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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 August 2016 (04.08.2016)

WIPO | PCT

(10) International Publication Number
WO 2016/122637 A1

(51) International Patent Classification:
G06F 12/16 (2006.01) *G06F 11/16* (2006.01)

(21) International Application Number:
PCT/US2015/013898

(22) International Filing Date:
30 January 2015 (30.01.2015)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP** [US/US]; 11445 Compaq Center Drive West, Houston, TX 77070 (US).

(72) Inventors: **SHERLOCK, Derek Alan**; 3404 E. Harmony Road, Fort Collins, CO 80528 (US). **RAY, Harvey**; 3404 E. Harmony Road, Fort Collins, CO 80528 (US).

(74) Agents: **SHOOKMAN, Jeb, A.** et al.; Hewlett Packard Enterprise, 3404 E. Harmony Road, Mail Stop 79, Fort Collins, CO 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, 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, 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))*

— *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))*

WO 2016/122637 A1

(54) Title: NON-IDEMPOTENT PRIMITIVES IN FAULT-TOLERANT MEMORY

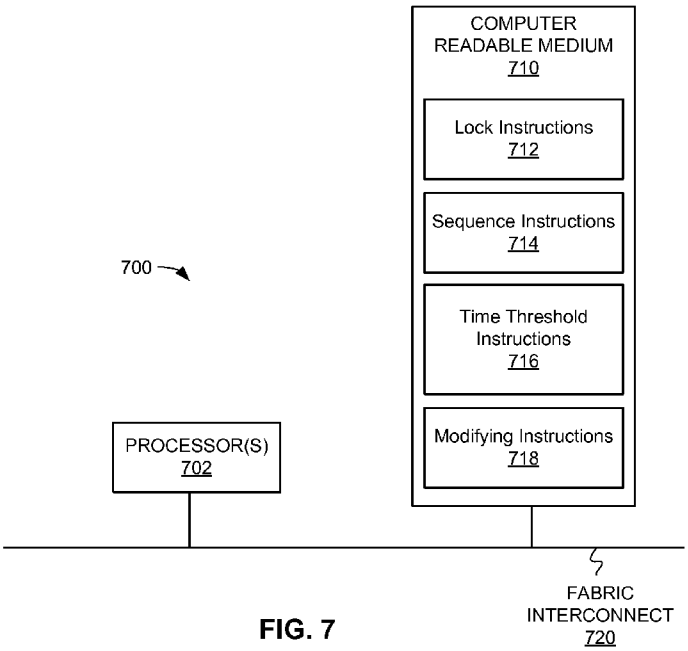


FIG. 7

(57) Abstract: According to an example, a retransmission sequence involving non-idempotent primitives in a fault-tolerant memory fabric may be modified. For example, a redundancy controller may request a sequence to access a stripe in the fault-tolerant memory fabric, wherein the sequence involves a non-idempotent primitive. In response to determining an expiration of a time threshold for the non-idempotent primitive, the redundancy controller may read other data in other cachelines in the stripe, calculate a new parity value by performing an idempotent exclusive-or primitive on the new data with the other data in the stripe, and write the new parity to the stripe using an idempotent write primitive.

NON-IDEMPOTENT PRIMITIVES IN FAULT-TOLERANT MEMORY

BACKGROUND

[0001] Current data storage devices such as volatile and non-volatile memory often include fault tolerance mechanisms to ensure that data remains available in the event of a device error or failure.

[0002] An example of a fault tolerance mechanism provided to current data storage devices is a redundant array of independent disks (RAID). RAID is a storage technology that controls multiple memory modules and provides fault tolerance by storing data with redundancy. RAID technology may store data with redundancy in a variety of ways. Examples of redundant data storage include duplicating data and storing the data in multiple memory modules and adding parity to store calculated error recovery bits. The multiple memory modules, which may include the data and associated parity, may be accessed concurrently by multiple redundancy controllers.

[0003] Another example of a fault tolerance mechanism provided to current data storage devices is an end-to-end retransmission scheme. The end-to-end retransmission scheme is utilized to create a reliable memory fabric that retransmits individual packets or entire routes that are lost enroute to a protocol agent due to transient issues such as electrical interference, or persistent issues such as the failure of a routing component, cable, or connector.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] 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:

[0005] FIG. 1A shows a simplified block diagram of a compute node to modify a retransmission sequence involving non-idempotent primitives in a fault-tolerant memory fabric, according to an example of the present disclosure;

[0006] FIG. 1B shows a simplified block diagram of a fault tolerant system to modify a retransmission sequence involving non-idempotent primitives in a fault-tolerant memory fabric, according to an example of the present disclosure;

[0007] FIG. 1C shows a redundancy controller, according to an example of the present disclosure;

[0001] FIG. 2 shows RAID stripes across multiple memory modules in a fault-tolerant memory fabric, according to an example of the present disclosure;

[0002] FIG. 3 shows a bounce diagram of an end-to-end retransmission sequence where a requested non-idempotent primitive is lost enroute to a media controller, according to an example of the present disclosure;

[0003] FIG. 4 shows a bounce diagram of an end-to-end retransmission sequence where a response packet to a requested non-idempotent primitive is lost, according to an example of the present disclosure;

[0004] FIG. 5 shows bounce diagram of a method to modify a retransmission sequence involving non-idempotent primitive packets in fault-tolerant memory, according to an example of the present disclosure;

[0005] FIG. 6 shows a flow diagram of a method to modify a retransmission sequence involving non-idempotent primitives in a fault-tolerant memory, according to an example of the present disclosure; and

[0006] FIG. 7 shows a schematic representation of a computing device, which may be employed to perform various functions of a redundancy controller, according to an example of the present disclosure.

DETAILED DESCRIPTION

[0007] 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.

[0008] In addition, the following terms will be used throughout the remainder of the present disclosure. The term fabric may mean some combination of interconnected fabric devices used to convey packet-based information between endpoint components. The term memory fabric may mean a fabric used, at least in part, to provide connectivity between redundancy controllers and media controllers. The term protocol agents may mean endpoints (e.g., producers and consumers of data) that communicate with each other over a memory fabric. The term redundancy controller may mean a requesting protocol agent that acts on behalf of a central processing unit (CPU), input output (I/O) device, or other user of memory, and generates requests such as read and write requests to one or more responding protocol agents (e.g., media controllers). The redundancy controller may be the attachment point where producers or consumers of data attach to the fabric. The redundancy controller may communicate with multiple media controllers and may implement redundant storage of data across more than one media controller on behalf of a CPU, I/O device, etc., such that the failure of a subset of the media controllers will not result in loss of data or interruption of system operation. The term media controller may mean a responding protocol agent that connects memory or storage devices to a memory fabric. The media controller may receive requests such as read and write requests, control the memory or storage devices, and

return corresponding responses. The media controller may be the attachment point where data storage attaches to the memory fabric.

[0009] The term command may mean a transaction sent from a processor, I/O device or other source to a redundancy controller, which causes the redundancy controller to issue a sequence. The term primitive may mean a single request issued by a redundancy controller to a media controller along with its corresponding response from the media controller back to the redundancy controller. The term idempotent primitive may mean a primitive that has no additional effect if the packet is received more than once. The term non-idempotent primitive may mean a primitive that has an additional effect if the packet is received more than once. The term sequence may mean an ordered set of primitives issued by a redundancy controller to one or more media controllers to execute a command received from a processor, I/O device or other source. The term locked sequence may mean a sequence that requires atomic access to multiple media controllers. The term cacheline may mean a unit of data that may be read from or written to a media controller by a redundancy controller. The term is not intended to be restrictive. The cacheline may include any type or size of data, such as a disk sector, a solid-state drive (SSD block), a RAID block or a processor cacheline. The term stripe may mean a set of one or more data cachelines, and associated redundancy information stored in one or more parity cachelines that are distributed across multiple memory modules. The term RAID degraded mode may mean a mode of operation of a RAID redundancy controller following the failure of a survivable subset of the media controllers or their memory devices. In degraded mode, reads and writes access the surviving media controllers only.

[0010] Disclosed herein are examples of methods to modify a retransmission sequence involving non-idempotent primitives in a fault-tolerant memory. The fault-tolerant memory may be a packet-switched memory fabric that connects one or more requesting protocol agents to a plurality of responding protocol agents. The fault-tolerant memory may, for instance, implement RAID storage technology. A retransmission sequence is used to create a reliable fabric when individual packets or entire routes may be lost

enroute to a protocol agent. An example of a retransmission sequence is a protocol layer end-to-end retransmission sequence, where entire protocol-layer primitives (e.g., read-request-to-read-return) are timed by the protocol agent that issues the request, and the entire primitive is repeated in the event that the entire primitive does not complete.

[0011] The protocol layer end-to-end retransmission sequence, however, may be unsafe for the carriage of non-idempotent primitives from a requesting protocol agent to a responding protocol agent in the fault-tolerant memory. For example, when a response fails to arrive back at the requesting protocol agent, the requesting protocol agent does not know whether the initial request had already reached the responding protocol agent. Thus, when the requesting protocol agent resends the request, the same packet may be received twice by the responding protocol agent.

[0012] Accordingly, the disclosed examples provide a method to safely use non-idempotent primitives across a protocol-layer end-to-end retransmission-protected memory fabric by modifying the sequence to utilize storage redundancy to reconstruct lost packets from the non-idempotent primitives. According to an example, a redundancy controller may request a sequence to access a stripe in the fault-tolerant memory fabric, wherein the sequence involves a non-idempotent primitive. In response to determining an expiration of a time threshold for the non-idempotent primitive, the redundancy controller may read other data in other cachelines in the stripe, use RAID to regenerate data that was potentially corrupted by a first transmission of the non-idempotent primitive, write the regenerated data to the stripe, reissue the non-idempotent primitive on an alternate fabric route, calculate a new parity value by performing an idempotent exclusive-or primitive on the new data with the other data in the stripe, and write the new parity to the stripe using the idempotent write primitive.

[0013] End-to-end retransmission sequences may require packets to be stored in a costly "replay buffer" by protocol agents and that an explicit end-to-end acknowledgement is delivered. For example, each request packet that

crosses the multi-hop memory fabric is acknowledged by a completion message crossing back in the other direction. The requesting protocol agent waits for the completion message while holding a copy of the requested packet in a replay buffer so that the requested packet may be resent if the completion message never arrives. The expiration of a predetermined time threshold while waiting for the completion message triggers a resend of the copy of the requested packet from the replay buffer, possibly via an alternate fabric route. Completion messages consume considerable memory fabric bandwidth, so it is advantageous to use a protocol-layer end-to-end retransmission scheme, where the protocol-layer responses serve double-purpose as delivery acknowledgements.

[0014] The protocol-layer end-to-end retransmission sequence of the disclosed examples may structure all traffic as primitives of request and response packets and rely upon the storage redundancy inherent in RAID to correct the aftermath of a lost packet in a non-idempotent sequence. A timer in the redundancy controller may monitor how long each request has been outstanding before a response is received, and if the timer reaches a predetermined time threshold, either the request packet or the response packet is known to have been lost. This approach has the advantages that (a) no dedicated replay buffer is required for storing individual request and response packets, as the entire protocol layer may be replayed instead, and (b) fabric overhead associated with completion packets or acknowledgement fields in packets may be avoided.

[0015] As discussed above, this end-to-end retransmission sequence may not be safe for the carriage of non-idempotent packets. For example, simple read and write primitives to a single media controller are idempotent provided proper precautions are taken to avoid race conditions between multiple redundancy controllers. That is, a second read primitive immediately following a first one returns identical data and a second write primitive immediately following a first one writes the same data. However, non-idempotent primitives such as swap, increment, etc. are non-idempotent. For example an increment primitive increases a stored value by one, but when delivered twice, the

increment primitive increases the stored value by two instead. Accordingly, when a response fails to arrive back at a requesting redundancy controller, the timer expiration informs the requesting redundancy controller that either the request packet or a resulting response packet has been lost. However, there is no way for the requesting redundancy controller to know which of these two packets was lost. As such, there is no way to know whether the request had already reached the media controller. Thus, the redundancy controller may resend the request, which would result in the same packet being received twice by the media controller.

[0016] Non-idempotent primitives, for example, are valuable in RAID storage. A write primitive to a RAID stripe typically includes a sequence of events whereby a redundancy controller reads old data from a media controller, then reads the old parity typically from a different media controller, writes the new data to the first media controller, calculates the new parity value from the new and old data and old parity values, and finally writes the new value of the parity to the appropriate media controller. Since individual read and write primitives are idempotent, the entire sequence may be implemented entirely composed of idempotent primitives. However, this is a very sub-optimal implementation. A more optimized implementation might, for example, merge the data read and data write into a single data swap primitive, which writes new data, while returning the prior data, or use an exclusive-or (XOR) primitive in place of a parity read primitive and parity write primitive. This increases performance and lowers memory fabric traffic by combining two primitives into one primitive. Unfortunately, the swap and XOR primitives are non-idempotent primitives. That is, if the swap primitive is received a second time, it cannot return the correct old data as it has already been overwritten in the first delivery of the swap primitive. Similarly, an XOR primitive is ineffective if received twice because the second transmission undoes the effect of the first transmission. Accordingly, the disclosed examples provide a method to safely use non-idempotent primitives in a protocol-layer end-to-end retransmission-protected memory fabric by modifying the retransmission sequence.

[0017] The disclosed examples may provide the technical benefits and advantages of an end-to-end retransmission-protected memory fabric without incurring the memory fabric overhead and replay-buffer storage costs associated with packet delivery acknowledgment. Furthermore, the disclosed examples do not incur the performance and memory fabric overhead costs of foregoing the swap and XOR primitive optimizations. In this regard, the disclosed method takes advantage of non-idempotent primitive optimizations while avoiding repeat delivery hazards that are associated with the carriage of non-idempotent primitives across a protocol-layer end-to-end retransmission-protected memory fabric during a retransmission sequence.

[0018] With reference to FIG. 1A, there is shown a block diagram of a compute node (e.g., computing device, input/output (I/O) server node) 100 for modifying a retransmission sequence involving non-idempotent primitives in a fault-tolerant memory fabric according to an example of the present disclosure. It should be understood that the compute node 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 compute node 100.

[0019] For example, the compute node 100 may include a processor 102, an input/output interface 106, a private memory 108, and a redundancy controller 110. In one example, the compute node 100 is a server but other types of compute nodes may be used. The compute node 100 may be a node of a distributed data storage system. For example, the compute node 100 may be part of a cluster of nodes that services queries and provides 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 prevent data loss and minimize down time in case of a node failure.

[0020] 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

private memory 108 may include volatile dynamic random access memory (DRAM) with or without 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. The 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 compute node 100 may receive data and user-input through the I/O interface 106. Where examples herein describe redundancy controller behavior occurring in response to commands issued by the processor 102, this should not be taken restrictively. The examples are also applicable if such commands are issued by an I/O device via interface 106.

[0021] The components of computing node 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. Alternatively, the processor 102 may use multiple different fabrics to communicate with the various components, such as PCIe for I/O, DDR3 for memory, and QPI for the redundancy controller.

[0022] The redundancy controller 110, for example, may act on behalf of the processor 102 and generate sequences of primitives such as read, write, swap, XOR, lock, unlock, etc. requests to one or more responding protocol agents (e.g., media controllers 120A-M) as discussed further below with respect to FIG. 1B. The redundancy controller 110 may communicate with multiple ones of the media controllers 120A-M, in which “M” represents an integer value greater than one, and may implement redundant storage of data across more than one media controller on behalf of the processor 102 such that the failure of a subset of the media controllers 120A-M will not result in loss of data or interruption of system operation. The redundancy controller 110, for example,

may generate certain sequences of primitives independently, not directly resulting from processor commands. These include sequences used for scrubbing, initializing, migrating, or error-correcting memory. The functionality of the redundancy controller 110 may be implemented by hardware.

[0023] With reference to FIG. 1B, there is shown a block diagram of a fault tolerant system 150 according to an example of the present disclosure. It should be understood that the system 150 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 system 150. The system 150 may include multiple compute nodes 100A-N (where the number of compute nodes is greater than or equal to 1), a network 140, and memory modules 104A-M.

[0024] The multiple compute nodes 100A-N may be coupled to the memory modules 104A-M through the network 140. The memory modules 104A-M may include media controllers 120A-M and memories 121A-M. Each media controller, for instance, may communicate with its associated memory and control access to the memory by the redundancy controllers 110A-N, which in turn are acting on behalf of the processors. The media controllers 120A-M provide access to regions of memory. The regions of memory are accessed by multiple redundancy controllers in the compute nodes 100A-N using access primitives such as read, write, lock, unlock, swap, XOR, etc. In order to support aggregation or sharing of memory, media controllers 120A-M may be accessed by multiple redundancy controllers (e.g., acting on behalf of multiple servers). Thus, there is a many-to-many relationship between redundancy controllers and media controllers. Each of the memories 121A-M 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.

[0025] As described in the disclosed examples, the redundancy controllers 110A-N may maintain fault tolerance across the memory modules 104A-M. The redundancy controller 110 may receive commands from one or more processors 102, I/O devices, or other sources. In response to receipt of these commands, the redundancy controller 110 generates sequences of primitive accesses to multiple media controllers 120A-M. The redundancy controller 110 may also generate certain sequences of primitives independently, not directly resulting from processor commands. These include sequences used for scrubbing, initializing, migrating, or error-correcting memory. The media controllers 120A-M may then respond to the requested primitives with a completion message.

[0026] RAID stripe locks acquired and released by the redundancy controller 110 guarantee atomicity for locked sequences. Accordingly, the shortened terms “stripe” and “stripe lock” has been used throughout the text to describe RAID stripes and locks on RAID stripes, respectively.

[0027] FIG. 1C shows a redundancy controller 110 according to an example of the present disclosure. The redundancy controller 110 in this example may be any one of redundancy controllers 110A-N shown. The redundancy controller 110 may include and process the functions of a time module 112, a sequence module 114, and a modification module 116.

[0028] The time module 112, for instance, may identify a time threshold for receiving an acknowledgement to a non-idempotent primitive. The sequence module 114, for instance, may initiate a sequence that includes at least one non-idempotent primitive to access a stripe in the fault-tolerant memory fabric. In response to a determination by the time module 112 that the time threshold for the non-idempotent primitive has expired, the modification module 116, for instance, may perform the following functions. The modification module 116 may request a read of other data in other cachelines in the stripe, request the use of RAID to regenerate data that was potentially corrupted by a first transmission of the non-idempotent primitive, request a write of the regenerated

data to the stripe, request a computation of a new parity value, and request an idempotent write primitive to write the new parity to the stripe.

[0029] In this example, modules 112-116 are circuits implemented in hardware. In another example, the functions of modules 112-116 may be machine readable instructions stored on a non-transitory computer readable medium and executed by a processor, as discussed further below in FIG. 7.

[0030] Referring to FIG. 2, the example fault tolerant scheme disclosed herein may use memory associated with multiple memory modules (memory module 1, memory module 2, and memory module 3) to store RAID blocks A1-Dp, in which "p" represents an integer value greater than one. According to this example, each RAID block may include a single cacheline. A cacheline is the largest unit of data that can be atomically read or written to a memory module. A cacheline could be of any size used by processors, such as 64 bytes. The use of a single cacheline RAID block size should not be taken restrictively. Cachelines A1, A2, B1, B2, C1, C2, D1, and D2 represent data cachelines that are distributed across memory module 1 and memory module 2. Cachelines Ap, Bp, Cp, and Dp represent parity cachelines that are stored in memory module 3. The parity cachelines provide redundancy for the data cachelines.

[0031] A stripe may include a combination of data cachelines from at least one memory module and parity cachelines from at least one other 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. For example, one stripe may include cachelines A1, A2, and Ap (stripe 1), another stripe may include cachelines B1, B2, and Bp (stripe 2), another stripe may include cachelines C1, C2, and Cp (stripe 3), and another stripe may include cachelines D1, D2, and Dp (stripe 4). The data cachelines in a stripe may or may not be sequential in the address space of the processor 102. A RAID memory group may include stripes 1-4. The example in FIG. 2 represents a RAID-4 configuration, where all parity cachelines are stored on a single memory module. Other RAID configurations, such as RAID-1 where the parity cachelines

are mirrors of the data cachelines, and RAID-5 where parity cachelines distributed across all memory modules, and other redundancy schemes are also covered by the present disclosure.

[0032] According to this example, if memory module 1 fails, the data cachelines from memory module 2 may be combined with the corresponding-stripe parity cachelines from memory module 3 (using the boolean exclusive-or function) to reconstruct the missing cachelines. For instance, if memory module 1 fails, then stripe 1 may be reconstructed by performing an exclusive-or function on data cacheline A2 and parity cacheline Ap to determine data cacheline A1. In addition, the other stripes may be reconstructed in a similar manner using the fault tolerant scheme of this example. In general, a cacheline on a single failed memory module may be reconstructed by using the exclusive-or function on the corresponding-stripe cachelines on all of the surviving memory modules. The use of the simple exclusive-or operation in the reconstruction of missing data should not be taken restrictively. Different data-recovery operations may involve different mathematical techniques. For example, RAID-6 commonly uses Reed-Solomon codes.

[0033] As discussed above, when a response packet is lost, for instance, the response packet is not received at a requesting redundancy controller within a predetermined time threshold in other end-to-end retransmission sequences, the requesting redundancy controller does not know whether the request packet was lost enroute to the media controller or whether the resulting response packet was lost on a return route to the requesting redundancy controller. These two scenarios are illustrated in FIGS. 3 and 4 below.

[0034] FIG. 3 shows a bounce diagram of an end-to-end retransmission sequence 300 where a requested non-idempotent primitive is lost enroute to a media controller, according to an example of the present disclosure. Redundancy controller 110 and three media controllers 120A-C are depicted in FIG. 3. Redundancy controller 110 accesses a stripe 301 that includes data cachelines stored in the first and second memory modules 303, 307 and a parity cacheline stored in the third memory module 305. It is noted that transactions

that are needed to update and compute a new parity are not shown to avoid clutter. Transactions to acquire and release stripe locks to ensure atomicity in the presence of multiple redundancy controllers are similarly omitted.

[0035] At arc 302, the redundancy controller 110 issues a non-idempotent primitive request packet, such as a swap primitive, to the media controller 120A of the first data memory module 303 in stripe 301. The swap primitive is a non-idempotent primitive that reads the old data, and then writes the new data to the first data memory module 303 in one merged primitive. As such, the redundancy controller 110 requests to swap new data with old data in arc 302. In the example of FIG. 3, however, the non-idempotent swap primitive request packet is lost enroute to the media controller 120A due to a memory fabric failure. Thus, after waiting for the expiration of a time threshold (i.e., a timeout), the redundancy controller 110 may reissue the non-idempotent swap primitive request packet to the media controller 120A via an alternate fabric route in accordance with the end-to-end retransmission sequence protocol, as shown in arc 302. If the non-idempotent swap primitive request packet is received and successful, the media controller 120A may return a response packet with the old data to the redundancy controller 110 as shown in arc 304.

[0036] In this scenario, the non-idempotent swap primitive does not have a harmful effect upon the end-to-end retransmission sequence because the initial non-idempotent primitive packet in arc 302 never reached the media controller 120A. In other words, the old data in the first data memory module 303 in stripe 301 remains unchanged, and therefore the reissued non-idempotent swap primitive request packet in arc 304 will receive the correct old data as shown in arc 306.

[0037] On the other hand, FIG. 4 shows a bounce diagram of an end-to-end retransmission sequence 400 where a response packet to a requested non-idempotent primitive is lost, according to an example of the present disclosure. Once again it is noted that transactions that are needed to update and compute a new parity are not shown to avoid clutter. Transactions to acquire and release

stripe locks to ensure atomicity in the presence of multiple redundancy controllers are similarly omitted.

[0038] At arc 402, the redundancy controller 110 issues a non-idempotent primitive request packet, such as a swap primitive, to the media controller 120A of the first data memory module 303 in stripe 301. Upon receiving the non-idempotent swap primitive request packet, the media controller 120A may store the new data in memory, and return old data from the first data memory module 303 in a response packet to the redundancy controller 110, as shown in arc 404. However, on the return route, the response packet of arc 404 is lost due to a memory fabric failure.

[0039] After waiting for the expiration of a time threshold, the redundancy controller 110 may reissue the non-idempotent swap primitive request packet to the media controller 120A via an alternate fabric route in accordance with the end-to-end retransmission sequence protocol, as shown in arc 406. However, this reissued non-idempotent swap primitive request packet is harmful because the new data has already been written to the first data memory module 303 in stripe 301 by the initial swap primitive of arc 402. That is, media controller 120A may receive the same non-idempotent swap primitive packet twice. In this regard, the media controller 120A returns a response packet to the redundancy controller 110 with corrupted data in arc 408. That is, the media controller 120 returns a response packet with incorrect “old” data that has already been overwritten with the new data in the initial swap primitive of arc 402.

[0040] Accordingly, in this scenario, the non-idempotent swap primitive has a harmful effect upon the end-to-end retransmission sequence because the initial primitive packet in arc 402 was received and executed by the media controller 120A. In other words, the old data in the first data memory module 303 in stripe 301 is swapped with the new data, and therefore the reissued non-idempotent swap primitive request packet in arc 402 will receive incorrect and corrupt data as shown in arc 408.

[0041] The methods disclosed below in FIGS. 5-7 describe examples of methods for modifying a retransmission sequence involving non-idempotent

primitive packets to avoid the duplicate access hazards described in FIG. 4 above. It should be apparent to those of ordinary skill in the art that the methods 500-700 represent generalized illustrations and that other sequences may be added or existing sequences may be removed, modified or rearranged without departing from the scopes of the methods 500-700.

[0042] FIG. 5 shows a bounce diagram of a method 500 to modify a retransmission sequence involving non-idempotent primitive packets in fault-tolerant memory, according to an example of the present disclosure. The disclosed example of FIG. 5 provides a method to safely use non-idempotent primitives across a protocol-layer end-to-end retransmission-protected memory fabric by modifying the sequence to utilize storage redundancy to reconstruct lost packets from the non-idempotent primitives.

[0043] At arc 502, the redundancy controller 110 issues a non-idempotent swap primitive request packet to the media controller 120A of the first data memory module 303 in stripe 301. Upon receiving the non-idempotent swap primitive request packet, the media controller 120A may return old data from the first data memory module 303 in a response packet to the redundancy controller 110, as shown in arc 504. However, on the return route, the response packet of arc 504 may be lost due to a memory fabric failure.

[0044] After waiting for the expiration of a time threshold, the redundancy controller 110 may use redundancy to reconstruct and write the old data in the first data memory module 303. For example, the redundancy controller 110 may issue a read primitive to the second data memory module 307 and may issue a read primitive to the parity memory module 305, as shown in arcs 506 and 508. As a result, the redundancy controller 110 may receive the old parity from the media controller 120B and data from media controller 120C, as shown in arcs 510 and 512. At arc 514, the redundancy controller 110 may then reconstruct the old data by performing an exclusive-or function on the old parity and the data received in arcs 510 and 512 and write the old data to the first data memory module 303. In response to writing the old data, the redundancy

controller 110 may receive a completion message from the media controller 120A at arc 516.

[0045] According to an example, the redundancy controller 110 may attempt an end-to-end retransmission of the non-idempotent swap primitive on an alternate fabric route, as shown at arc 518. Upon receiving the non-idempotent swap primitive request, the media controller 120A may return old data from the first data memory module 303 in a response packet to the redundancy controller 110, as shown at arc 520. As shown at arc 522, the redundancy controller 110 may then issue an idempotent write primitive to commit a new parity to the parity memory module 305 to keep the stripe 301 consistent. At arc 524, the redundancy controller 110 may receive a completion message from the media controller 120B. At this point, method 500 has successfully modified a retransmission sequence to utilize redundancy and an alternate fabric route to safely recover from a fabric failure.

[0046] With reference to FIG. 6, there is shown a flow diagram of a method 600 to modify a retransmission sequence involving non-idempotent primitives in a fault-tolerant memory, according to an example of the present disclosure.

[0047] Prior to performing a sequence that involves at least one non-idempotent primitive, a determination is made as to whether the redundancy controller is in a degraded mode. The degraded mode is an operation of the redundancy controller following the failure of a survivable subset of the media controllers or their memory devices. In a degraded mode, reads and writes access the surviving media controllers only. According to an example, in response to determining that the redundancy controller is in the degraded mode, an alternate sequence that involves only idempotent primitives may be requested. On the other hand, in response to determining that the redundancy controller is not in the degraded mode, the redundancy controller may perform the sequence that involves the at least one non-idempotent primitive.

[0048] In this regard, the redundancy controller 110 may request a lock from the media controller 120B. The stripe 301 may include data stored in at

least one data memory module and parity stored in at least one parity module. Since there is typically no single point of serialization for multiple redundancy controllers that access a single stripe, a point of serialization may be created at the media controller 120B. The point of serialization may be created at the media controller 120B because any sequence that modifies the stripe 301 has to access the parity. Accordingly, the redundancy controller 110 may obtain the lock for the stripe 301. The lock may provide the redundancy controller 110 with exclusive access to the stripe. For instance, the lock prevents a second redundancy controller from concurrently performing a second sequence that accesses multiple memory modules in the stripe 301 during the locked sequence.

[0049] As shown in block 610, the redundancy controller, for instance, may request a sequence to access a stripe in the fault-tolerant memory fabric. In this example, the redundancy controller is not in a degraded mode and the sequence includes at least one non-idempotent primitive. The non-idempotent primitive may be, for example, an optimized swap primitive that merges a data read primitive and a data write primitive into a single primitive. In another example, the non-idempotent primitive may be an optimized exclusive-or primitive that merges a parity read primitive and a parity write primitive merged into a single primitive. Specifically, the optimized exclusive-or primitive includes performing an idempotent exclusive-or primitive on old data from the cacheline with the new data from the cacheline to determine a resulting exclusive-or value, performing the idempotent exclusive-or primitive on the resulting exclusive-or value with an old parity to determine a new parity, and writing the new parity to the stripe using the idempotent write primitive.

[0050] According to an example, the redundancy controller may set a timer to monitor the duration between requesting a non-idempotent primitive and receiving a completion message for the requested non-idempotent primitive from an associated media controller. In block 620, the redundancy controller may determine that the duration for the non-idempotent primitive has exceeded a predetermined time threshold (e.g., the time threshold has expired). The time threshold, for instance, is a predefined duration in which a completion message

to a requested primitive is expected to be received at the redundancy controller from the media controller.

[0051] In response to determining an expiration of a time threshold for the non-idempotent primitive, the redundancy controller may read other data in other cachelines in the stripe as shown in block 630 and use RAID to regenerate data that was potentially corrupted by a first transmission of the non-idempotent primitive as shown in block 640. As a result, the redundancy controller may request a write primitive to write the regenerated data to the stripe as shown in block 650. Furthermore, the redundancy controller may reissue the non-idempotent primitive on an alternate fabric route in block 660 and calculate a new parity value by performing an idempotent exclusive-or primitive on the new data with the other data in the stripe as shown in block 670. In this regard, for instance, the redundancy controller may request an idempotent write primitive to write the new parity to the stripe, as shown in block 680. Accordingly, the retransmission sequence is modified in method 600 to avoid data corruption in the stripe and safely use non-idempotent primitives in a protocol-layer end-to-end retransmission-protected memory fabric.

[0052] According to another example, if the redundancy controller receives the completion message within the predetermined time threshold, then the redundancy may acknowledge that the requested non-idempotent primitive has been completed on the stripe and proceed with the next primitive of the sequence. For example, the redundancy controller may request another non-idempotent primitive in the sequence and iteratively perform the steps in blocks 620-680 if necessary. Once the sequence has completed, the redundancy controller 110 may then release the lock for the stripe 301.

[0053] Some or all of the operations set forth in the methods 500 and 600 may be contained as utilities, programs, or subprograms, in any desired computer accessible medium. In addition, the methods 500 and 600 may be embodied by computer programs, which may exist in a variety of forms both active and inactive. For example, they may exist as machine readable instructions, including source code, object code, executable code or other

formats. Any of the above may be embodied on a non-transitory computer readable storage medium.

[0054] Examples of non-transitory computer readable storage media include conventional computer system RAM, ROM, EPROM, EEPROM, and magnetic or optical disks or tapes. It is therefore to be understood that any electronic device capable of executing the above-described functions may perform those functions enumerated above.

[0055] Turning now to FIG. 7, a schematic representation of a computing device 700, which may be employed to perform various functions of the redundancy controller 110 as depicted in FIG. 1C, is shown according to an example implementation. The device 700 may include a processor 702 coupled to a computer-readable medium 710 by a fabric interconnect 720. The computer readable medium 710 may be any suitable medium that participates in providing instructions to the processor 702 for execution. For example, the computer readable medium 710 may be non-volatile media, such as an optical or a magnetic disk; volatile media, such as memory.

[0056] The computer-readable medium 710 may store instructions to perform methods 500 and 600. For example, the computer-readable medium 710 may include machine readable instructions such as lock instructions 712 to request and release a lock to perform a sequence with a non-idempotent primitive on the stripe, sequence instructions 714 to request a sequence with a non-idempotent primitive on the stripe, time threshold instructions 716 to determine whether a time threshold for receiving a completion message for the non-idempotent primitive has expired, and in response to determining the expiration of the time threshold, modifying instructions 718 to initiate a use of RAID to regenerate data that was potentially corrupted by a first transmission of the non-idempotent primitive, initiate a write of the regenerated data to the stripe, initiate a computation of a new parity value, and initiate the idempotent write primitive to write the new parity to the stripe. In this regard, the computer-readable medium 710 may include machine readable instructions to perform methods 500 and 600 when executed by the processor 702.

[0057] 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 modifying a retransmission sequence involving non-idempotent primitives in a fault-tolerant memory fabric, the method comprising:

requesting, by a redundancy controller, a sequence to access a stripe in the fault-tolerant memory fabric, wherein the sequence involves a non-idempotent primitive; and

in response to determining an expiration of a time threshold for the non-idempotent primitive,

reading other data in other cachelines in the stripe,

using redundant array of independent disks (RAID) to regenerate data that was potentially corrupted by a first transmission of the non-idempotent primitive,

writing the regenerated data to the stripe,

reissuing the non-idempotent primitive on an alternate fabric route,

calculating a new parity value by performing an idempotent exclusive-or primitive on the new data with the other data in the stripe, and

writing the new parity to the stripe using the idempotent write primitive.

2. The method of claim 1, further comprising:

acknowledging that the requested non-idempotent primitive has been completed on the stripe in response to receiving a response to the requested non-idempotent primitive within the time threshold.

3. The method of claim 1, wherein the time threshold is a predefined duration in which a completion message to a requested primitive is expected to be received at the redundancy controller from the media controller.

4. The method of claim 1, further comprising:

obtaining a stripe lock prior to requesting the sequence to access the stripe in the memory fabric; and

releasing the stripe lock in response to completing the sequence.

5. The method of claim 1, wherein the non-idempotent primitive comprises an optimized swap primitive that merges a data read primitive and a data write primitive into a single primitive.

6. The method of claim 1, wherein the non-idempotent primitive comprises an optimized exclusive-or primitive that merges a parity read primitive and a parity write primitive merged into a single primitive.

7. The method of claim 6, wherein the optimized exclusive-or primitive further comprises:

performing the idempotent exclusive-or primitive on old data from the cacheline with the new data from the cacheline to determine a resulting exclusive-or value;

performing the idempotent exclusive-or primitive on the resulting exclusive-or value with an old parity to determine a new parity; and

writing the new parity to the stripe using the idempotent write primitive.

8. The method of claim 2, wherein acknowledging that the non-idempotent

primitive has been completed on the stripe further comprises:

requesting another non-idempotent primitive in the sequence;

in response to determining an expiration of the time threshold for the other non-idempotent primitive,

reading other data in the other cachelines in the stripe,

using RAID to regenerate data that was potentially corrupted by a first transmission of the other non-idempotent primitive,

writing the regenerated data to the stripe,

calculating a new parity value by performing an idempotent exclusive-or primitive on the new data with the other data in the stripe, and

writing the new parity to the stripe using the idempotent write primitive.

9. The method of claim 1, wherein prior to requesting the sequence to access the stripe, determining whether the redundancy controller is in degraded mode; and

in response to determining that the redundancy controller is in the degraded mode, requesting an alternate sequence involving only idempotent primitives.

10. A redundancy controller to modify a retransmission sequence involving non-idempotent primitives in a fault-tolerant memory fabric, the redundancy controller comprising:

a time module to identify a time threshold for receiving an acknowledgement to a non-idempotent primitive;

a sequence module to initiate a sequence that includes at least one non-idempotent primitive to access a stripe in the fault-tolerant memory fabric; and

a modification module to, in response to the time module determining an expiration of the time threshold for the non-idempotent primitive:

request a read of other data in other cachelines in the stripe,

request the use of redundant array of independent disks (RAID) to regenerate data that was potentially corrupted by a first transmission of the non-idempotent primitive,

request a write of the regenerated data to the stripe,

request a computation of a new parity value, wherein the computation includes performing an idempotent exclusive-or primitive on the new data with the other data in the stripe, and

request the idempotent write primitive to write the new parity to the stripe.

11. The redundancy controller of claim 10, wherein in response to receiving the acknowledgement of the non-idempotent swap primitive within the time threshold, the sequence module is to acknowledge that the requested non-idempotent primitive has been completed on the stripe.

12. The redundancy controller of claim 10, wherein the sequence module is to:

obtain a stripe lock prior to requesting the sequence to access the stripe in the memory fabric; and

release the stripe lock in response to completing the sequence.

13. A non-transitory computer readable medium to obtain a lock for a stripe from a media controller to perform a sequence of non-idempotent primitives, including machine readable instructions executable by a processor to:

request a lock on the stripe;

request a sequence with a non-idempotent primitive on the stripe;

in response to determining an expiration of a time threshold for receiving a completion message for the non-idempotent primitive,

initiate a use of redundant array of independent disks (RAID) to regenerate data that was potentially corrupted by a first transmission of the non-idempotent primitive,

initiate a write of the regenerated data to the stripe,

initiate a computation of a new parity value, wherein the computation includes performing an idempotent exclusive-or primitive on the new data with the other data in other cachelines in the stripe, and

initiate the idempotent write primitive to write the new parity to the stripe;

in response to receiving a response to the non-idempotent swap primitive within the time threshold, acknowledge the completion of the non-idempotent primitive on the stripe; and

release the lock for the stripe.

14. The non-transitory computer readable medium of claim 13, wherein to acknowledge the completion of the non-idempotent primitive on the stripe, the machine readable instructions are executable by the processor to request another non-idempotent primitive in the sequence.

15. The non-transitory computer readable medium of claim 13, wherein in response to determining the expiration of the time threshold for the other non-idempotent primitive, the machine readable instructions are executable by the processor to:

initiate a read of the other data in the other cachelines in the stripe,

initiate a use of RAID to regenerate data that was potentially corrupted by

a first transmission of the non-idempotent primitive,

initiate a write of the regenerated data to the stripe,

initiate a computation of a new parity value by performing an idempotent exclusive-or primitive on the new data with the other data in the stripe, and

initiate a write of the new parity to the stripe using the idempotent write primitive.

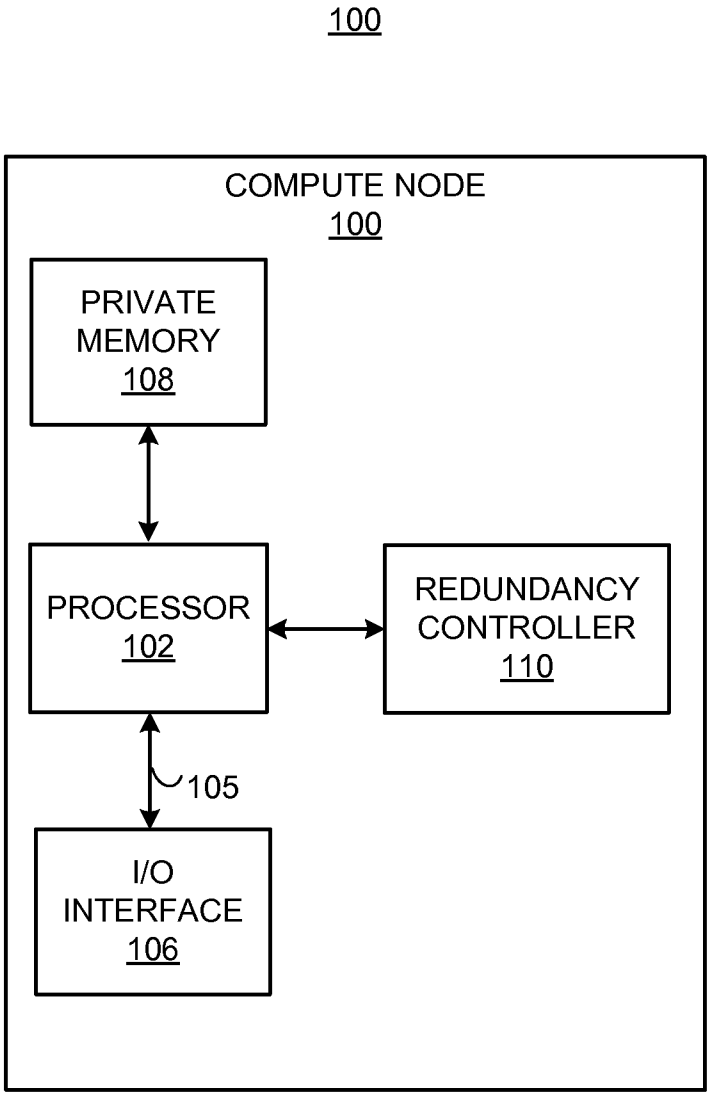

100
COMPUTE NODE
100
PRIVATE
MEMORY
108
PROCESSOR
102
REDUNDANCY
CONTROLLER
110
105
I/O
INTERFACE
106

FIG. 1A

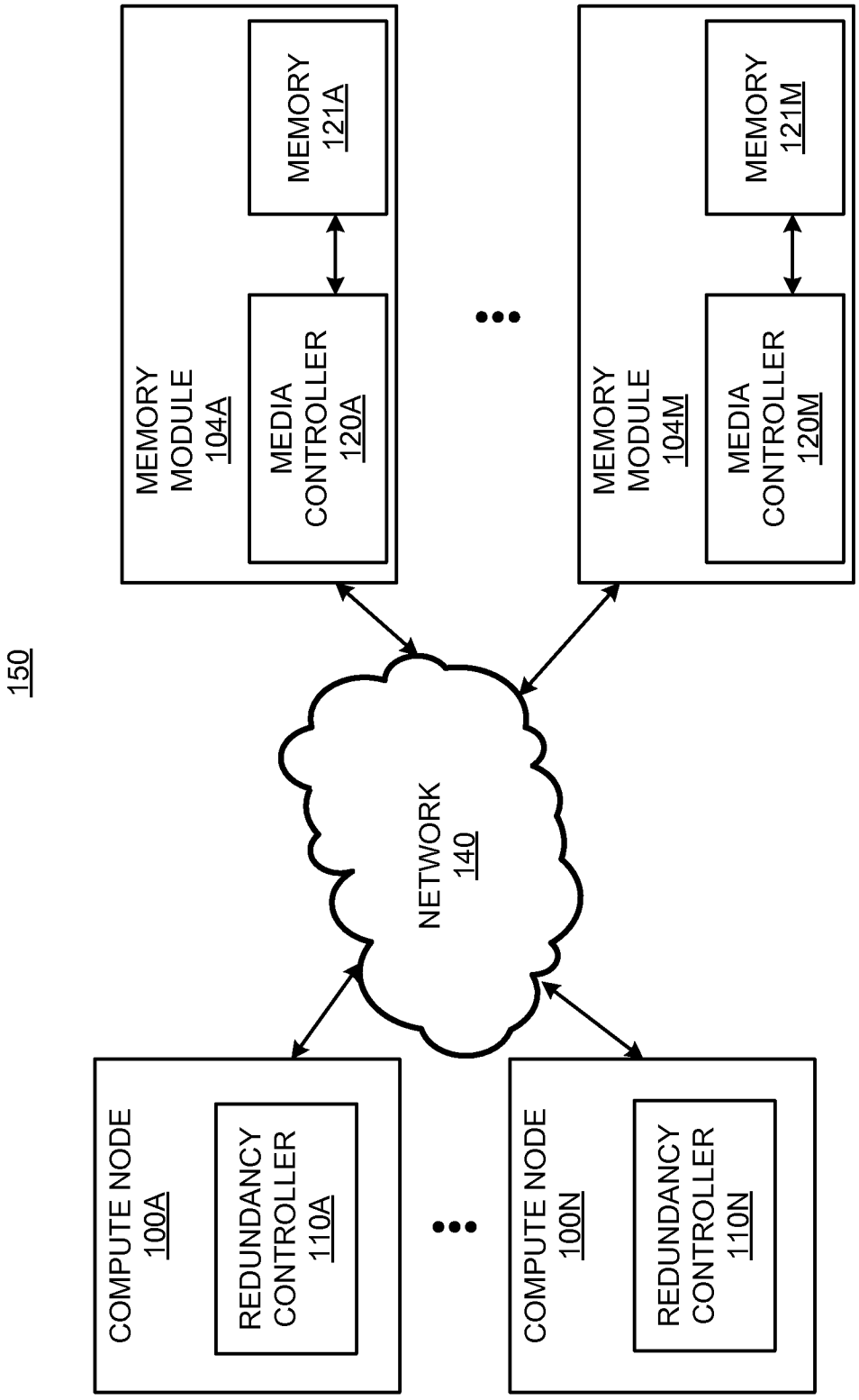

150
COMPUTE NODE 100A
REDUNDANCY CONTROLLER 110A
COMPUTE NODE 100N
REDUNDANCY CONTROLLER 110N
NETWORK 140
MEMORY MODULE 104A
MEDIA CONTROLLER 120A
MEMORY 121A
MEMORY MODULE 104M
MEDIA CONTROLLER 120M
MEMORY 121M

FIG. 1B

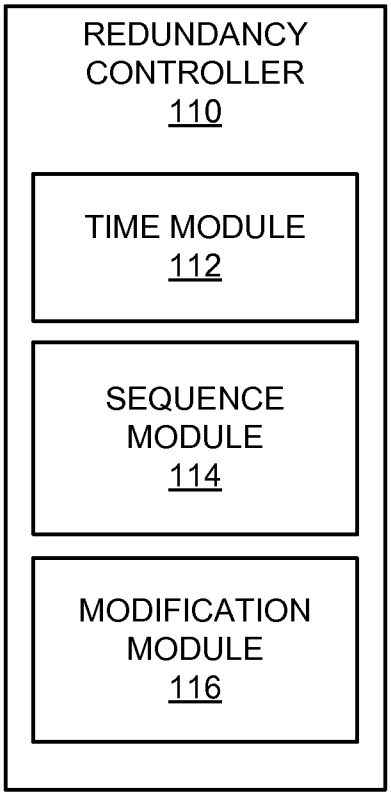

REDUNDANCY CONTROLLER
110
TIME MODULE
112
SEQUENCE MODULE
114
MODIFICATION MODULE
116

FIG. 1C

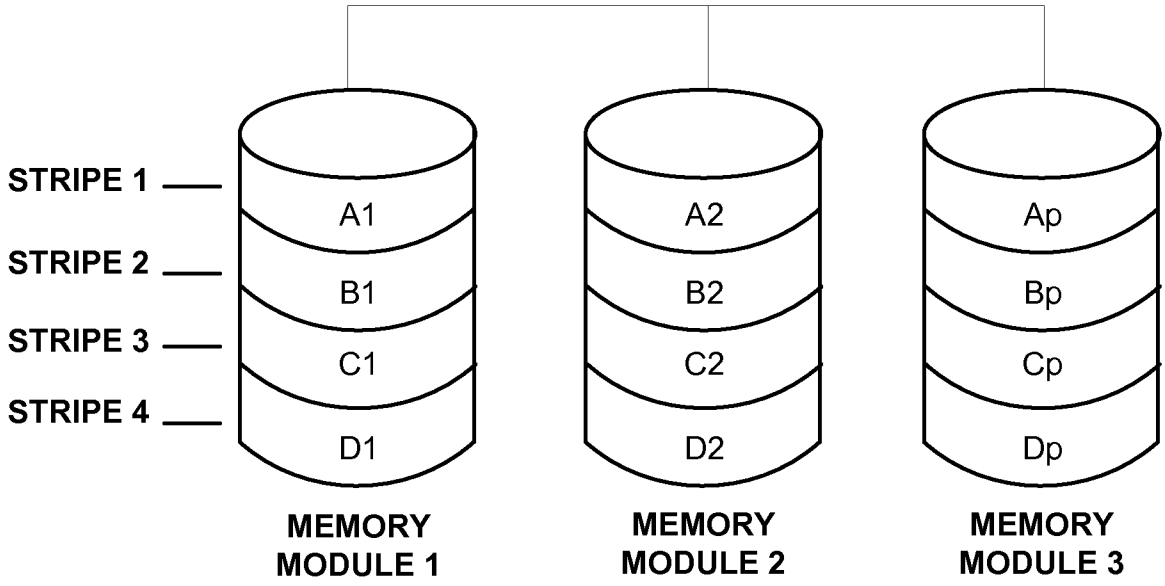

STRIPE 1
STRIPE 2
STRIPE 3
STRIPE 4
A1
B1
C1
D1
A2
B2
C2
D2
Ap
Bp
Cp
Dp
MEMORY MODULE 1
MEMORY MODULE 2
MEMORY MODULE 3

FIG. 2

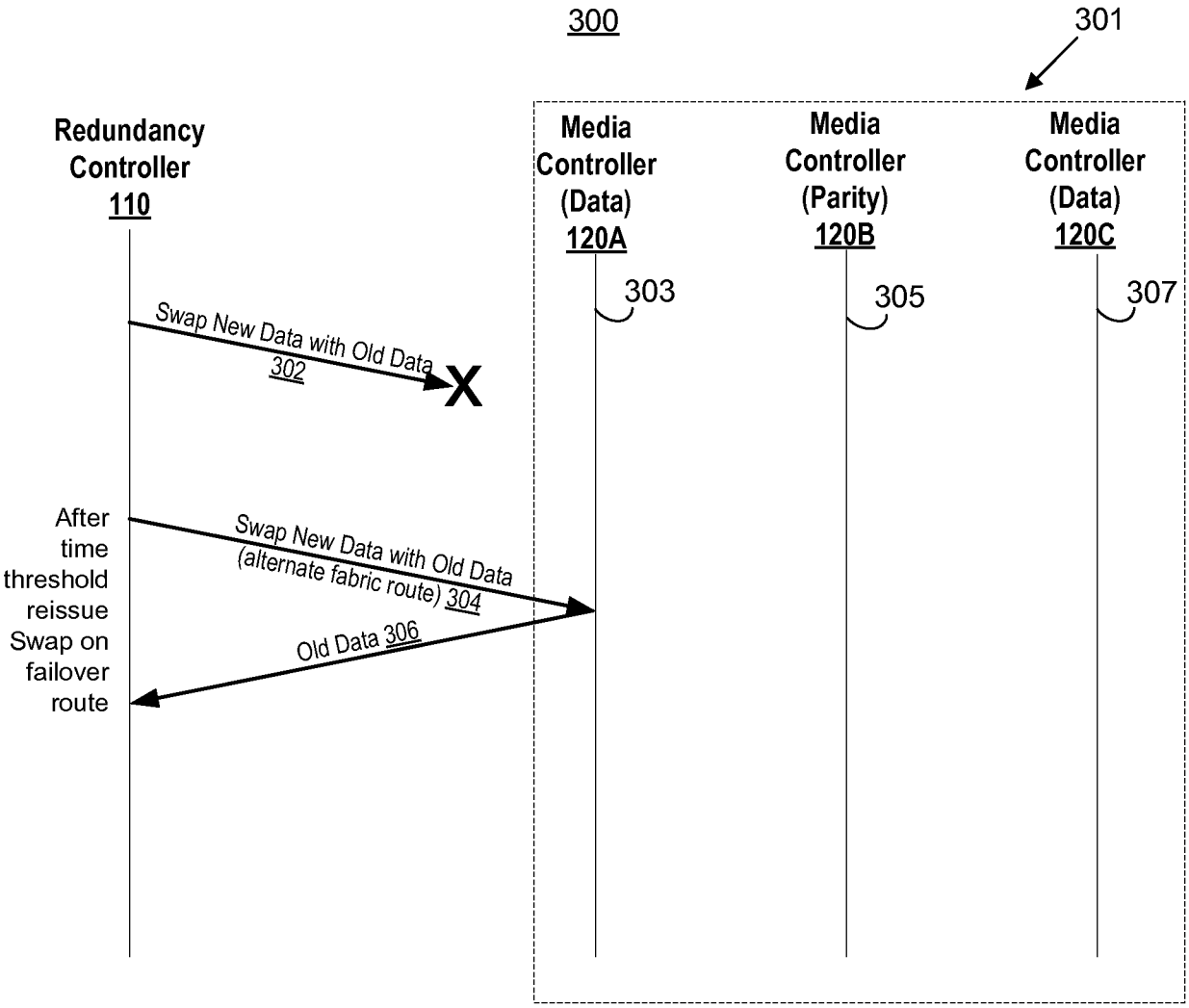

300
301
Redundancy Controller 110
Media Controller (Data) 120A
Media Controller (Parity) 120B
Media Controller (Data) 120C
303
305
307
Swap New Data with Old Data 302
X
After time threshold reissue Swap on failover route
Swap New Data with Old Data (alternate fabric route) 304
Old Data 306

FIG. 3

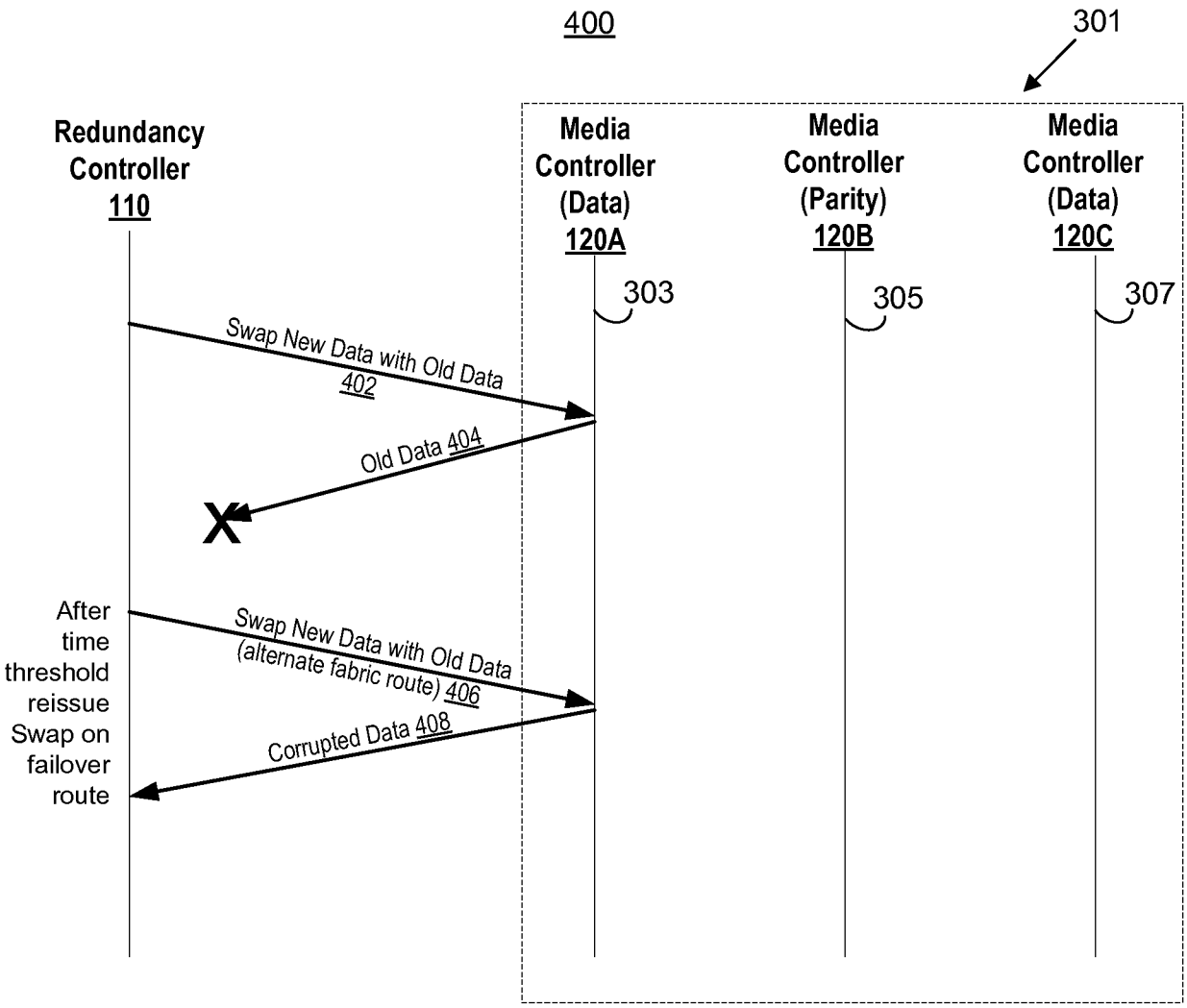

400
301
Redundancy Controller 110
Media Controller (Data) 120A
Media Controller (Parity) 120B
Media Controller (Data) 120C
303
305
307
Swap New Data with Old Data 402
Old Data 404
X
After time threshold reissue Swap on failover route
Swap New Data with Old Data (alternate fabric route) 406
Corrupted Data 408

FIG. 4

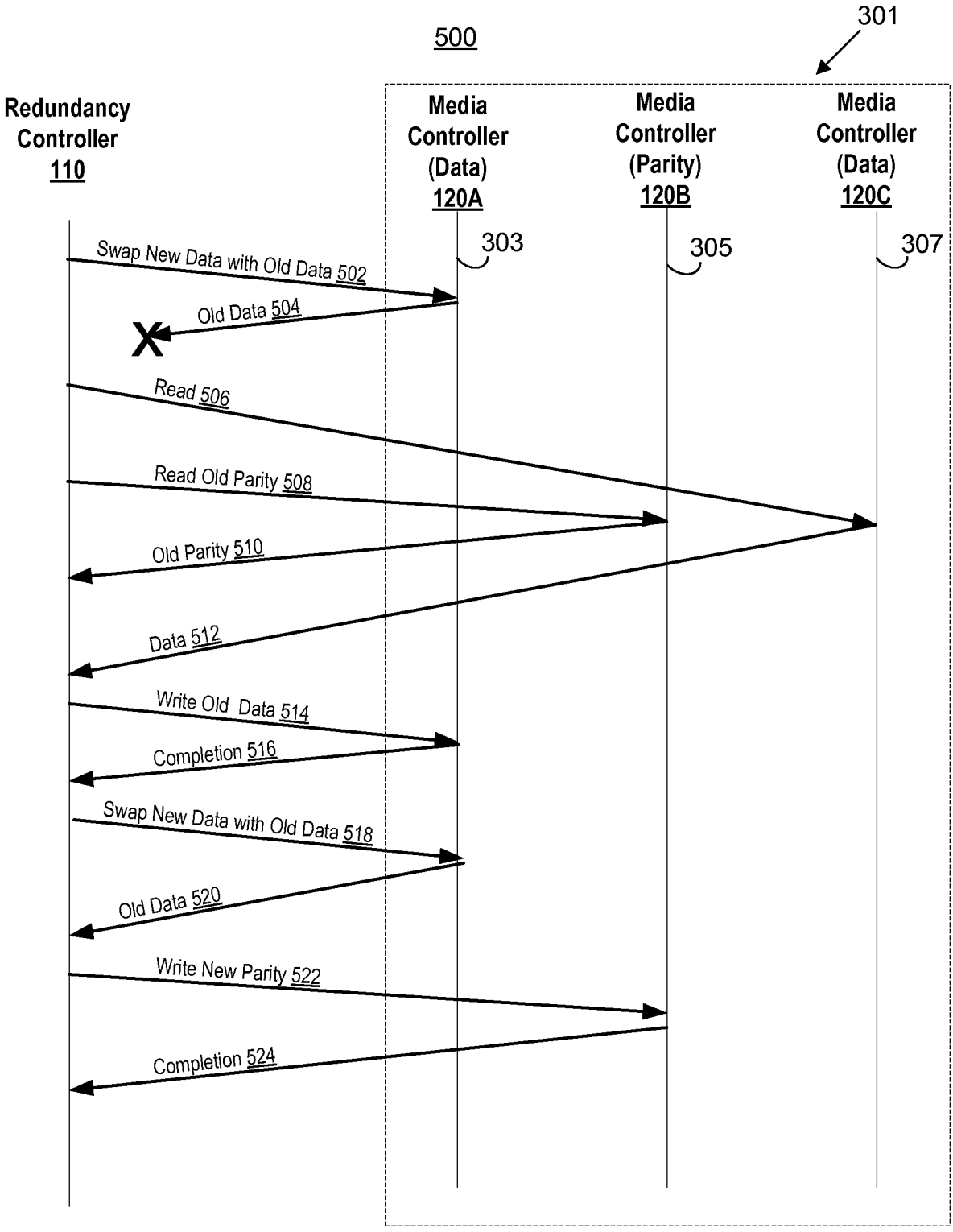

500
301
Redundancy Controller 110
Media Controller (Data) 120A
Media Controller (Parity) 120B
Media Controller (Data) 120C
303
305
307
Swap New Data with Old Data 502
Old Data 504
Read 506
Read Old Parity 508
Old Parity 510
Data 512
Write Old Data 514
Completion 516
Swap New Data with Old Data 518
Old Data 520
Write New Parity 522
Completion 524

FIG. 5

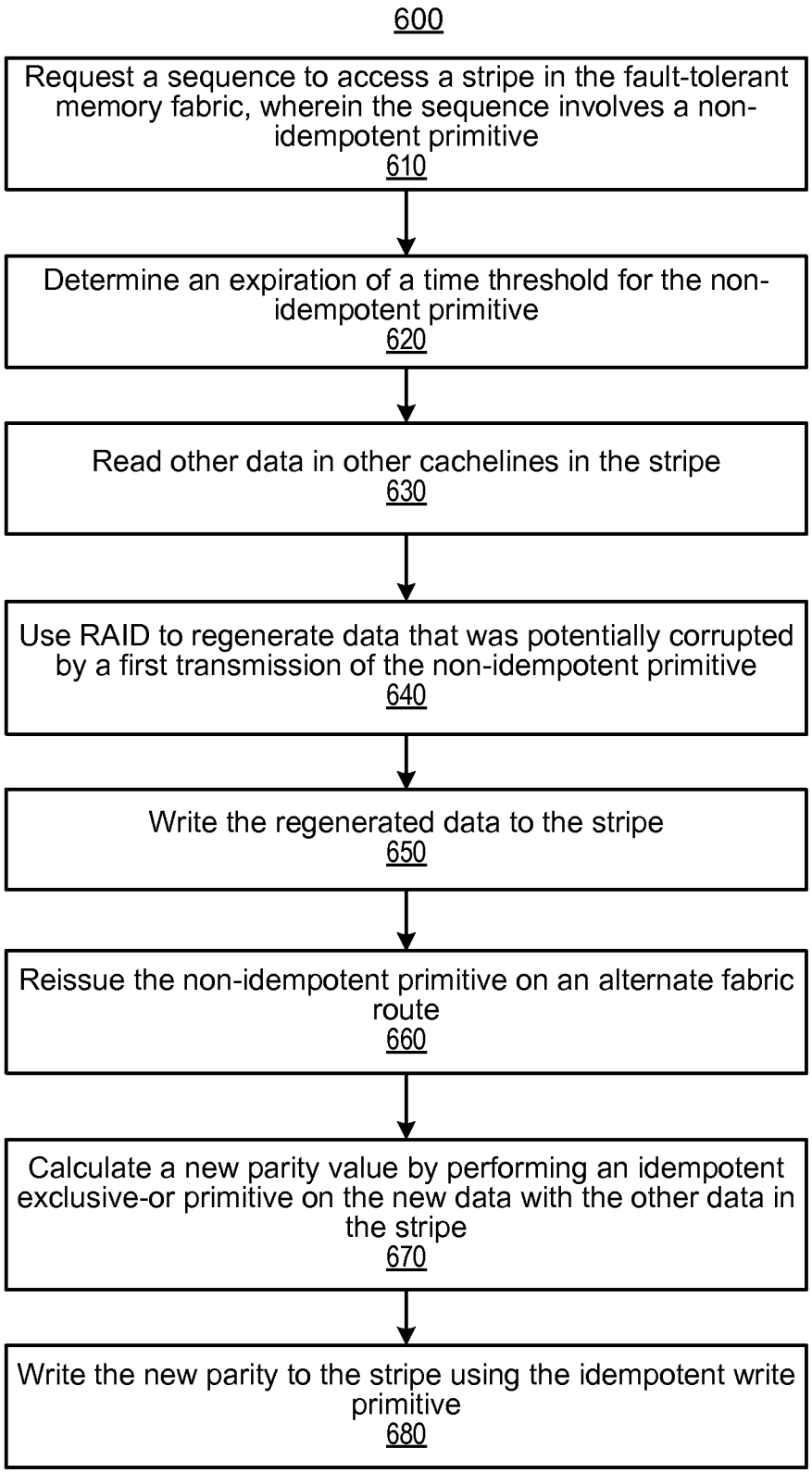

600
Request a sequence to access a stripe in the fault-tolerant memory fabric, wherein the sequence involves a non-idempotent primitive
610
Determine an expiration of a time threshold for the non-idempotent primitive
620
Read other data in other cachelines in the stripe
630
Use RAID to regenerate data that was potentially corrupted by a first transmission of the non-idempotent primitive
640
Write the regenerated data to the stripe
650
Reissue the non-idempotent primitive on an alternate fabric route
660
Calculate a new parity value by performing an idempotent exclusive-or primitive on the new data with the other data in the stripe
670
Write the new parity to the stripe using the idempotent write primitive
680

FIG. 6

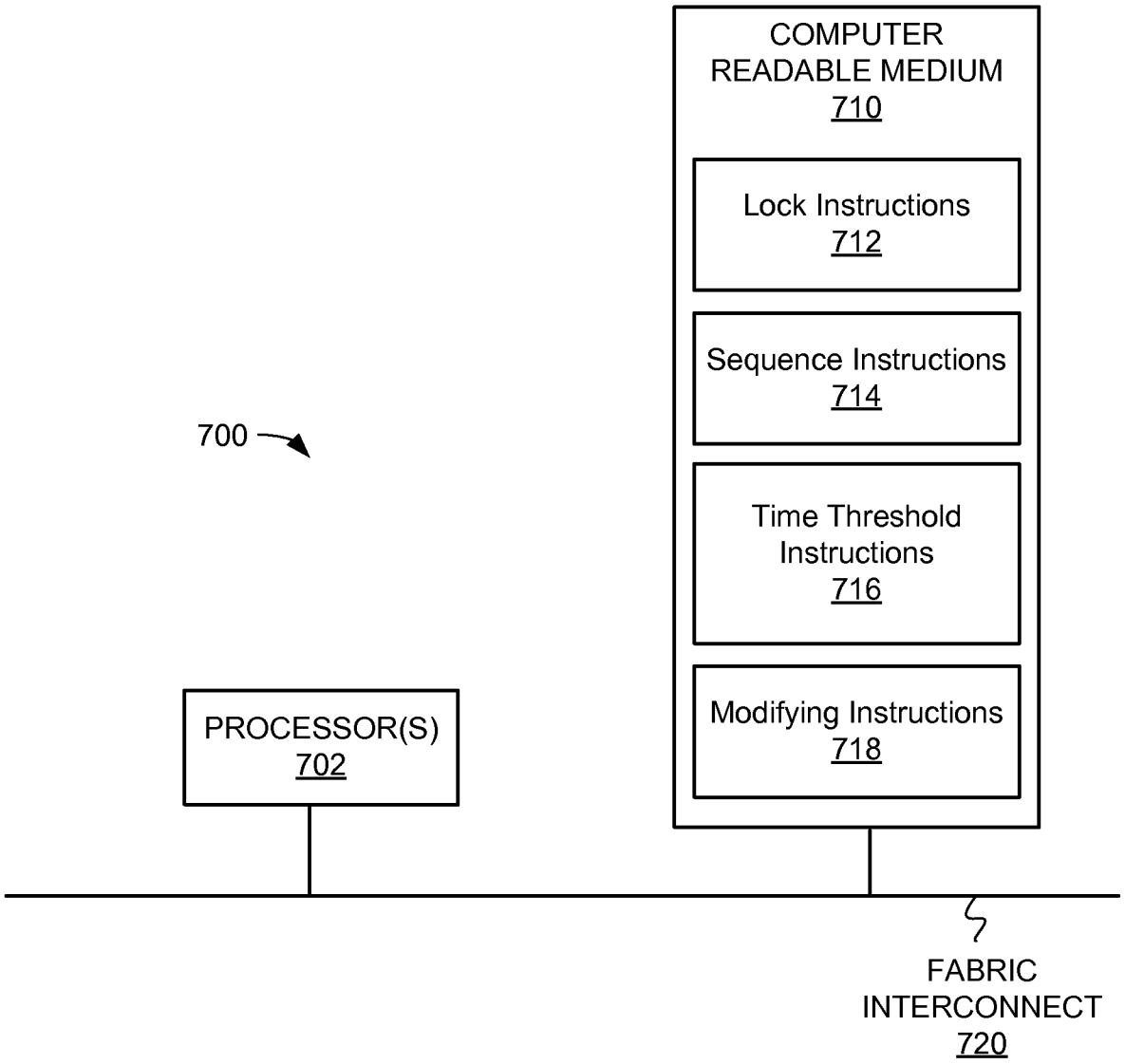

COMPUTER READABLE MEDIUM 710
Lock Instructions 712
Sequence Instructions 714
Time Threshold Instructions 716
Modifying Instructions 718
700
PROCESSOR(S) 702
FABRIC INTERCONNECT 720

FIG. 7

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2015/013898

A. CLASSIFICATION OF SUBJECT MATTER

G06F 12/16(2006.01)i, G06F 11/16(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 12/16; G06F 11/07; G06F 11/00; H04L 12/56; G06F 11/16

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: retransmission, non-idempotent, primitive, fault-tolerant, stripe, redundancy, regenerate, route

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2007-0028041 A1 (BASAVARAJ HALLYAL et al.) 01 February 2007 See paragraphs [0005], [0013], [0023]; claims 1, 9; and figure 2.	1-15
A	US 5633999 A (RICHARD F. CLOWES et al.) 27 May 1997 See column 15, lines 39-42; column 16, lines 38-41; claim 1; and figure 2.	1-15
A	US 2011-0246819 A1 (PAUL J. CALLAWAY et al.) 06 October 2011 See paragraphs [0057], [0082]-[0083]; and figures 4-5.	1-15
A	US 2011-0213928 A1 (GARY W. GRUBE et al.) 01 September 2011 See paragraphs [0092]-[0099]; and figures 6A-6C.	1-15
A	WO 02-091689 A1 (SUN MICROSYSTEMS, INC.) 14 November 2002 See page 2, lines 24-34; page 10, line 34 - page 11, line 23; and figure 11.	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	Date of mailing of the international search report
04 September 2015 (04.09.2015)	**08 October 2015 (08.10.2015)**

Name and mailing address of the ISA/KR
International Application Division
Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon Metropolitan City, 35208,
Republic of Korea
Facsimile No. +82-42-472-7140

Authorized officer
BYUN, Sung Cheal
Telephone No. +82-42-481-8262

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

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.
PCT/US2015/013898

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2007-0028041 A1	01/02/2007	None	
US 05633999 A	27/05/1997	GB 2251502 A	08/07/1992
		US 5748870 A	05/05/1998
US 2011-0246819 A1	06/10/2011	CA 2657882 A1	21/02/2008
		CA 2657882 C	07/02/2012
		EP 2049995 A2	22/04/2009
		EP 2049995 A4	11/11/2009
		US 2008-126853 A1	29/05/2008
		US 2009-106328 A1	23/04/2009
		US 2013-297970 A1	07/11/2013
		US 7480827 B2	20/01/2009
		US 7975173 B2	05/07/2011
		US 8468390 B2	18/06/2013
		WO 2008-021636 A2	21/02/2008
		WO 2008-021636 A3	02/10/2008
US 2011-0213928 A1	01/09/2011	US 2011-213929 A1	01/09/2011
		US 2011-214011 A1	01/09/2011
		US 2014-250270 A1	04/09/2014
		US 2014-351631 A1	27/11/2014
		US 2014-351633 A1	27/11/2014
		US 2014-380128 A1	25/12/2014
		US 8725940 B2	13/05/2014
		US 8850113 B2	30/09/2014
WO 02-091689 A1	14/11/2002	US 2003-0023893 A1	30/01/2003
		US 6883108 B2	19/04/2005