe, and the hierarchal stripe locks are released for the source stripe and the destination stripe. In response to receiving the data-migrated token, the hierarchal stripe locks are released for the source stripe and the destination stripe.

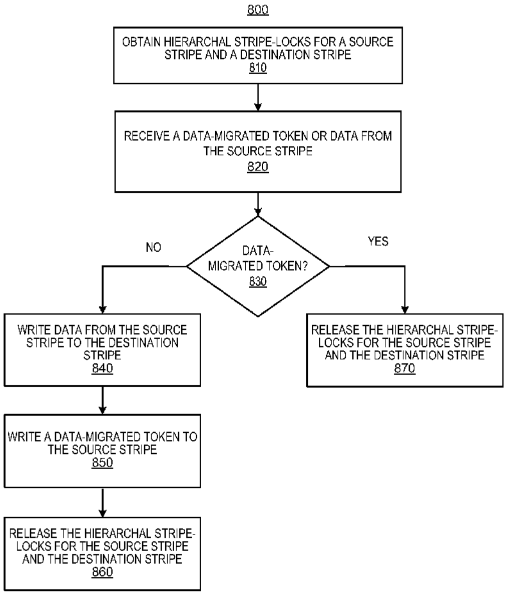
800
OBTAIN HIERARCHAL STRIPE-LOCKS FOR A SOURCE STRIPE AND A DESTINATION STRIPE 810
RECEIVE A DATA-MIGRATED TOKEN OR DATA FROM THE SOURCE STRIPE 820
DATA-MIGRATED TOKEN? 830
NO
YES
WRITE DATA FROM THE SOURCE STRIPE TO THE DESTINATION STRIPE 840
RELEASE THE HIERARCHAL STRIPE-LOCKS FOR THE SOURCE STRIPE AND THE DESTINATION STRIPE 870
WRITE A DATA-MIGRATED TOKEN TO THE SOURCE STRIPE 850
RELEASE THE HIERARCHAL STRIPE-LOCKS FOR THE SOURCE STRIPE AND THE DESTINATION STRIPE 860

FIG. 8

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

Published:

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

LIVE MIGRATION OF DATA

BACKGROUND

[0001] Current data storage devices such as volatile and non-volatile memory often include a fault tolerance to ensure that data is not lost in the event of a device error or failure. An example of a fault tolerance provided to current data storage devices involves storing data with redundancy. Examples of redundant data storage methods include duplicating data and storing the data in multiple locations and adding parity bits to store calculated error recovery bits. A consistency between the data and its redundancy should be maintained during a migration to another memory location to prevent data corruption and preserve fault tolerance.

BRIEF DESCRIPTION OF THE DRAWINGS

[0002] Features of the present disclosure are illustrated by way of example and not limited in the following figure(s), in which like numerals indicate like elements, in which:

[0003] FIG. 1 shows a simplified block diagram of a computing device for a live migration of data, according to an example of the present disclosure;

[0004] FIG. 2A shows striping across multiple memory modules in a fault tolerant scheme, according to an example of the present disclosure;

[0005] FIG. 2B shows a storage subsystem for a live migration of data from a source stripe to a destination stripe according to an example of the present disclosure;

[0006] FIG. 3 shows a bounce diagram of a live migration method using hierarchal stripe-locks, according to an example of the present disclosure;

[0007] FIG. 4 shows a state diagram of a method for staging an entry into and an exit from a live migration of data, according to an example of the present disclosure;

[0008] FIG. 5 shows a bounce diagram of an armed pre-migration read method, according to another example of the present disclosure;

[0009] FIG. 6 shows a bounce diagram of an armed post-migration read method, according to an example of the present disclosure;

[0010] FIG. 7 shows a bounce diagram of an armed write method, according to an example of the present disclosure; and

[0011] FIG. 8 shows a flow diagram of a live migration method using hierarchal stripe-locks, according to an example of the present disclosure.

DETAILED DESCRIPTION

[0012] For simplicity and illustrative purposes, the present disclosure is described by referring mainly to an example thereof. In the following description, numerous specific details are set forth in order to provide a thorough understanding of the present disclosure. It will be readily apparent however, that the present disclosure may be practiced without limitation to these specific details. In other instances, some methods and structures have not been described in detail so as not to unnecessarily obscure the present disclosure. As used herein, the terms “a” and “an” are intended to denote at least one of a particular element, the term “includes” means includes but not limited to, the term “including” means including but not limited to, and the term “based on” means based at least in part on.

[0013] Disclosed herein are examples of a method for live migration of data from a source stripe to a destination stripe. Live migration refers to the process of moving data from one memory module to another memory module without impacting the availability of the memory modules. A stripe may include a combination of data blocks from at least one data memory module and at least one parity memory module. The disclosed method, for instance, enables live migration of memory without intervention by an operating system. The memory may be concurrently accessed by one or more processors (e.g., servers) while parity-data consistency is maintained for each stripe affected by the migration. Also disclosed herein are a computing device and hardware system for implementing the methods.

[0014] According to an example, hierarchal stripe locks may be obtained for a source stripe and a destination stripe. One of the hierarchal stripe locks may be determined as a primary stripe lock and the other hierarchal stripe lock may be determined as a secondary stripe lock. The primary stripe lock, for instance, may be obtained prior to the secondary stripe lock to avoid a deadlock. Either valid data or a data-migrated token may then be received from the source stripe. In response to receiving valid data, data from the source stripe may be written to the destination stripe, a data-migrated token may be written to the source stripe, a parity may be updated for the source stripe and the destination stripe, and the hierarchal stripe locks may be released for the source stripe and the destination stripe. In response to receiving the data-migrated token, the hierarchal stripe locks may be released for the source stripe and the destination stripe. A data-migrated token may be communicated and stored in lieu of data in a source buffer data cache line.

[0015] According to an example, one of the hierarchal stripe locks may be determined as a primary stripe lock and the other hierarchal stripe lock may be determined as a secondary stripe lock. The ordering of one of the hierarchal stripe locks is based on stripe numbers and identification numbers of media controllers for the source stripe and the destination stripe in according to one example. In this regard, a lower identification number may be sequenced as the primary stripe lock and the higher identification number may be sequenced as the secondary stripe lock. Alternatively, a higher identification number may be sequenced as the primary stripe lock and the lower identification number may be sequenced as the secondary stripe lock.

[0016] An entry into the live migration of the data may include two stages. A first stage may include arming reads for each distributed redundancy controller. The armed reads for each distributed redundancy controller are to redirect each read to the destination stripe when the data-migrated token is received. A second stage may include arming writes for each of the distributed redundancy controllers after completion of the first stage.

[0017] An armed write for each of the distributed redundancy controllers is to obtain the hierarchal stripe locks for the source stripe and the destination stripe, write the data to the destination stripe, write the data-migrated token to the source stripe, update a parity for the source stripe and the destination stripe, and release the hierarchal stripe locks for the source stripe and the destination stripe. According to an example, the live migration may only occur after all the media controllers are armed for reads in the first stage and armed for writes in the second stage. An exit from the live migration of the data may include a disarming stage. The disarming stage may include determining that the data-migrated token is received for all source stripes in a source memory buffer and updating each of the distributed redundancy controllers to only access the destination stripes in a destination memory buffer.

[0018] The benefits of volatile memory (e.g., fast, low-latency) and non-volatile memory (e.g., persistent) are being combined in emerging memory technologies. Consequently, memory may require increased fault tolerance to protect persistent user data and file systems. In-memory and memory mapped storage paradigms may require a fault tolerance scheme such as redundant array of independent disks (RAID) to provide a robust fault-tolerant solution. The implementation of RAID in load/store computer architectures requires that, in addition to memory and cache coherency, parity-data consistency must be maintained to prevent data-corruption and provide fault tolerance. Furthermore, processors require relatively low latency load/store access to coherent memory. As such, memory migration must occur concurrently with load/store accesses from processors while maintaining parity-data consistency. Memory migration may be used to move data from a failing memory module to a stable memory module, move data off of a lightly used memory module to repurpose the memory module, defragment in-memory file systems to reclaim free space, alter RAID memory groups, and etc.

[0019] Generally speaking, the disclosed examples provide a live memory migration of that maintains parity-data consistency and is invisible to an operating system. The disclosed examples coordinate the entry into and exit from a migration mode to avoid race conditions and deadlock from distributed redundancy controllers. The disclosed examples may move RAIDED memory from one memory module to another module, move RAIDED contents from a failing memory module to another memory module, move memory contents from one region of RAIDED memory to another region, consolidate two RAID memory groups, and reconfigure a RAID memory group. Additionally, the disclosed examples may provide migration overlap between a source region and a destination region. That is, for instance, the source region and the destination region may be almost identical, except that one storage module within the RAID set is being migrated.

[0020] With reference to FIG. 1, there is shown a block diagram of a computing device 100 that may implemented as a platform for the live migration of data from a source stripe to a destination stripe according to an example of the present disclosure. It should be understood that the computing device 100 may include additional components and that one or more of the components described herein may be removed and/or modified without departing from a scope of the computing device 100.

[0021] For example, the computing device 100 may include a processor 102, memory modules 104A-N (where N is an integer greater than 1), an input/output interface 106, private memory 108, and a redundancy controller 110 (e.g., a RAID controller). In one example, the computing device 100 is a server but other types of computing devices may be used. The computing device 100 may be a node of a distributed data storage system. For example, the computing device 100 may be part of a cluster of nodes that services queries and provide data storage for multiple users or systems, and the nodes may communicate with each other to service queries and store data. The cluster of nodes may provide data redundancy to minimize down time and prevent data loss in case of a node failure. Also, the components of the computing device 100 are shown in a single computer as an example, however, in other examples the components may exist on multiple computing devices and the components may comprise multiple processors, redundancy controllers, memory modules, interfaces, etc.

[0022] The processor 102 may be a microprocessor, a micro-controller, an application specific integrated circuit (ASIC), field programmable gate array (FPGA), or other type of circuit to perform various processing functions. The memory modules 104A-N may each include media controllers 120A-N and memory 121A-N. According to an example, each memory module may include its own media controller and memory. The media controller, for instance, may communicate with the memory and control access to the memory by the processor 102. The private memory 108 and memory 121A-N may include volatile dynamic random access memory (DRAM) with battery backup, non-volatile phase change random access memory (PCRAM), spin transfer torque-magnetoresistive random access memory (STT-MRAM), resistive random access memory (reRAM), memristor, FLASH, or other types of memory devices. For example, the memory may be solid state, persistent, dense, fast memory. Fast memory can be memory having an access time similar to DRAM memory.

[0023] The input/output (I/O) interface 106 may include a hardware and/or a software interface. The I/O interface 106 may be a network interface connected to a network, such as the Internet, a local area network, etc. The computing system 100 may receive data and user-input through the I/O interface 106. The computing system 100 may be connected to the memory modules 104A-N, which may be provided on the computing device 100 or on a separate computing device.

[0024] The components of computing device 100 may be coupled by a bus 105, where the bus 105 may be a communication system that transfers data between the various components of the computing device 100. In examples, the bus 105 may be a Peripheral Component Interconnect (PCI), Industry Standard Architecture (ISA), PCI-Express, HyperTransport®, NuBus, a proprietary bus, and the like.

[0025] The redundancy controller 110 may maintain fault tolerance across the memory modules 104A-N according to an example. The redundancy controller 110 is depicted as including a migration engine 112, a stripe locking module 114, and a read/write module 116. Blocks 112, 114, and 116 are shown to illustrate the functionality of the redundancy controller 110. However, the functionality is implemented by hardware. The migration engine 112 for example is hardware of the redundancy controller 110 and the modules 114 and 116 for example are hardware of the redundancy controller 110, and the engine 112 and the modules 114 and 116 may not be machine readable instructions executed by a general purpose computer. The migration engine 112, for example, may initiate a migration mode and iterate through an entire region of memory whose contents are to be migrated from a source memory region to a destination memory region during the migration mode. The stripe locking module 114, for example, may acquire and release a lock for a given memory region and hierarchal locks for a plurality of memory regions. The read/write module 116, for example, may process read or write operations to the memory.

[0026] FIG. 2A depicts striping across multiple memory modules in a fault tolerant scheme, according to an example of the present disclosure. In FIG. 2A, the fault tolerant scheme utilizes a RAID-4 configuration that uses block-level striping with a dedicated parity memory module to provide redundancy to the multiple memory modules. Examples of the present disclosure, however, are not limited to the RAID-4 configuration, and may include various other fault tolerance schemes such as RAID-1, RAID-5, and RAID-6, etc. The example in FIG. 2A is used herein to define the terminology used throughout this disclosure.

[0027] Referring to FIG. 2A, the fault tolerant scheme may use memory associated with multiple memory modules (memory module 1, memory module 2, and memory module 3) to store memory blocks A1-Dp. According to an example, each memory block may include a single cache line. A cache line is the smallest unit of data that can be atomically read or written to a memory module. A cache line could be of any size used by processors, such as 64 bytes. Memory blocks A1, A2, B1, B2, C1, C2, D1, and D2 represent data blocks that are distributed across memory module 1 and memory module 2 to increase data throughput. Memory blocks Ap, Bp, Cp, and Dp represent parity blocks that are stored in memory module 3. The parity blocks provide redundancy for the data blocks. A stripe may include a combination of data blocks from at least one data memory module and at least one parity memory module. In other words, a stripe may include memory blocks distributed across multiple modules which contain redundant information, and must be atomically accessed to maintain the consistency of the redundant information. The memory blocks in a stripe may or may not be sequential in the address space of the processor. For example, one stripe may include memory blocks A1, A2, and

Ap (stripe 1), another stripe may include memory blocks B1, B2, and Bp (stripe 2), another stripe may include memory blocks C1, C2, and Cp (stripe 3), and another stripe may include memory blocks D1, D2, and Dp (stripe 4). A RAID memory group may include stripes 1-4.

[0028] According to this example, if memory module 1 fails, the data blocks from memory module 2 may be combined with the parity blocks from memory module 3 (using the Boolean XOR function) to reconstruct the missing data blocks. For instance, if memory module 1 fails, then stripe 1 may be reconstructed by performing an XOR function on data block A2 and parity block Ap to determine data block A1. In addition, the other stripes may be reconstructed in a similar manner using the fault tolerant scheme of this example. In general, a data block on a single failed memory module may be reconstructed by XORing the corresponding blocks on all of the surviving memory modules.

[0029] With reference to FIG. 2B, there is shown storage subsystem 250 for the live migration of data from a source stripe to a destination stripe according to an example of the present disclosure. The storage subsystem 250 may include multiple redundancy controllers 110A-N that each attempt to access a source stripe 210 and a destination stripe 220 during the live migration of data. Generally speaking, the multiple redundancy controllers 110A-N need to be configured to safely co-exist without creating data/parity consistency hazards or deadlocks. The methods disclosed below in FIGS. 3-8 describe examples of a live migration method for the multiple redundancy controllers 110A-N to ensure correctness and the intermediate states necessary to ensure a safe transition into and from the live migration of the data.

[0030] FIGS. 3-7 respectively depict diagrams of a live migration method 300, a staging method 400, an armed pre-migration read method 500, an armed post-migration read method 600, and an armed migration write method 700 for live migration of data between a source stripe and a destination stripe according to examples of the present disclosure. It should be apparent to those of ordinary skill in the art that the methods 300-700 represent generalized illustrations and that other operations may be added or existing operations may be removed, modified or rearranged without departing from the scopes of the methods 300-700.

[0031] According to an example, the methods discussed below may include a processor 102, a redundancy controller 110, a migration engine 112, a source stripe 210, and a destination stripe 220. The source stripe 210 and the destination stripe 220 may each include a data cache line and a parity cache line, as shown in FIG. 3. Moreover, each data cache line and parity cache line may be associated with its own media controller. Moreover, the media controllers associated with the source stripe 210 may be a disjoint set of media controllers from those associated with the destination stripe 220. Alternatively, the media controllers associated with the source stripe 210 and destination stripe 220 may be a subset, a superset, an identical set, a reordered set, etc. of one another.

[0032] With reference to FIG. 3, there is shown a bounce diagram of a live migration method 300 using hierarchal stripe-locks, according to an example of the present disclosure.

[0033] In FIG. 3, the migration engine 112 may initiate a live migration of data from the source stripe 210 to the destination stripe 220 by transmitting a migration request to the redundancy controller 110, as shown at arc 302. According to one example, the migration engine 112 may be built into the redundancy controller 110. Upon receiving the migration request from the migration engine 112, the redundancy controller 110 may request a stripe lock and parity from a media controller that hosts the parity in the destination stripe 220, as shown in arc 304.

[0034] Since there is no single point of serialization with multiple concurrent redundancy controllers, a point of serialization is created at the memory module that hosts the parity. As such, the media controller of the parity memory module is accessed by all the redundancy controllers that are talking to the same set of memory modules for serialization. According to an example, the stripe lock may be an active queue inside the media controller that hosts the parity. The active queue may include a flag or bit that indicates whether a particular stripe is locked. That is, the media controller of the parity memory module may (i) keep track of all pending lock requests for a stripe, grant the lock requests one at a time so that each requestor gets a turn in sequence to hold the lock for that stripe and (ii) perform this independently for unrelated stripes. In this regard, any subsequent accesses by other redundant controllers to the locked stripe are in conflict and may be added to a conflict queue for later access when the stripe lock is released.

[0035] As shown in arc 306, the media controller that hosts the parity in the destination stripe 220 may grant the stripe lock for the destination stripe 220 and return an old parity to the requesting redundancy controller 110. The lock is granted, for instance, if the destination stripe 220 is not locked. The redundancy controller 110 may now request a stripe lock and parity from a media controller that hosts the parity in the source stripe 210, as shown in arc 308. As shown in arc 310, the media controller that hosts the parity in the source stripe 210 may grant the stripe lock for the source stripe 210 and return an old parity to the requesting redundancy controller 110. The lock is granted, for instance, if the source stripe 210 is not locked. According to an example, the sequences represented by arcs 308 and 310 are omitted if the parity storage location (i.e., the media controller identity and stripe number within the media controller) is identical for the source stripe 210 and the destination stripe 220.

[0036] According to an example, the stripe locking between the source stripe 210 and the destination stripe 220 is coordinated by a hierarchal stripe locking protocol in order to avoid a deadlock situation. An example of a deadlock situation may occur if two concurrent independent migration operations occur. One independent migration operation may have its source stripe parity at location A and destination stripe parity at location B. The other independent migration operation may have the reverse. If each independent migration operation were to attempt to obtain a source stripe lock before obtaining a destination stripe lock, a deadlock could result where each of the two independent migration operations obtains its source lock, and then is blocked from obtaining its destination lock by the other independent migration operation. On the other hand, if all migration operations observe a consistent order of locking, based, for instance, on stripe address and device number, then each of the independent migration operations may complete in turn, with no deadlock hazard.

[0037] According to an example, the conditions required to prevent such a deadlock are that (i) all redundancy controllers make mutually consistent choices with respect to the primacy of the locks, and that these choices are consistent between all types of hierarchically-locked operations, as further discussed in below in FIGS. 3 and 7. The stripe locking protocol, for instance, may order one of the stripe locks as a primary stripe lock and another stripe lock as a secondary stripe lock and may obtain the primary stripe lock prior to the secondary stripe lock. For example, the ordering of the hierarchal stripe locks may be based on stripe numbers and media controller identification numbers for the source stripe and the destination stripe. That is, the media controller having a lower identification number may be sequenced as the primary stripe lock and the media controller having the higher identification number may be sequenced as the secondary stripe lock. Alternatively, a higher identification number may be sequenced as the primary stripe lock and the media controller having the lower identification number may be sequenced as the secondary stripe lock If the media controller identification numbers are identical, then the stripe number (i.e., address) within the memory module may be used to similarly distinguish primary from secondary. If the stripe number is also identical, then the second lock operation must be omitted. In FIG. 3, for instance, the media controller that hosts the parity in the destination stripe 220 may have a lower identification number than the media controller that hosts the parity in the source stripe 210. Thus, according to the stripe locking protocol, the stripe lock for the destination stripe 220 is acquired prior to the stripe lock for the source stripe 210 in this example. According to another example, a lower addressed stripe may be sequenced before a higher addressed stripe, or vice versa.

[0038] As shown in arcs 312 and 314, the redundancy controller 110 may read and receive data from the source stripe 210. According to an example, the data received from the source stripe 210 may include one of a data-migrated token or valid data.

[0039] In response to receiving the valid data, the redundancy controller 110 may continue the process of copying data from the source stripe 210 to the destination stripe 220. In particular, the redundancy controller 110 may read and receive data from the destination stripe 220, as shown in arcs 316 and 318. The data may be read from the destination stripe 220 in order to calculate the new data to write to the parity of the destination stripe 220. As shown in arcs 320 and 322, the redundancy controller 110 may then write data from the source stripe 210 to the data cache line of the destination stripe 220 and receive a complete message in return. As shown in arcs 324 and 326, the redundancy controller 110 may then write a data-migrated token to the data cache line of the source stripe 210 and receive a complete message in return. The redundancy controller 110 may write a new parity to the parity cache line of the source stripe 210, release the stripe lock from the parity cache line of the source stripe 210, and receive a complete message in return, as shown in arcs 328 and 330. The new source parity value written in arc 328 is calculated from the old parity value returned in arc 310, modified to reflect the change in data value from the value received in arc 312 to the data-migrated token value written in arc 324. According to an example, the sequences represented by arcs 328 and 330 are omitted if the parity storage location is identical for the source stripe 210 and the destination stripe 220. In addition, the redundancy controller may write a new parity to the parity cache line of the destination stripe 220, release the stripe lock from the parity cache line of the destination stripe 220, and receive a complete message in return, as shown in arcs 332 and 334. The new destination parity value written in arc 332 is calculated from the old parity value returned in arc 306, modified to reflect the change in data value from the value received in arc 318 to the new migrated value written in arc 320.

[0040] In response to receiving a data-migrated token, the redundancy controller 110 may determine that the data from the source stripe 210 has already been migrated to the data cache line of the destination stripe 220. Accordingly, the redundancy controller 110 may then simply release the stripe lock from the source stripe 210 and receive a complete message in return, as shown in arcs 328 and 330. According to an example, the sequences represented by arcs 328 and 330 are omitted if the parity storage location in the source stripe 210 and the parity storage location in the destination stripe 220 are the same. As shown in arcs 332 and 334, the redundancy controller 110 may then release the stripe lock from the destination stripe 220 and receive a complete message in return.

[0041] According to an example, the source stripe 210 and the destination stripe 220 may be identical with the exception that one of the memory modules is being replaced by a new memory module. In this instance, only the data and parity stored in the memory module being replaced is migrated. However, the parity calculation still involves all of the memory modules. According to another example, the migration engine 112 may then iterate through an entire region of memory whose contents are to be migrated from a source memory region to a destination memory region in accordance with the live migration method 300. Once the live migration is complete, all write and read operations will target the destination memory location.

[0042] Referring to FIG. 4, there is shown a state diagram of a method 400 for staging an entry into and an exit from the live migration of data. FIG. 4 describes an overall state sequence used to prepare for migration (i.e., arm read, arm write states), perform the migration (migrate state), and finally to resume a normal state (disarm state). According to an example, the staging includes the implementation of these states shown in FIG. 4 to ensure that the multiple redundancy controllers 110A-N in the storage subsystem 250 may safely co-exist without creating data/parity consistency hazards or deadlocks, as discussed above.

[0043] A normal state where no migration is taking place is shown in state 405. According to an example, a transition into the live migration mode may begin by starting to arm reads for each of the multiple redundancy controllers 110A-N in the storage sub-system 250. Arming the reads, for instance, may provide that each of the multiple redundancy controllers 110A-N redirect all reads to the destination stripe 220 if a data-migrated token is received when reading the source stripe 210. State 415 depicts a stage where the redundancy controllers 110A-N are being armed for reads.

[0044] Armed writes are unsafe to comingle with unarmed reads targeting the source stripe until all redundancy controllers 110A-N are in an armed read state. Thus, there can never be an armed write concurrent with an unarmed read, according to an example of the present disclosure. After all of the redundancy controllers 110A-N have been armed for reads, as shown in state 420, the redundancy controllers 110A-N may then begin to the armed for writes, as shown in state 425. Arming the writes, for instance, may provide that, for each write performed by a redundancy controller, the redundancy controller first obtains hierarchal stripe locks for the source stripe 210 and the destination stripe 220, writes data to the destination stripe 220, writes a data-migrated token to the source stripe 210, and then releases the hierarchical stripe locks for the source stripe 210 and the destination stripe 220.

[0045] According to an example, unarmed writes may safely comingle with armed writes. That is, any armed write has the side effect of performing a cache line migration and any unarmed write has the opposite side-effect, and they undo any earlier migration for that cache line. Thus, the effectiveness of migration may only be guaranteed once it is known that there are no further unarmed writes occurring. The armed write state ensures that all subsequent migrations in the migrate state may be effective because state transition 430 does not occur until all redundancy controllers 110A-N are in the armed write state.

[0046] After all the redundancy controllers110A-N are armed for writes, as shown in state 430, the redundancy controllers 110A-N are prepared to safely migrate data as directed by the migration engine 112. As shown in state 435, the multiple redundancy controllers 110A-N are directed by the migration engine 112 to iterate through the aperture of memory that is to be migrated. The method for the live migration 300 in FIG. 3 may be implemented in state 435. Once the migration engine 112 has migrated all cache lines while in the migrate state, there remains no unmigrated cache lines in the source stripe 210 (i.e., every read of the source stripe 210 returns a data-migrated token, and every read of the destination stripe 220 returns the data). Only once this state is reached, may it be safe to begin redirecting all reads and writes to directly access the destination stripe 220 without the armed read or armed write behaviors.

[0047] Accordingly, state transition 440 allows disarming of the redundancy controllers only after migration is known to be complete. Since the disarmed redundancy controllers send all writes directly to the destination stripe 220, unarmed writes no longer result in the undoing of cache line migrations, as they did in the armed write state. As shown in state 440, the redundancy controllers 110A-N may be disarmed as shown in state 445. Because of the source memory has already been migrated in state 445, the data-migrated token is valid for the entire source memory aperture that has been migrated. As such, the destination memory contains all of the migrated source memory contents. Accordingly, disarming redundancy controllers 110A-N may include updating the redundancy controllers 110A-N to only use the destination memory for future reads and writes. Once all the redundancy controllers 110A-N are disarmed, as shown in state 450, the redundancy controllers 110A-N are ready for normal state again as shown in state 405.

[0048] With reference to FIG. 5, there is shown a bounce diagram of an armed pre-migration read method 500, according to an example of the present disclosure. The processor 102, for instance, may initiate a read of data from the source stripe 210 via the redundancy controller 110, as shown in arc 502. Since this read takes place prior to the migration of data from the source stripe 210 to the destination stripe 220, the source stripe 210 returns the requested data to indicate that the requested data is still located in the data cache line of the source stripe 210 as shown in arc 504.

[0049] With reference to FIG. 6, there is shown a bounce diagram of an armed post-migration read method 600, according to an example of the present disclosure. The processor 102, for instance, may initiate a read of data from the source stripe 210 via the redundancy controller 110, as shown in arc 602. Since this read takes place after to the migration of data from the source stripe 210 to the destination stripe 220, the source stripe 210 may return a data-migrated token to indicate that the requested data is no longer located in the data cache line of the source stripe 210 as shown in arc 604. Accordingly, as shown in arc 606, the redundancy controller 110 may initiate a read of the migrated data from the data cache line of the destination stripe 220. In response read request by the redundancy controller 110, media controller of the destination stripe 220 may return the requested data from its data cache line, as shown in arc 608.

[0050] With reference to FIG. 7, there is shown a bounce diagram of an armed write method 700, according to an example of the present disclosure. The armed write method 700 may be implemented prior, during, or subsequent to the live migration according to an example.

[0051] In FIG. 7, the processor 102 may initiate a write request to the redundancy controller 110, as shown in arc 702. The write request, for instance, may be to write new data to the source stripe 210. Upon receiving the write request from the processor 102, the redundancy controller 110 may request a stripe lock and parity from a media controller that hosts the parity in the destination stripe 220, as shown in arc 704. As shown in arc 706, the media controller that hosts the parity in the destination stripe 220 may grant the stripe lock for the destination stripe 220 and return an old parity to the requesting redundancy controller 110. The stripe lock is granted, for instance, if the destination stripe 220 is not locked.

[0052] The redundancy controller 110 may now request a stripe lock and parity from the media controller that hosts the parity in the source stripe 210, as shown in arc 708. As shown in arc 710, the media controller that hosts the parity in the source stripe 210 may grant the stripe lock for the source stripe 210 and return an old parity to the requesting redundancy controller 110. The lock is granted, for instance, if the source stripe 210 is not locked. According to an example, the sequences represented by arcs 308 and 310 are omitted if the parity storage location is identical for the source stripe 210 and the destination stripe 220.

[0053] According to an example, the stripe locking between the source stripe 210 and the destination stripe 220 is coordinated by the hierarchal stripe locking protocol discussed above in order to avoid a deadlock situation. For example, according to the stripe locking protocol, the stripe lock for the destination stripe 220 is acquired prior to the stripe lock for the source stripe 210 in this example.

[0054] As shown in arcs 712 and 714, the redundancy controller 110 may read and receive old data from the data cache line of the destination stripe 220. The redundancy controller 110 may then write the new data to the data cache line of the destination stripe 220 and receive a complete message in return, as shown in arcs 716 and 718.

[0055] The redundancy controller 110 may read and receive old data from the data cache line of the source stripe 210, as shown in arcs 712 and 714. According to an example, the redundancy controller 110 may then write a data-migrated token to the data cache line of the source stripe 210 and receive a complete message in return as shown in arcs 724 and 726.

[0056] The redundancy controller 110 may write a new parity to the parity cache line of the source stripe 210, unlock the source stripe 210, and receive a complete message in return, as shown in arcs 728 and 730. The new source parity value written in arc 728 is calculated from the old parity value returned in arc 710, modified to reflect the change in data value from the value received in arc 722 to the data-migrated token value written in arc 724. Alternatively, the redundancy controller 110 may just unlock the source stripe 210 if a data-migrated token is returned during the read of the source stripe 210. According to an example, the sequences represented by arcs 728 and 730 are omitted if the parity storage location is identical for the source stripe 210 and the destination stripe 220. In addition, the redundancy controller may write a new parity to the parity cache line of the destination stripe 220, release the stripe lock from the parity cache line of the destination stripe 220, and receive a complete message in return, as shown in arcs 732 and 734. The new destination parity value written in arc 732 is calculated from the old parity value returned in arc 706, modified to reflect the change in data value from the value read in arc 714 to the new write-data value written in arc 716.

[0057] With reference to FIG. 8, there is shown a flow diagram of a live migration method 800 using hierarchal stripe-locks, according to an example of the present disclosure. The method 800 may be implemented using the processor 102 of computing device 100 according to an example.

[0058] In block 810, stripe locking module, for instance, may obtain hierarchal stripe locks for a source stripe and a destination stripe. According to an example, one of the hierarchal stripe locks may be determined as a primary stripe lock and the other hierarchal stripe lock may be determined as a secondary stripe lock. The primary stripe lock, for instance, may be obtained prior to the secondary stripe lock. The ordering of one of the hierarchal stripe locks is based on stripe numbers and identification numbers of media controllers for the source stripe and the destination stripe in according to one example. If the media controller identification numbers are identical, then the stripe number (i.e., address) within the memory module may be used to similarly distinguish primary from secondary. If the stripe number is also identical, then the second lock operation must be omitted.

[0059] In block 820, the read/write module 116, for instance, may receive a data-migrated token from the source stripe or valid data for the source stripe. A data-migrated token indicates that the data has already been migrated. If data is returned, then migration can proceed as shown in block 830.

[0060] In response to receiving read data for the source stripe, the read/write module 116, for instance, may write data from the source stripe to the destination stripe (840), write a data-migrated token to the source stripe (850), update a parity for the source stripe and the destination stripe, and release the hierarchal stripe locks for the source stripe and the destination stripe (860).

[0061] In response to receiving the data-migrated token, the read/write module 116, for instance, may release the hierarchal stripe locks for the source stripe and the destination stripe (870).

[0062] According to an example, an entry into the live migration of the data includes two stages. The first stage may include arming reads for each distributed redundancy controller. The armed reads for each distributed redundancy controller are to redirect each read to the destination stripe when the data-migrated token is received. A second stage may include arming writes for each of the distributed redundancy controllers. The armed writes for each of the distributed redundancy controllers are to obtain the hierarchal stripe locks for the source stripe and the destination stripe, write the to the destination stripe, write the data-migrated token to the source stripe, and release the hierarchal stripe locks for the source stripe and the destination stripe.

[0063] According to another example, an exit from the live migration of the data includes a disarming stage. The disarming stage may include determining that the data-migrated token is received for all source stripes in a source memory buffer and updating each of the distributed redundancy controllers to only access the destination stripes in a destination memory buffer.

[0064] What has been described and illustrated herein are examples of the disclosure along with some variations. The terms, descriptions and figures used herein are set forth by way of illustration only and are not meant as limitations. Many variations are possible within the scope of the disclosure, which is intended to be defined by the following claims -- and their equivalents -- in which all terms are meant in their broadest reasonable sense unless otherwise indicated.

CLAIMS

What is claimed is:

1. A method for live migration of data from a source stripe to a destination stripe, the method comprising:

obtaining, by a redundancy controller, hierarchal stripe locks for the source stripe and the destination stripe;

receiving a data-migrated token or the data from the source stripe;

in response to receiving the data,

writing the data from the source stripe to the destination stripe,

writing a data-migrated token to the source stripe, and

releasing the hierarchal stripe locks for the source stripe and the destination stripe; and

in response to receiving a data-migrated token indicating the data is already migrated from the source stripe, releasing the hierarchal stripe locks for the source stripe and the destination stripe.

2. The method of claim 1, comprising staging an entry into the live migration of the data, wherein a first stage comprises:

arming reads for each distributed redundancy controller, wherein the armed reads for each distributed redundancy controller are to redirect each read to the destination stripe when the data-migrated token indicating the data is already migrated from the source stripe.

3. The method of claim 2, wherein the staging comprises a second stage and the second stage comprises:

arming writes for each of the distributed redundancy controllers, wherein each write performed by a distributed redundancy controller is to:

obtain the hierarchal stripe locks for the source stripe and the destination stripe,

write the data to the destination stripe,

write the data-migrated token to the source stripe, and

release the hierarchal stripe locks for the source stripe and the destination stripe.

4. The method of claim 1, comprising staging an exit from the live migration of the data, wherein staging the exit comprises:

determining that the data migrated token is received for all source stripes in a source memory buffer; and

updating each of the distributed redundancy controllers to only access the destination stripes in a destination memory buffer.

5. The method of claim 1, wherein obtaining the hierarchal stripe locks comprises:

ordering one of the hierarchal stripe locks as a primary stripe lock and another of the hierarchal stripe locks as a secondary stripe lock; and

obtaining the primary stripe lock prior to the secondary stripe lock.

6. The method of claim 5, wherein the ordering of one of the hierarchal stripe locks is based on stripe numbers and identification numbers of media controllers for the source stripe and the destination stripe.

7. A computing device for live migration of data from a source stripe to a destination stripe, comprising a hardware redundancy controller to:

acquire hierarchal stripe locks for the source stripe and the destination stripe;

receive one of the data and a data-migrated token from the source stripe;

copy the data from the source stripe to the destination stripe and release the hierarchal stripe locks for the source stripe and the destination stripe if the data is received from the source stripe; and

release the hierarchal stripe locks for the source stripe if the data-migrated token is received from the source stripe.

8. The computing device of claim 7, wherein to copy data from the source stripe to the destination stripe, the hardware redundancy controller is to write a data-migrated token to the source stripe.

9. The computing device of claim 7, wherein the hardware redundancy controller is to arm each distributed redundancy controller to redirect reads to the destination stripe when the data-migrated token is received as a first stage prior to the live migration of data.

10. The computing device of claim 9, wherein the hardware redundancy controller is to arm writes for each of the distributed redundancy controllers, wherein each write is to acquire the hierarchal stripe locks for the source stripe and the destination stripe, copy the data from the source stripe to the destination stripe, write the data-migrated token to the source stripe, and release the hierarchal stripe locks for the source stripe and the destination stripe as a second stage prior to the live migration of data.

11. The computing device of claim 7, wherein the hardware redundancy controller is to determine that the data-migrated token is received for all source stripes in a source memory buffer and update each of the distributed

redundancy controllers to only access the destination stripes in a destination memory buffer subsequent to the live migration of data.

12. The computing device of claim 7, wherein to acquire the hierarchal stripe locks, the hardware redundancy controller is to:

order one of the hierarchal stripe locks as a primary stripe lock and another of the hierarchal stripe locks as a secondary stripe lock; and

acquire the primary stripe lock prior to the secondary stripe lock.

13. A system comprising:

a plurality of redundancy controllers;

a plurality of memory modules each including a media controller and a memory, wherein the memory modules are connected to the plurality of redundancy controllers through a memory bus,

wherein each of the redundancy controllers is to:

initiate, by a migration engine, the live migration of data from a source stripe to a destination stripe, wherein the source stripe and the destination stripe are in at least one of the plurality of memory modules;

acquire, by a stripe locking module, hierarchal stripe locks for a source stripe and a destination stripe; and

receive data-migrated token or the data for the source stripe,

wherein in response to receiving the data for the source stripe, write the data from the source stripe to the destination stripe, write a data-migrated token to the source stripe, update a parity for the source stripe and the destination stripe, and release the hierarchal stripe locks for the source stripe and the destination stripe, and

wherein in response to receiving the data-migrated token, release the hierarchal stripe locks for the source stripe and the destination stripe.

14. The system of claim 13, each of the redundancy controllers is to stage an entry into the live migration of the data, wherein to stage the entry is to:

arm reads for each distributed redundancy controller, wherein the armed reads for each distributed redundancy controller are to redirect each read to the destination stripe when the data-migrated token is received; and

arm writes for each of the distributed redundancy controllers, wherein each write by a distributed redundancy controller is to:

obtain the hierarchal stripe locks for the source stripe and the destination stripe,

write the data from the source stripe to the destination stripe,

write the data-migrated token to the source stripe,

update the parity for the source stripe and the destination stripe, and

release the hierarchal stripe locks for the source stripe and the destination stripe.

15. The system of claim 13, wherein the each of the redundancy controllers is to stage an exit from the live migration of the data, wherein to stage the exit is to:

determine that the data-migrated token is received for all source stripes in a source memory buffer; and

update each of the distributed redundancy controllers to only access the destination stripes in a destination memory buffer.

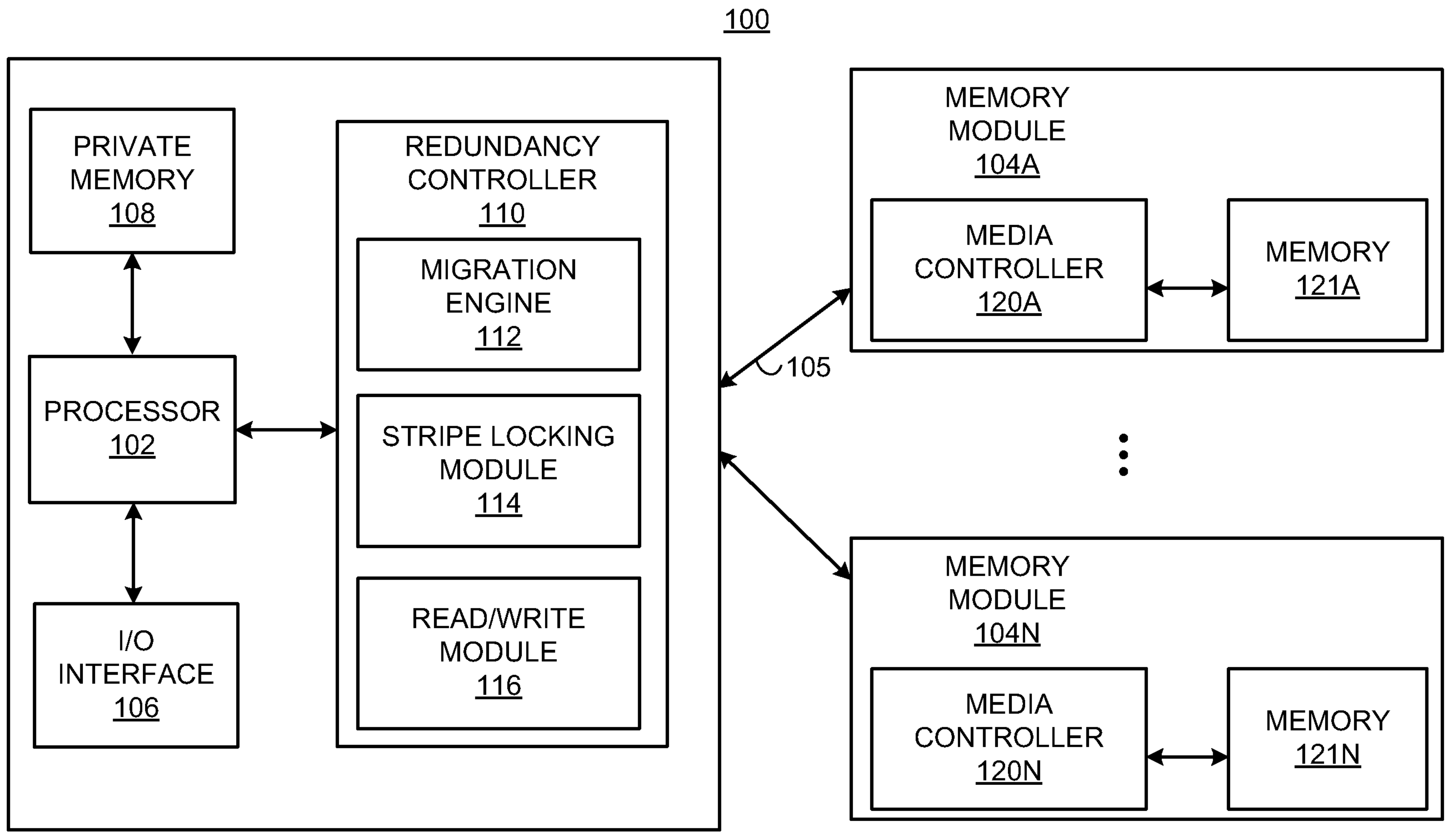
100
PRIVATE MEMORY 108
PROCESSOR 102
I/O INTERFACE 106
REDUNDANCY CONTROLLER 110
MIGRATION ENGINE 112
STRIPE LOCKING MODULE 114
READ/WRITE MODULE 116
105
MEMORY MODULE 104A
MEDIA CONTROLLER 120A
MEMORY 121A
MEMORY MODULE 104N
MEDIA CONTROLLER 120N
MEMORY 121N

FIG. 1

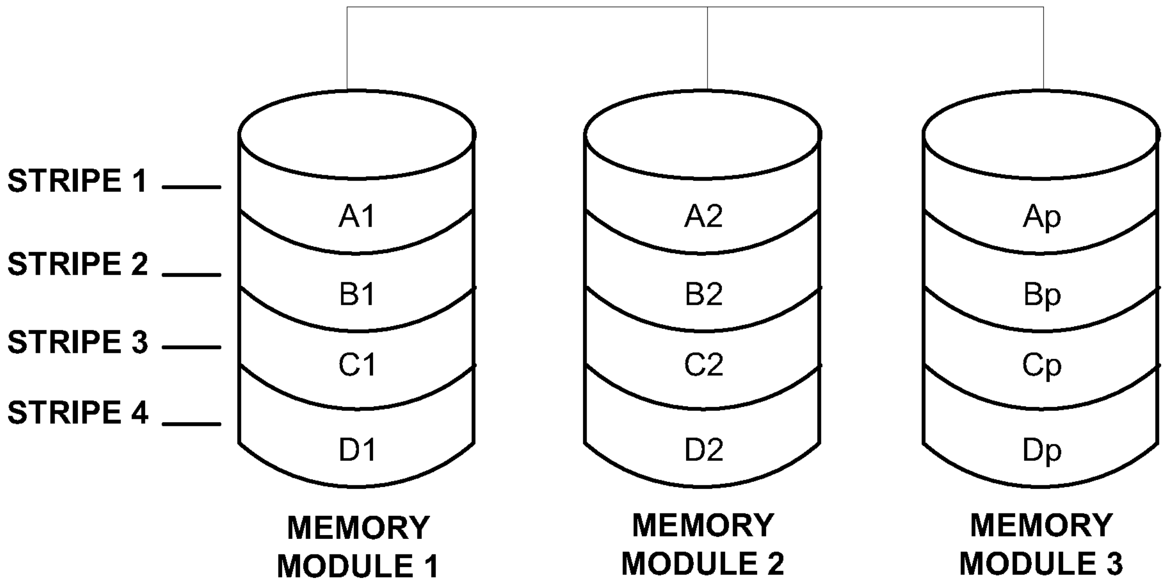
STRIPE 1
STRIPE 2
STRIPE 3
STRIPE 4
A1
B1
C1
D1
A2
B2
C2
D2
Ap
Bp
Cp
Dp
MEMORY MODULE 1
MEMORY MODULE 2
MEMORY MODULE 3

FIG. 2A

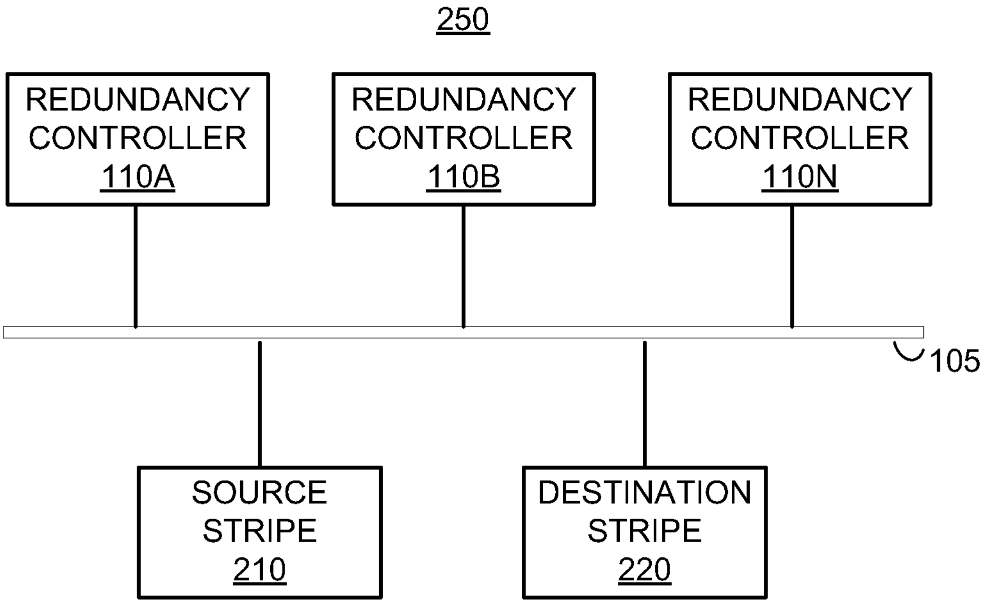
250
REDUNDANCY CONTROLLER 110A
REDUNDANCY CONTROLLER 110B
REDUNDANCY CONTROLLER 110N
105
SOURCE STRIPE 210
DESTINATION STRIPE 220

FIG. 2B

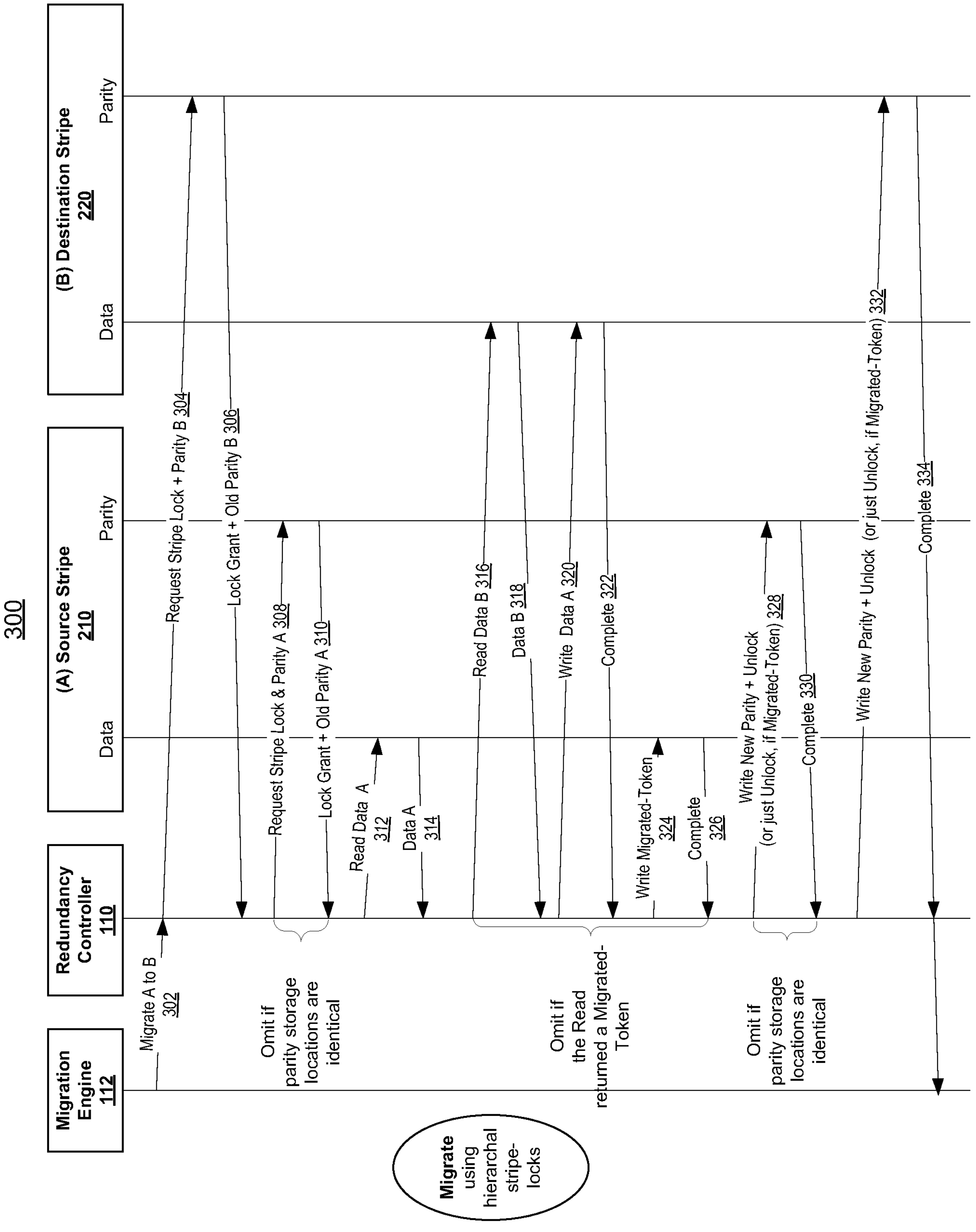
300
Migration Engine 112
Redundancy Controller 110
(A) Source Stripe 210
Data
Parity
(B) Destination Stripe 220
Data
Parity
Migrate A to B 302
Request Stripe Lock + Parity B 304
Lock Grant + Old Parity B 306
Request Stripe Lock & Parity A 308
Lock Grant + Old Parity A 310
Omit if parity storage locations are identical
Read Data A 312
Data A 314
Read Data B 316
Data B 318
Write Data A 320
Complete 322
Write Migrated-Token 324
Complete 326
Omit if the Read returned a Migrated-Token
Write New Parity + Unlock (or just Unlock, if Migrated-Token) 328
Complete 330
Omit if parity storage locations are identical
Write New Parity + Unlock (or just Unlock, if Migrated-Token) 332
Complete 334
Migrate using hierarchal stripe-locks

FIG. 3

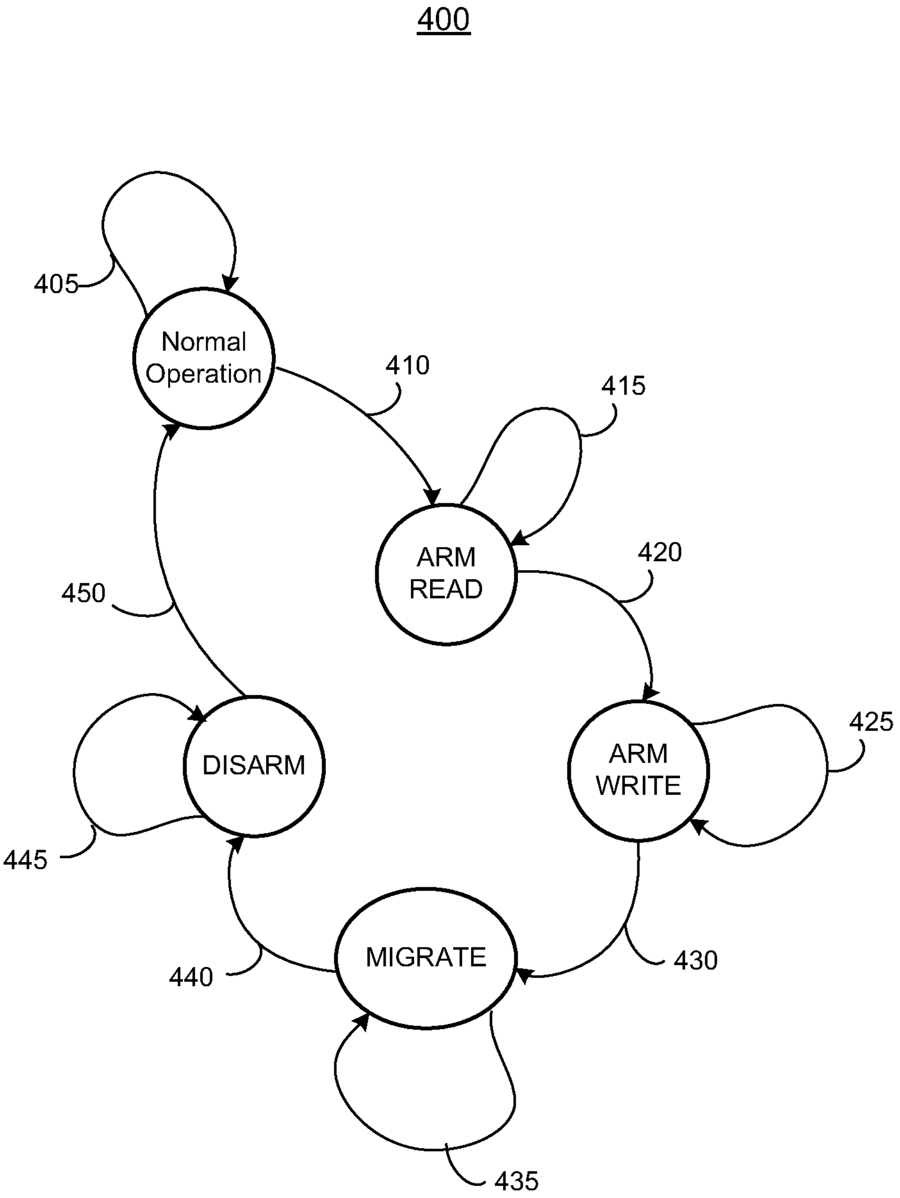
400
405
Normal Operation
410
415
ARM READ
420
450
DISARM
ARM WRITE
425
445
MIGRATE
440
430
435

FIG. 4

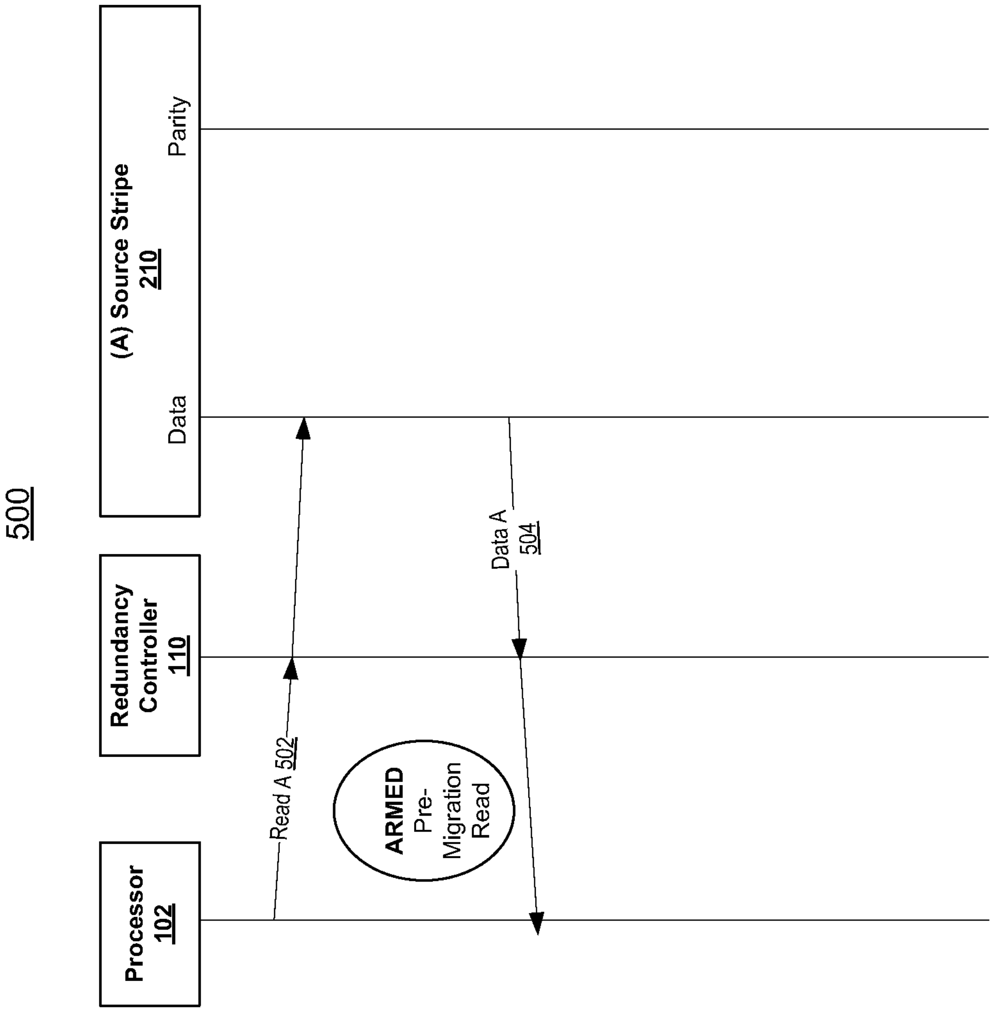
500
Processor
102
Redundancy
Controller
110
(A) Source Stripe
210
Data
Parity
Read A 502
ARMED
Pre-
Migration
Read
Data A
504

FIG. 5

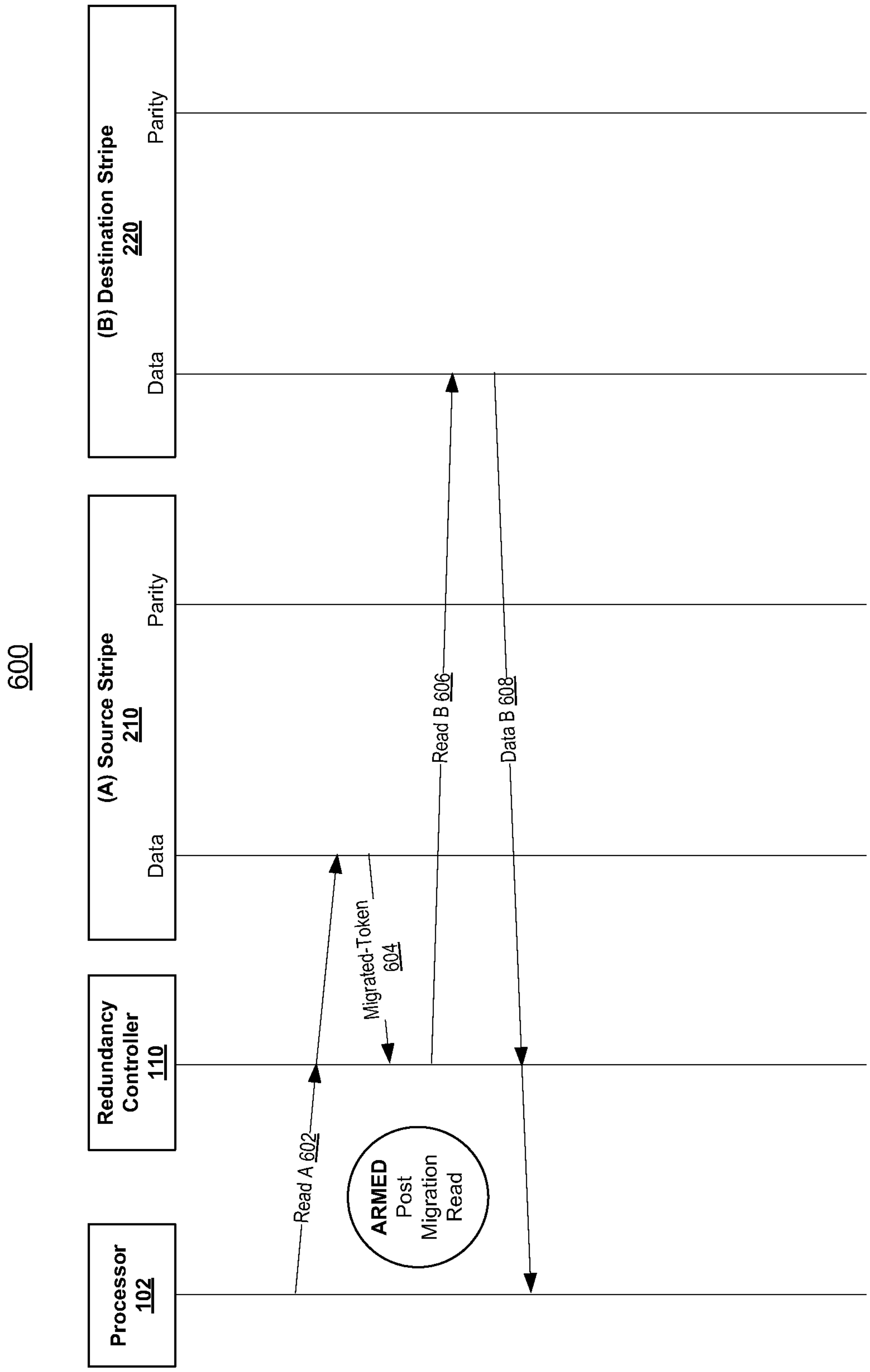
600
(B) Destination Stripe 220
Data
Parity
(A) Source Stripe 210
Data
Parity
Redundancy Controller 110
Processor 102
Read A 602
Migrated-Token 604
Read B 606
Data B 608
ARMED Post Migration Read

FIG. 6

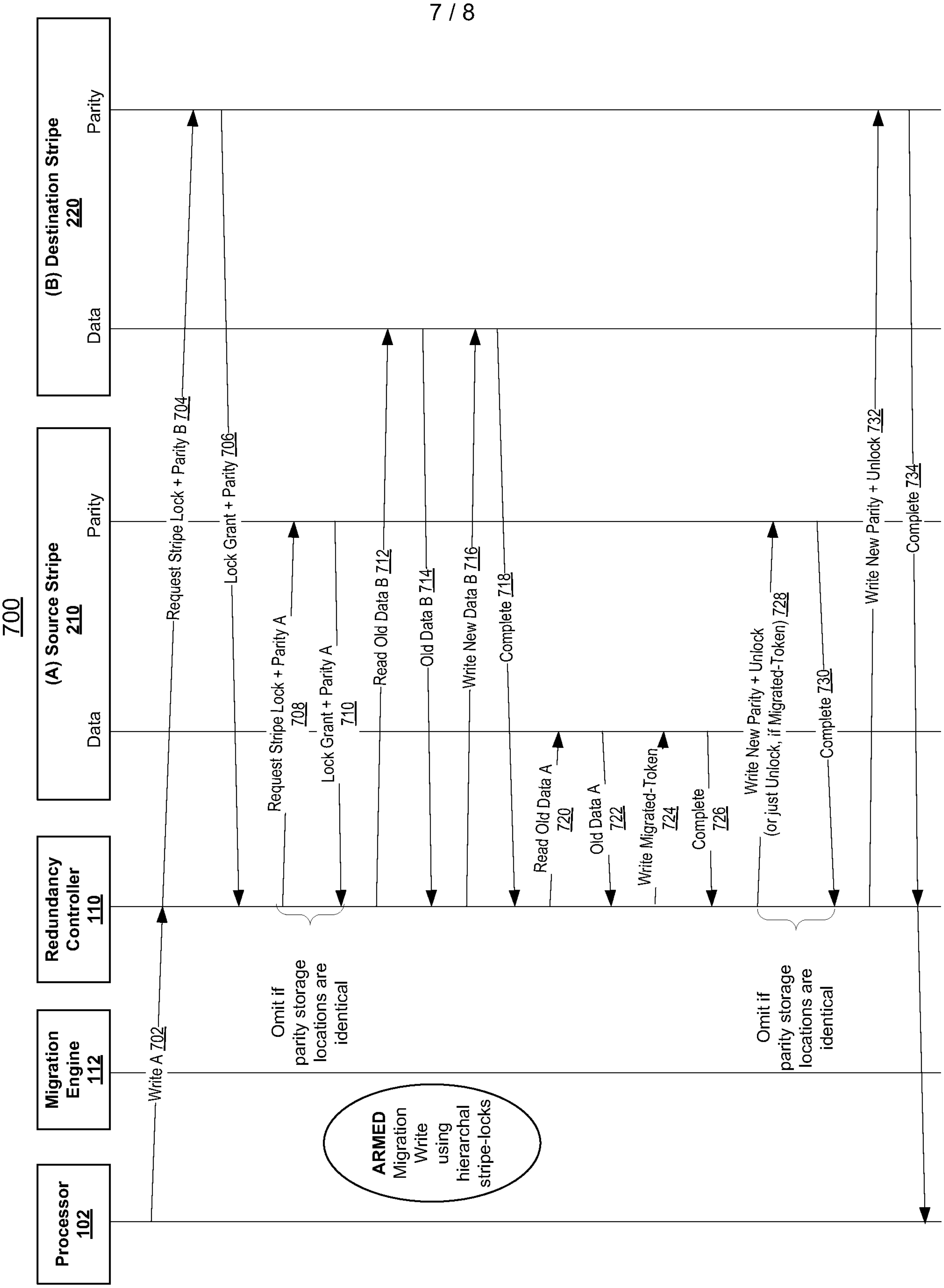
700
Processor
102
Migration
Engine
112
Redundancy
Controller
110
(A) Source Stripe
210
Data
Parity
(B) Destination Stripe
220
Data
Parity
Write A 702
Request Stripe Lock + Parity B 704
Lock Grant + Parity 706
Request Stripe Lock + Parity A
708
Lock Grant + Parity A
710
Read Old Data B 712
Old Data B 714
Write New Data B 716
Complete 718
Read Old Data A
720
Old Data A
722
Write Migrated-Token
724
Complete
726
Write New Parity + Unlock
(or just Unlock, if Migrated-Token) 728
Complete 730
Write New Parity + Unlock 732
Complete 734
Omit if
parity storage
locations are
identical
Omit if
parity storage
locations are
identical
ARMED
Migration
Write
using
hierarchal
stripe-locks

FIG. 7

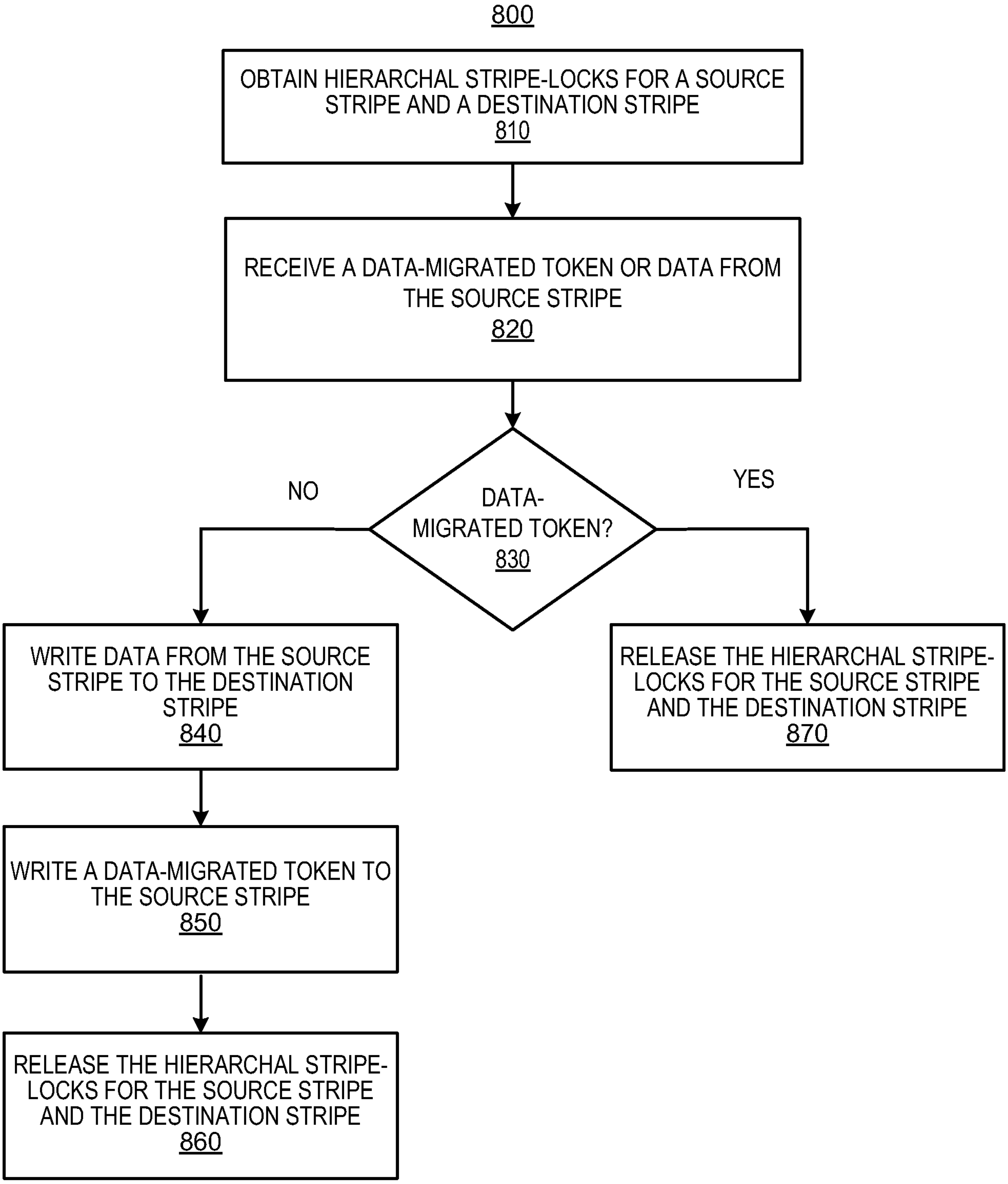
800
OBTAIN HIERARCHAL STRIPE-LOCKS FOR A SOURCE STRIPE AND A DESTINATION STRIPE
810
RECEIVE A DATA-MIGRATED TOKEN OR DATA FROM THE SOURCE STRIPE
820
DATA-MIGRATED TOKEN?
830
NO
YES
WRITE DATA FROM THE SOURCE STRIPE TO THE DESTINATION STRIPE
840
WRITE A DATA-MIGRATED TOKEN TO THE SOURCE STRIPE
850
RELEASE THE HIERARCHAL STRIPE-LOCKS FOR THE SOURCE STRIPE AND THE DESTINATION STRIPE
860
RELEASE THE HIERARCHAL STRIPE-LOCKS FOR THE SOURCE STRIPE AND THE DESTINATION STRIPE
870

FIG. 8

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2014/049193

A. CLASSIFICATION OF SUBJECT MATTER

G06F 11/10(2006.01)i

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)
G06F 11/10; G06F 15/16; G06F 12/00; G06F 13/00

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
Korean utility models and applications for utility models
Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
eKOMPASS(KIPO internal) & Keywords: redundancy, data, migration, lock, token, and similar terms.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 8341459 B2 (BALAKUMAR KAUSHIK et al.) 25 December 2012 See column 2, lines 57-67; column 7, lines 32-63; and figures 1A, 2.	1-15
A	US 7996608 B1 (PARESH CHATTERJEE et al.) 09 August 2011 See column 15, line 16 - column 17, line 9; and figures 12-13.	1-15
A	US 6654830 B1 (JAMES A. TAYLOR et al.) 25 November 2003 See column 14, line 35 - column 16, line 9; and figures 6-8.	1-15
A	US 6502165 B1 (GREGORY TAD KISHI et al.) 31 December 2002 See column 13, line 66 - column 14, line 45; and figure 11.	1-15
A	US 6467024 B1 (THOMAS WILLIAM BISH et al.) 15 October 2002 See column 11, line 10 - column 12, line 4; and figure 9.	1-15

□ 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 26 February 2015 (26.02.2015)	Date of mailing of the international search report **26 February 2015 (26.02.2015)**
Name and mailing address of the ISA/KR International Application Division Korean Intellectual Property Office 189 Cheongsa-ro, Seo-gu, Daejeon Metropolitan City, 302-701, Republic of Korea Facsimile No. ++82 42 472 3473	Authorized officer YU, Jae Chon Telephone No. +82-42-481-8647

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2014/049193

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 8341459 B2	25/12/2012	US 08788878 B2	22/07/2014
		US 2009-037679 A1	05/02/2009
		US 2013-080827 A1	28/03/2013
US 7996608 B1	09/08/2011	US 08639878 B1	28/01/2014
US 6654830 B1	25/11/2003	CN 1241134 C	08/02/2006
		CN 1268703 A	04/10/2000
		GB 2351375 A	27/12/2000
		JP 2000-339098 A	08/12/2000
		KR 10-0644011 B1	13/11/2006
		US 06446141 B1	03/09/2002
		US 06553408 B1	22/04/2003
		US 06640278 B1	28/10/2003
US 6502165 B1	31/12/2002	CN 1159651 C	28/07/2004
		CN 1299097 A	13/06/2001
		DE 10055603 A1	21/06/2001
		DE 10055603 B4	13/12/2007
US 6467024 B1	15/10/2002	None	