US 20210182190A1

(19) United States

(12) Patent Application Publication

(10) Pub. No.: US 2021/0182190 A1

(43) Pub. Date: Jun. 17, 2021

Gao et al.

(54) **INTELLIGENT DIE AWARE STORAGE DEVICE SCHEDULER**

(71) Applicant: **Pure Storage, Inc.**, Mountain View, CA (US)

(72) Inventors: **Ying Gao**, San Jose, CA (US); **Boris Feigin**, San Francisco, CA (US); **Hari Kannan**, Sunnyvale, CA (US)

(21) Appl. No.: **17/159,986**

(22) Filed: **Jan. 27, 2021**

Related U.S. Application Data

(63) Continuation-in-part of application No. 15/336,618, filed on Oct. 27, 2016.

(60) Provisional application No. 62/365,864, filed on Jul. 22, 2016.

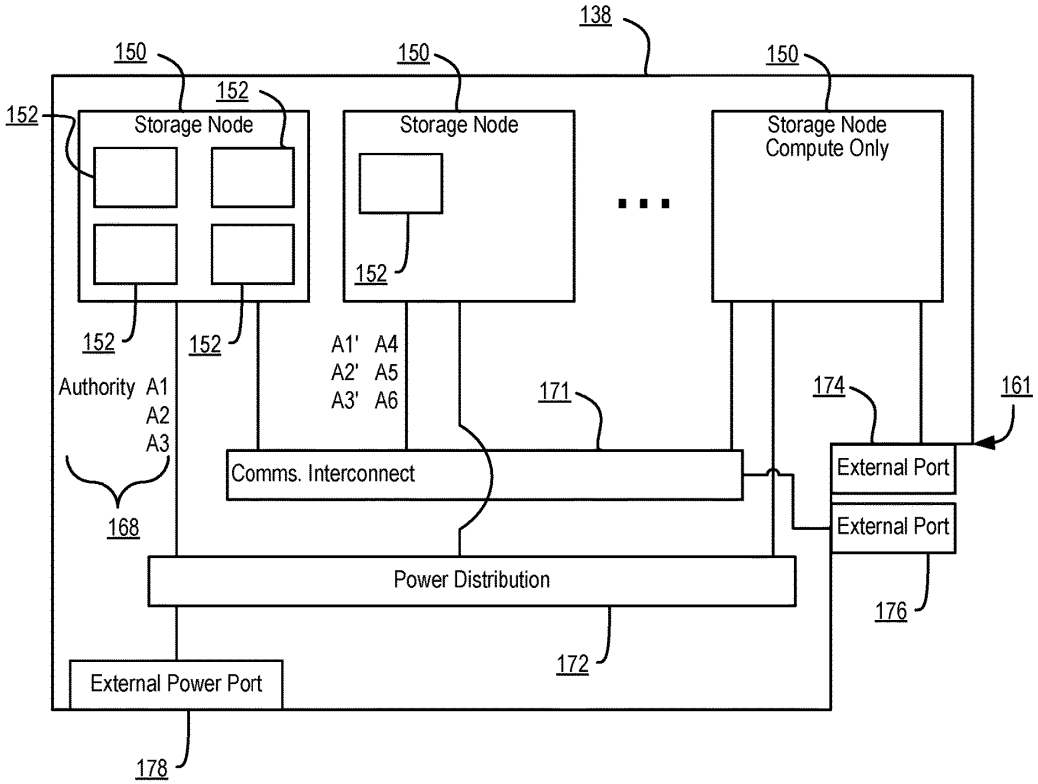
Publication Classification

(51) **Int. Cl.**
G06F 12/02 (2006.01)

(52) **U.S. Cl.**
CPC ***G06F 12/0269*** (2013.01); *G06F 2212/702* (2013.01)

(57) **ABSTRACT**

A scheduling system for a memory controller is provided. The system includes operation queues and a scheduler. The scheduler receives operation requests, prioritizes each operation request according to one or more policies, and inserts each operation request into an operation queue.



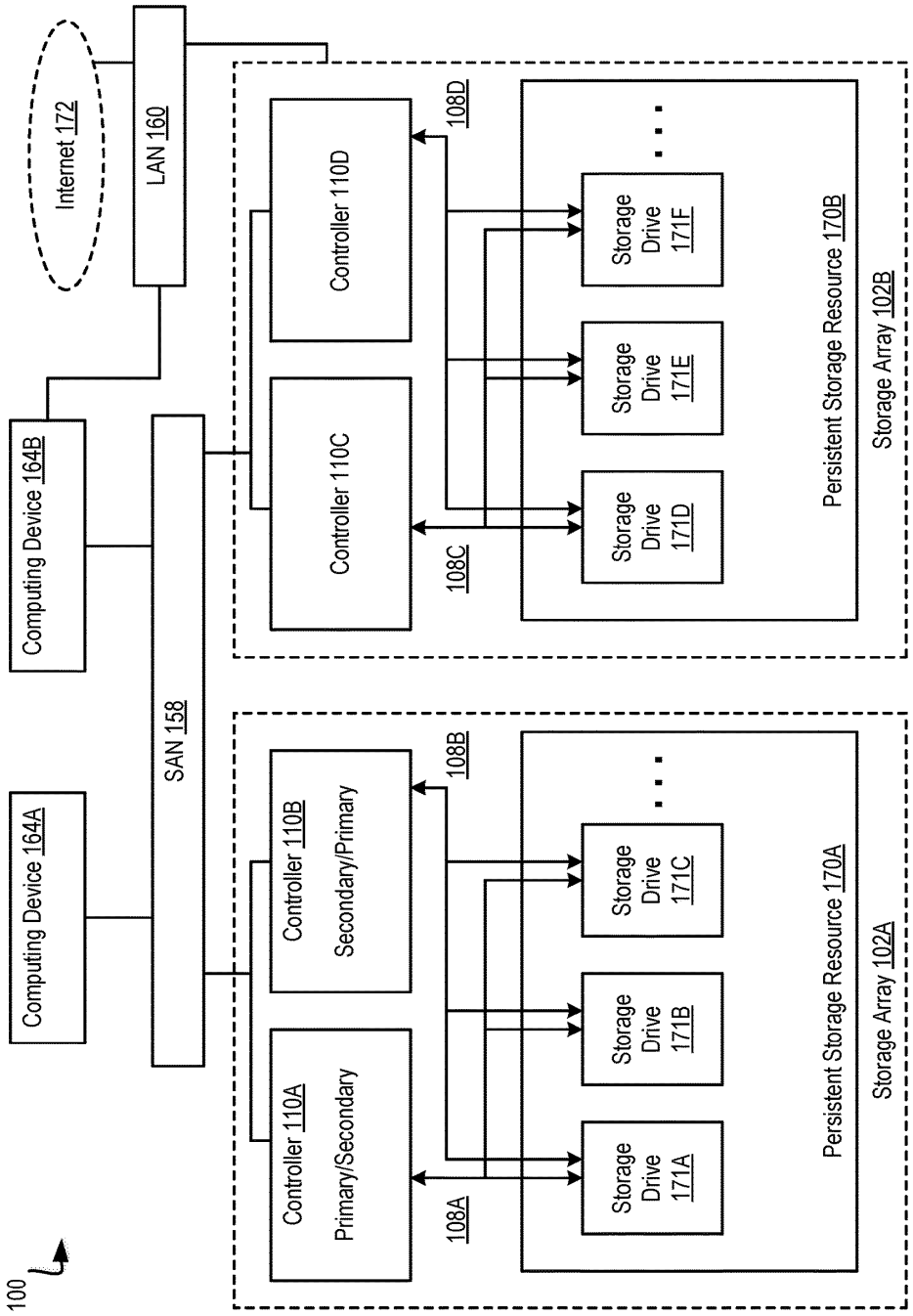

100
Computing Device 164A
Computing Device 164B
Internet 172
LAN 160
SAN 158
Controller 110A
Primary/Secondary
Controller 110B
Secondary/Primary
Controller 110C
Controller 110D
108A
108B
108C
108D
Storage Drive 171A
Storage Drive 171B
Storage Drive 171C
Storage Drive 171D
Storage Drive 171E
Storage Drive 171F
Persistent Storage Resource 170A
Persistent Storage Resource 170B
Storage Array 102A
Storage Array 102B

FIG. 1A

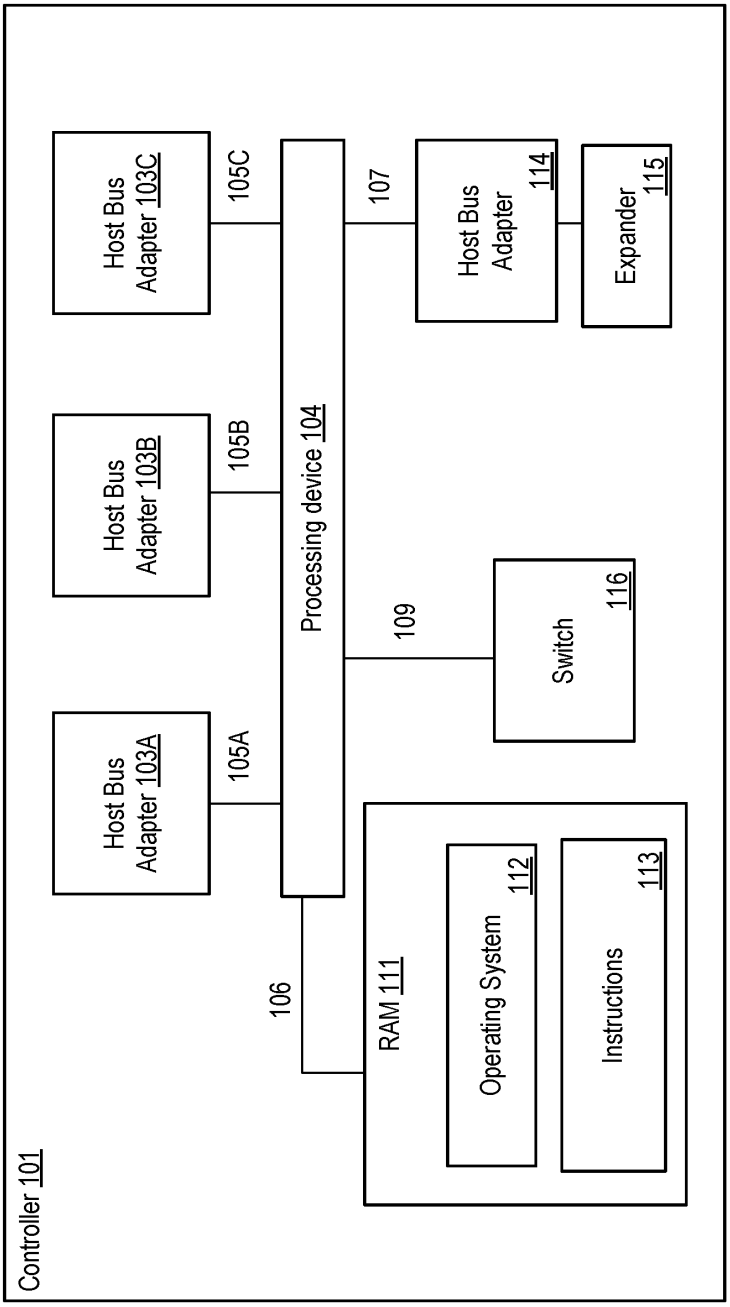

Controller 101
Host Bus Adapter 103A
Host Bus Adapter 103B
Host Bus Adapter 103C
105A
105B
105C
Processing device 104
106
107
109
RAM 111
Operating System 112
Instructions 113
Host Bus Adapter 114
Expander 115
Switch 116

FIG. 1B

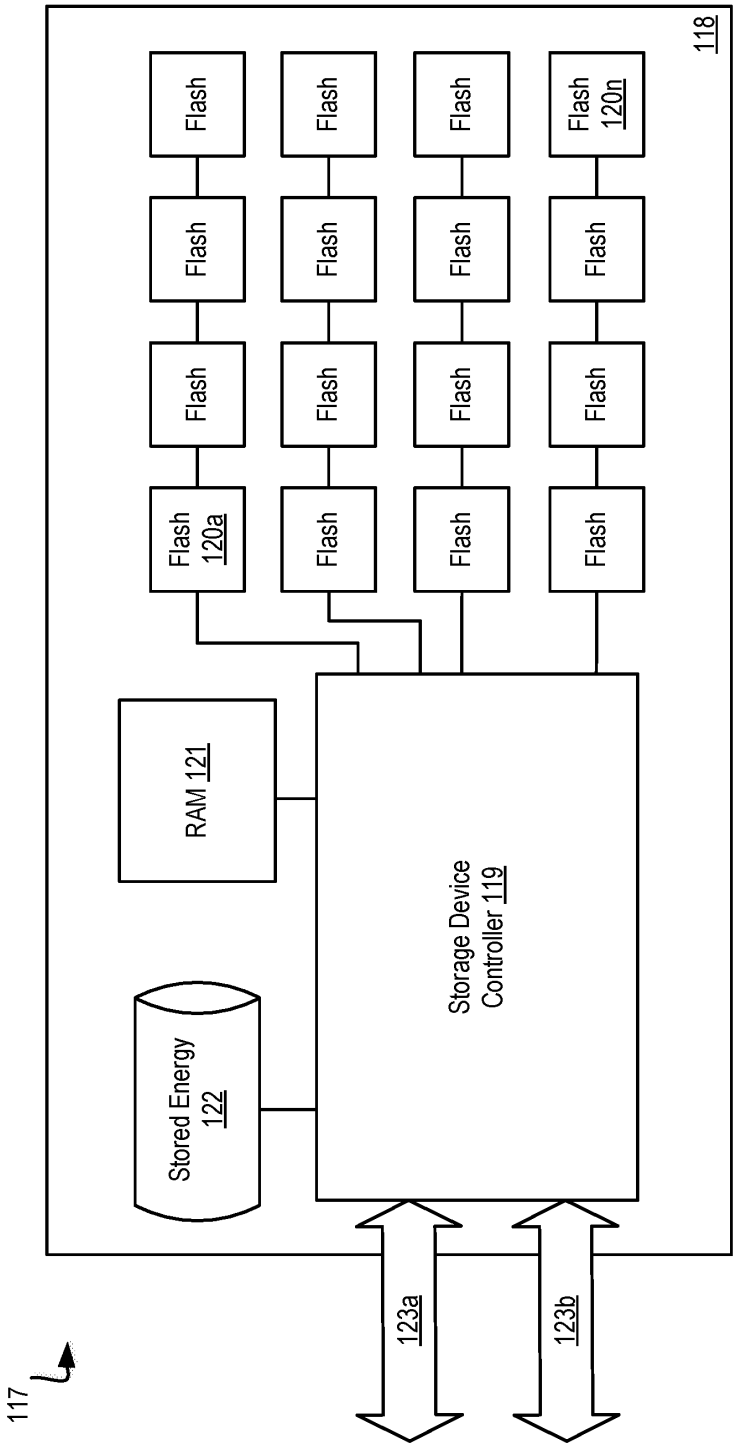

118
Flash
Flash
Flash
Flash 120a
Flash
Flash
Flash
Flash
Flash
Flash
Flash
Flash
Flash
Flash
Flash
Flash 120n
RAM 121
Stored Energy 122
Storage Device Controller 119
123a
123b
117

FIG. 1C

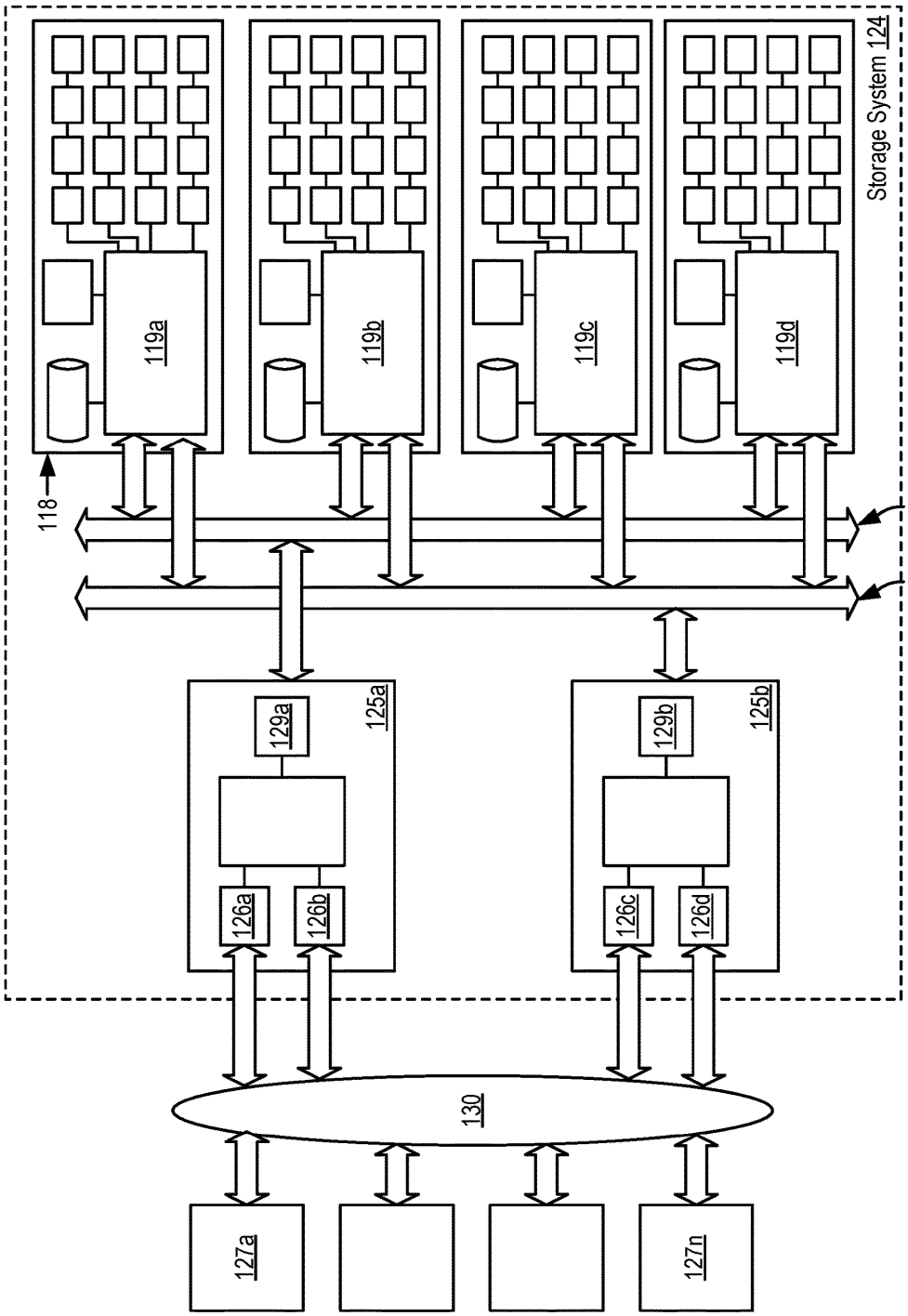

Storage System 124
119a
119b
119c
119d
118
128b
128a
125a
125b
129a
129b
126a
126b
126c
126d
130
127a
127n

FIG. 1D

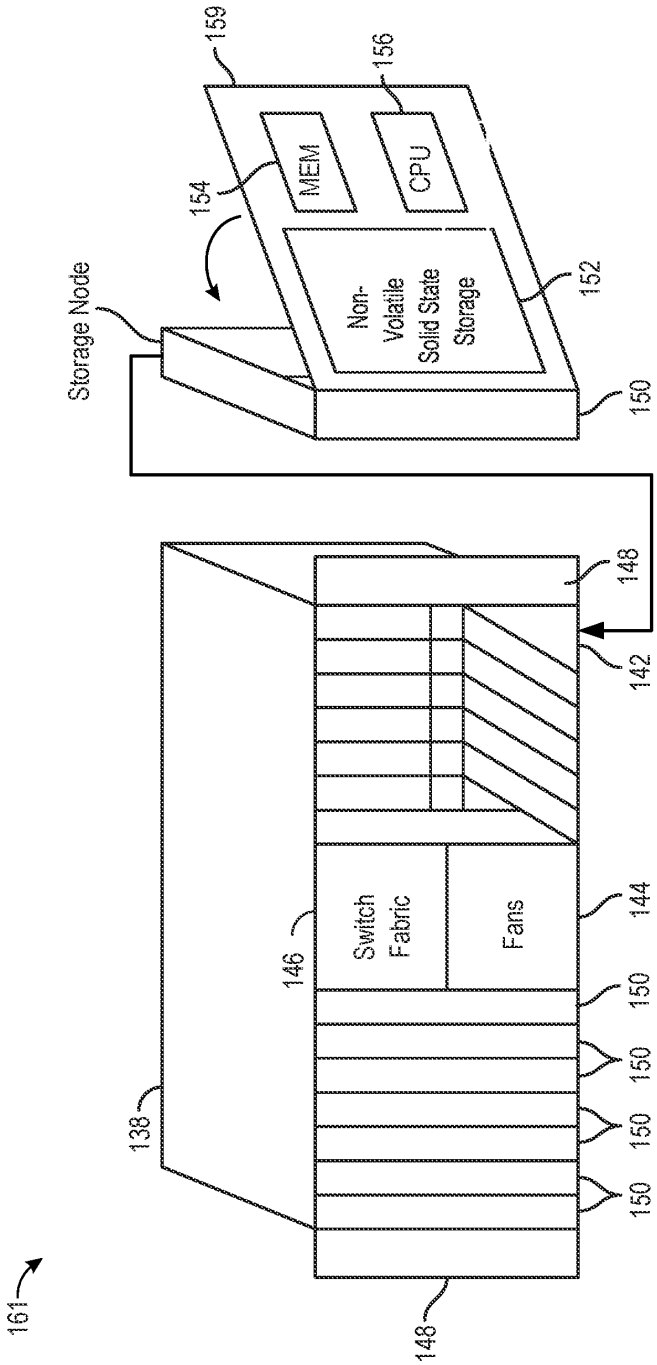

159
156
MEM
CPU
154
Storage Node
Non-
Volatile
Solid State
Storage
152
150
148
142
Switch
Fabric
Fans
146
144
150
150
150
150
138
161
148

FIG. 2A

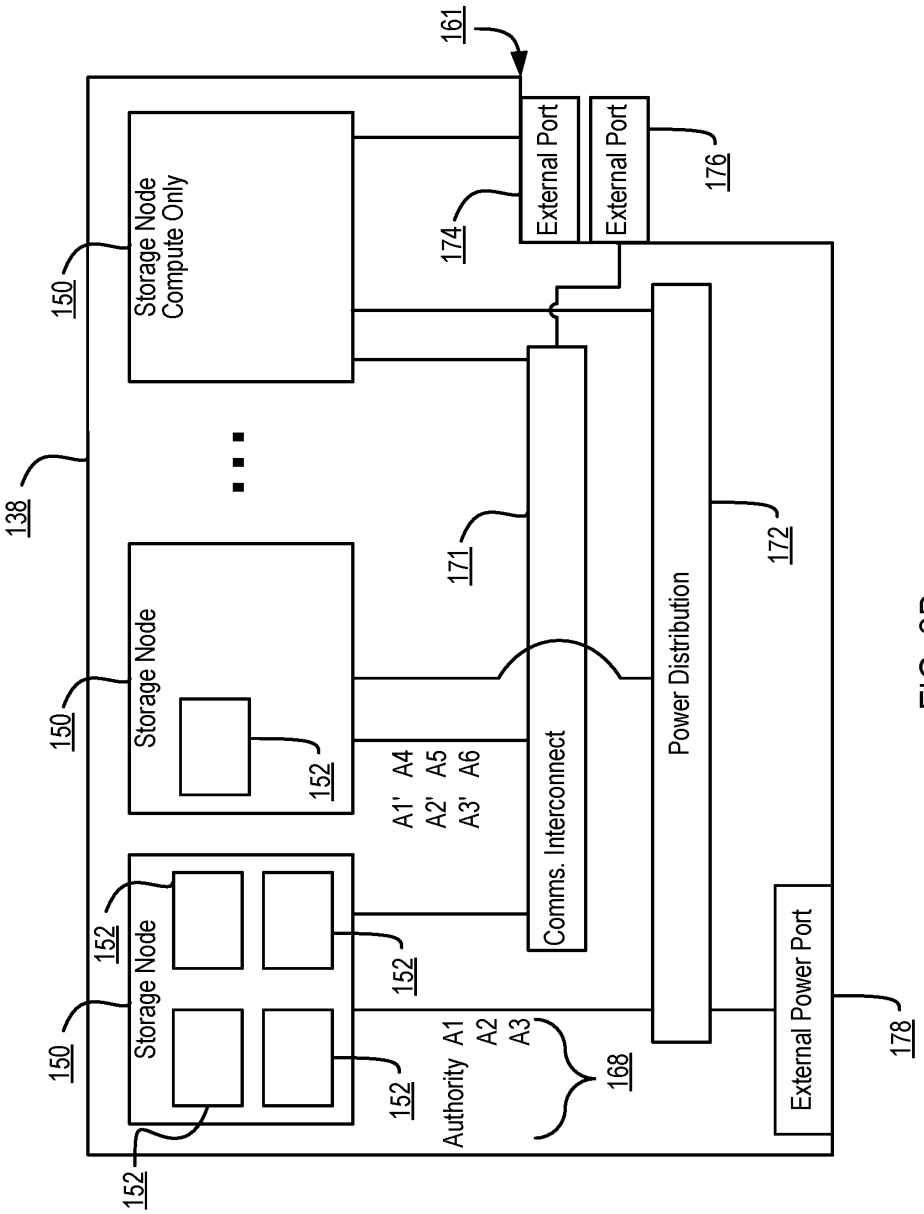

161
138
150
Storage Node
Compute Only
150
Storage Node
152
150
Storage Node
152
152
152
152
A1' A4
A2' A5
A3' A6
Authority A1
A2
A3
168
171
Comms. Interconnect
174
External Port
External Port
176
Power Distribution
172
External Power Port
178

FIG. 2B

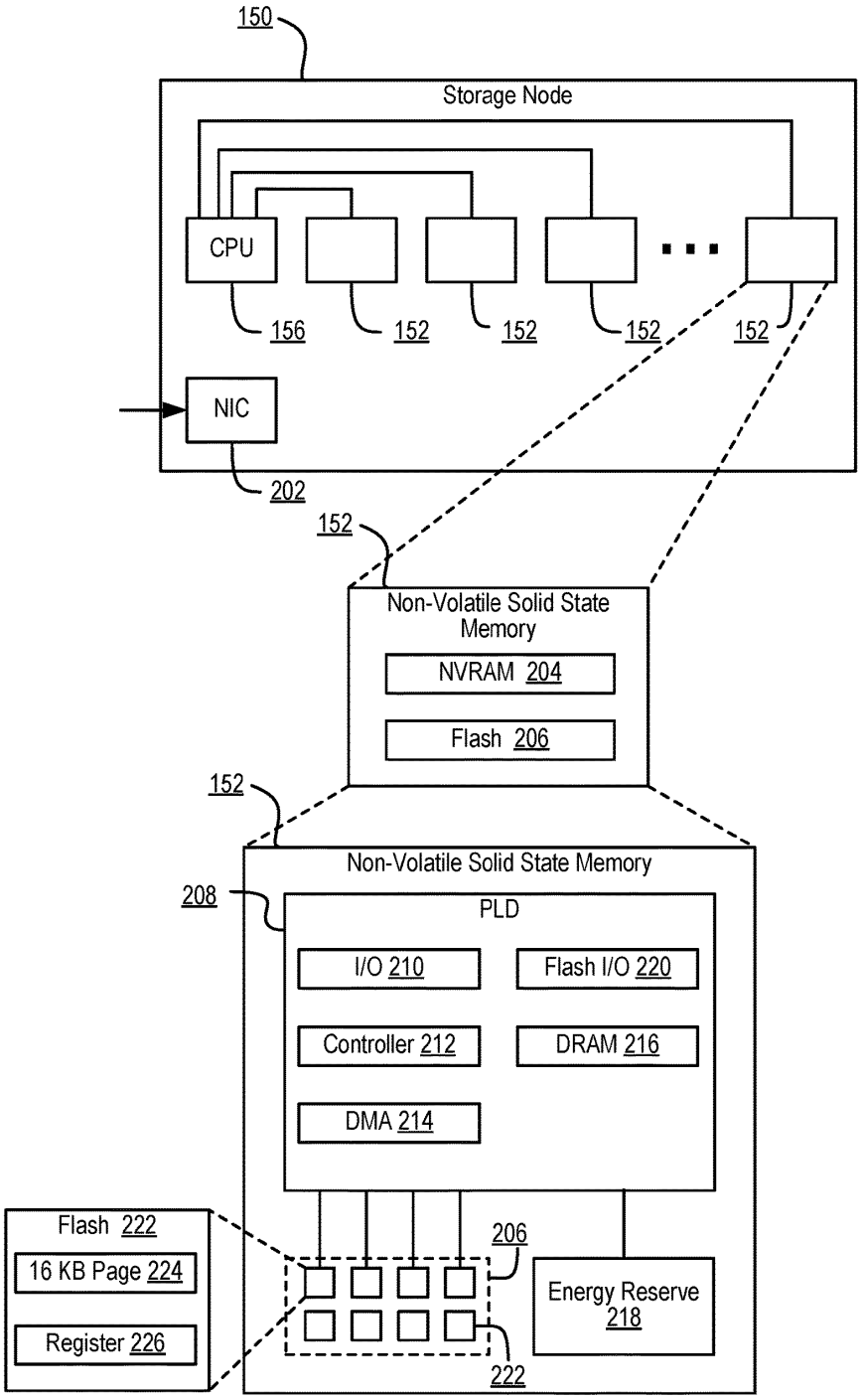

150
Storage Node
CPU
156
152
152
152
152
NIC
202
152
Non-Volatile Solid State Memory
NVRAM 204
Flash 206
152
Non-Volatile Solid State Memory
208
PLD
I/O 210
Flash I/O 220
Controller 212
DRAM 216
DMA 214
Flash 222
16 KB Page 224
Register 226
206
Energy Reserve 218
222

FIG. 2C

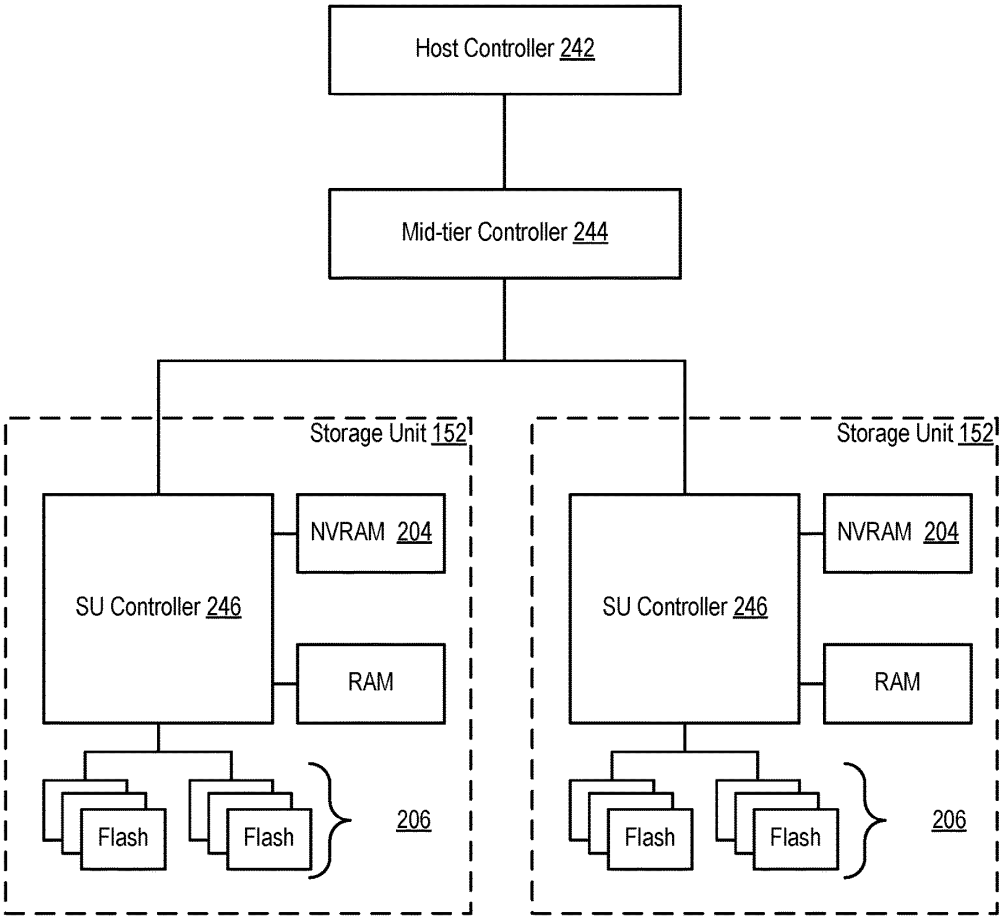

Host Controller 242
Mid-tier Controller 244
Storage Unit 152
SU Controller 246
NVRAM 204
RAM
Flash
Flash
206
Storage Unit 152
SU Controller 246
NVRAM 204
RAM
Flash
Flash
206

FIG. 2D

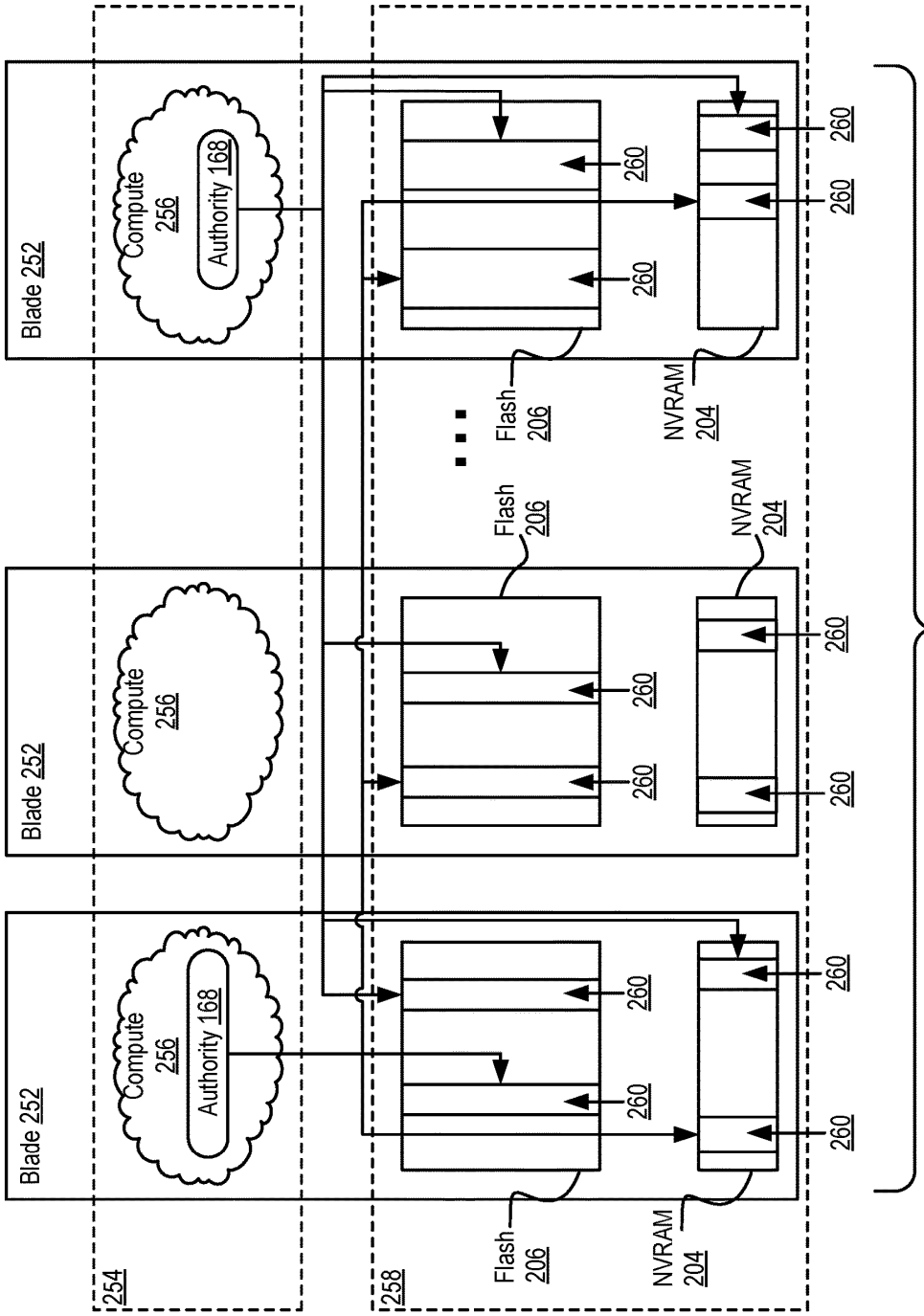

Blade 252
Compute 256
Authority 168
Blade 252
Compute 256
Blade 252
Compute 256
Authority 168
254
258
Flash 206
NVRAM 204
Flash 206
NVRAM 204
Flash 206
NVRAM 204
260
161

FIG. 2E

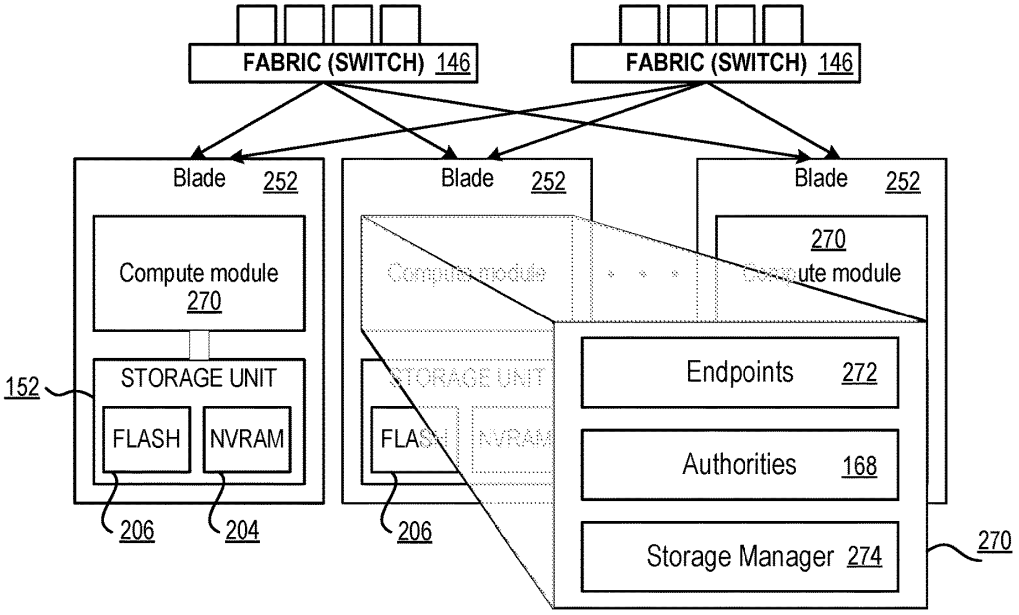

FABRIC (SWITCH) 146
FABRIC (SWITCH) 146
Blade 252
Blade 252
Blade 252
Compute module 270
STORAGE UNIT
FLASH
NVRAM
152
206
204
206
270
Endpoints 272
Authorities 168
Storage Manager 274
270

FIG. 2F

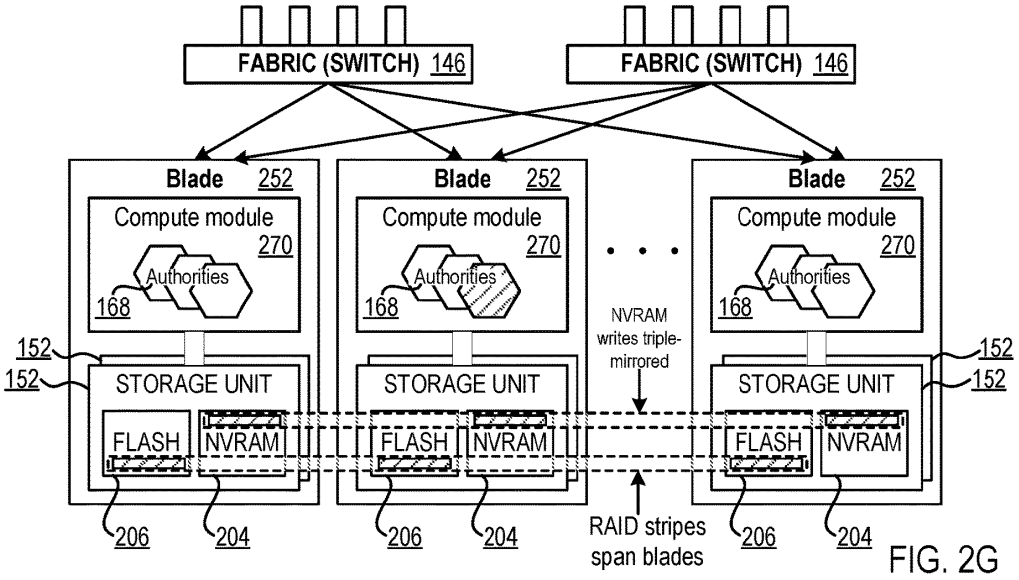

FABRIC (SWITCH) 146
FABRIC (SWITCH) 146
Blade 252
Blade 252
Blade 252
Compute module 270
Authorities 168
STORAGE UNIT
FLASH
NVRAM
152
152
NVRAM writes triple-mirrored
RAID stripes span blades
206
204

FIG. 2G

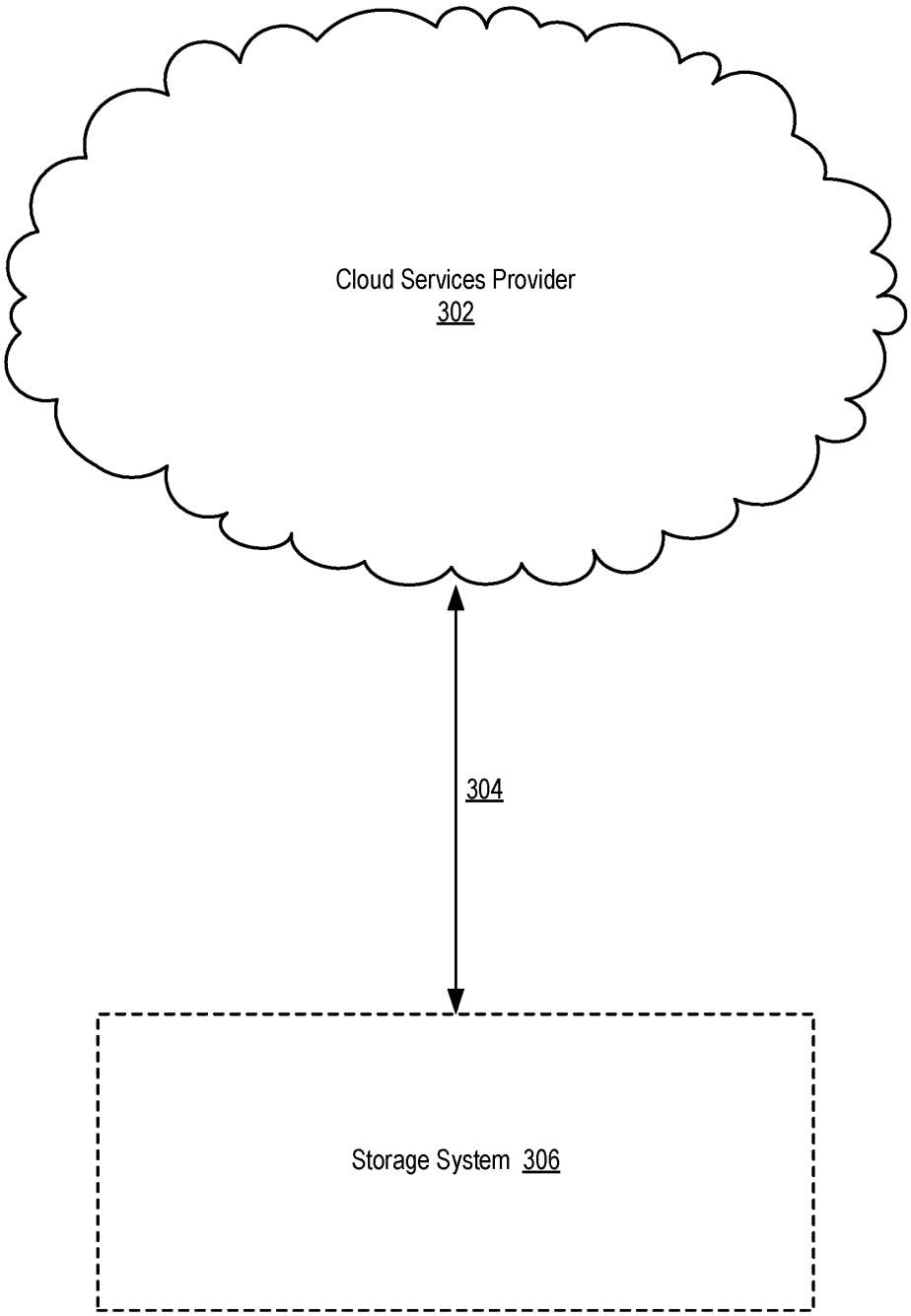


FIG. 3A

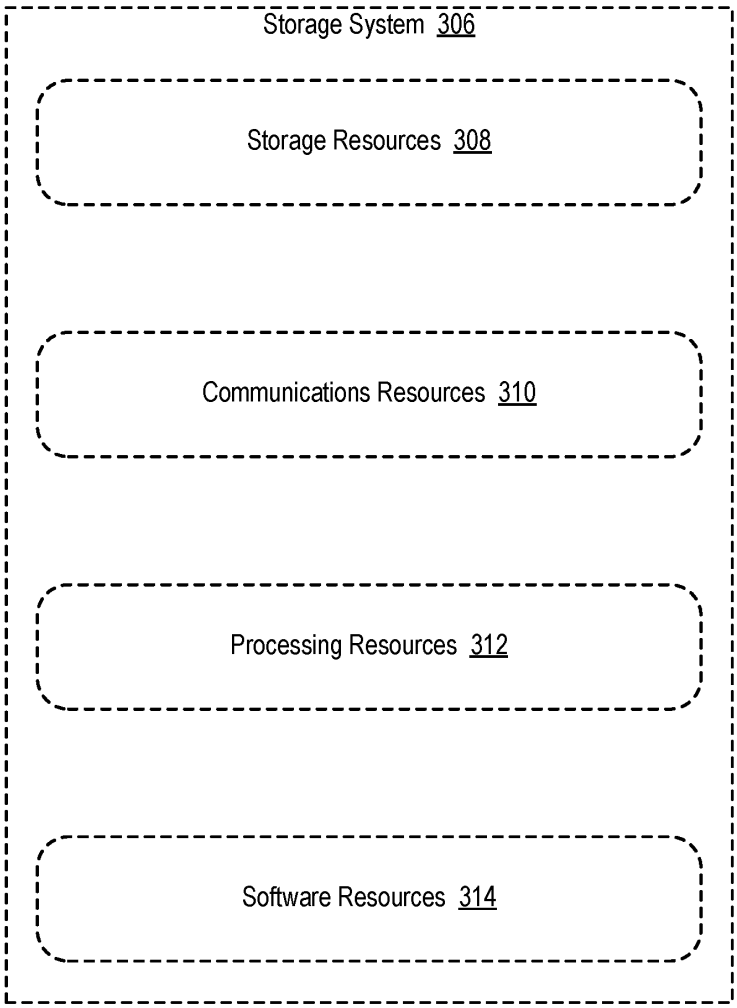

Storage System 306
Storage Resources 308
Communications Resources 310
Processing Resources 312
Software Resources 314

FIG. 3B

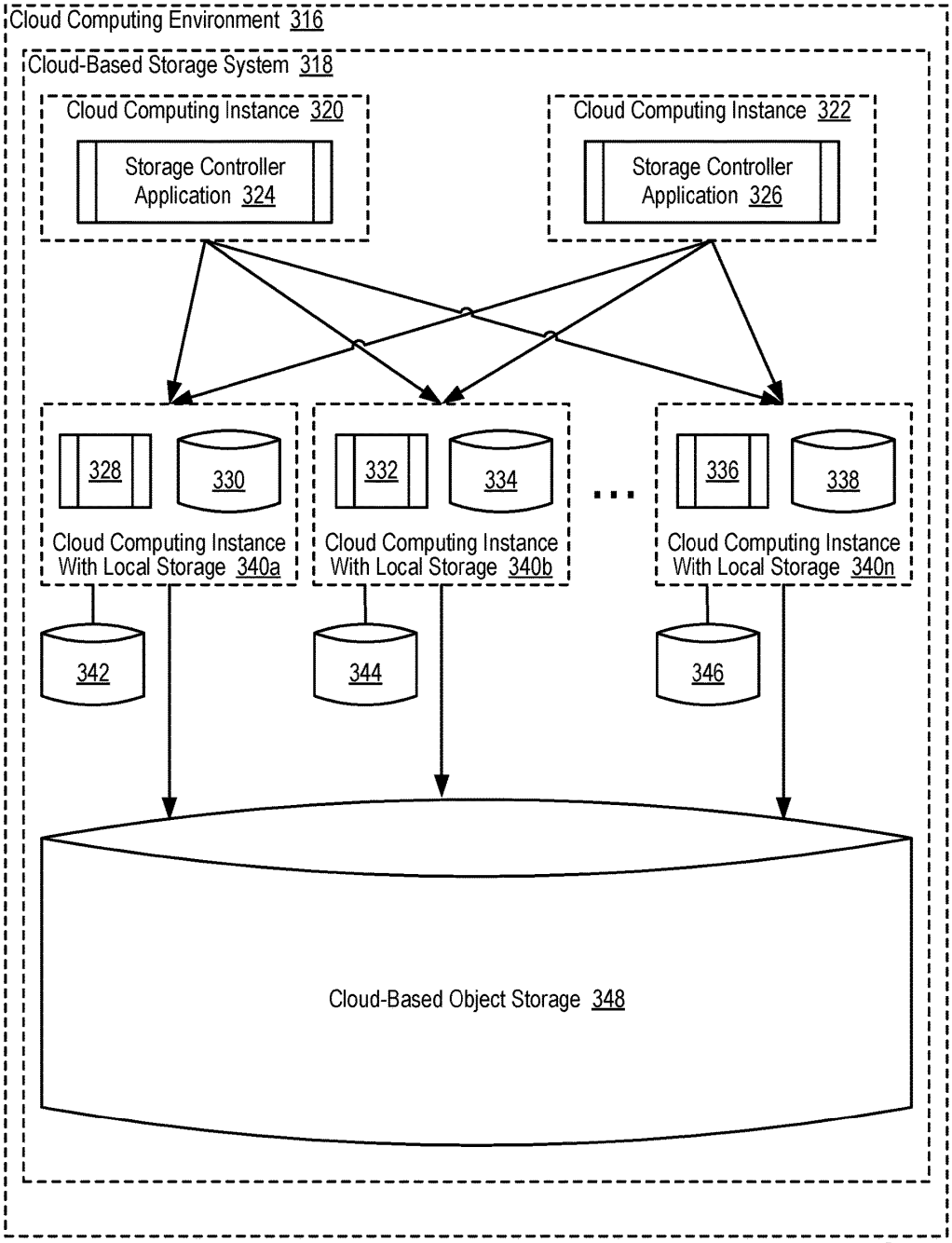

Cloud Computing Environment 316
Cloud-Based Storage System 318
Cloud Computing Instance 320
Storage Controller Application 324
Cloud Computing Instance 322
Storage Controller Application 326
328
330
332
334
336
338
Cloud Computing Instance With Local Storage 340a
Cloud Computing Instance With Local Storage 340b
Cloud Computing Instance With Local Storage 340n
342
344
346
Cloud-Based Object Storage 348

FIG. 4

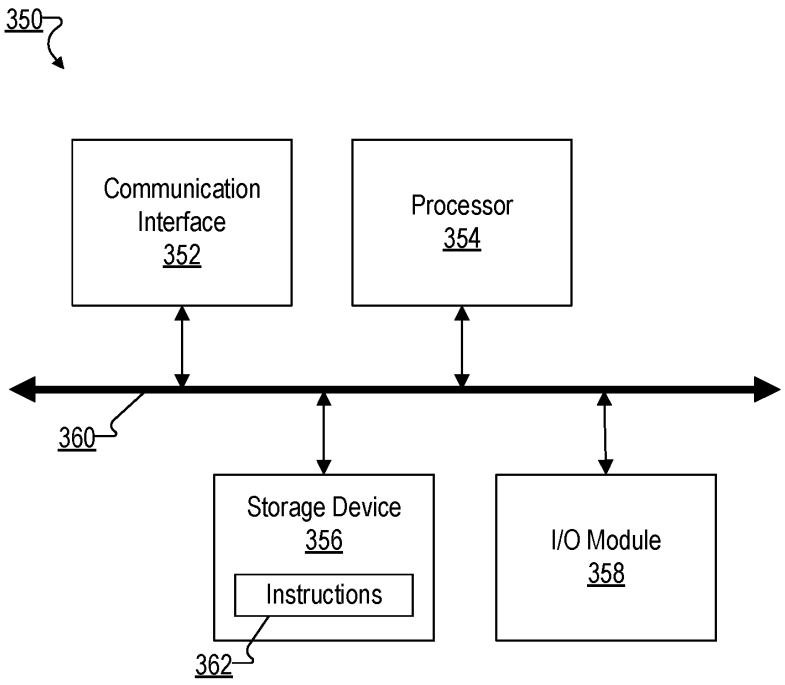

350
Communication Interface 352
Processor 354
360
Storage Device 356
Instructions
362
I/O Module 358

FIG. 5

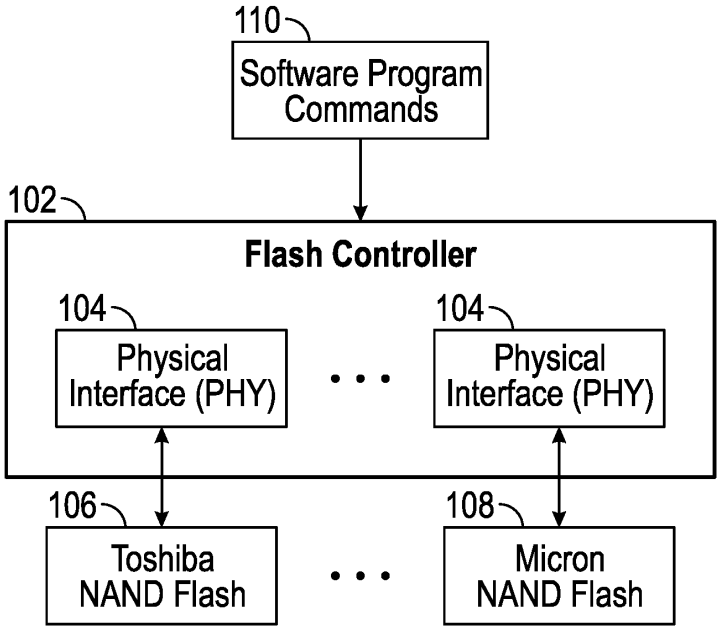

110
Software Program Commands
102
Flash Controller
104
Physical Interface (PHY)
104
Physical Interface (PHY)
106
Toshiba NAND Flash
108
Micron NAND Flash

FIG. 6

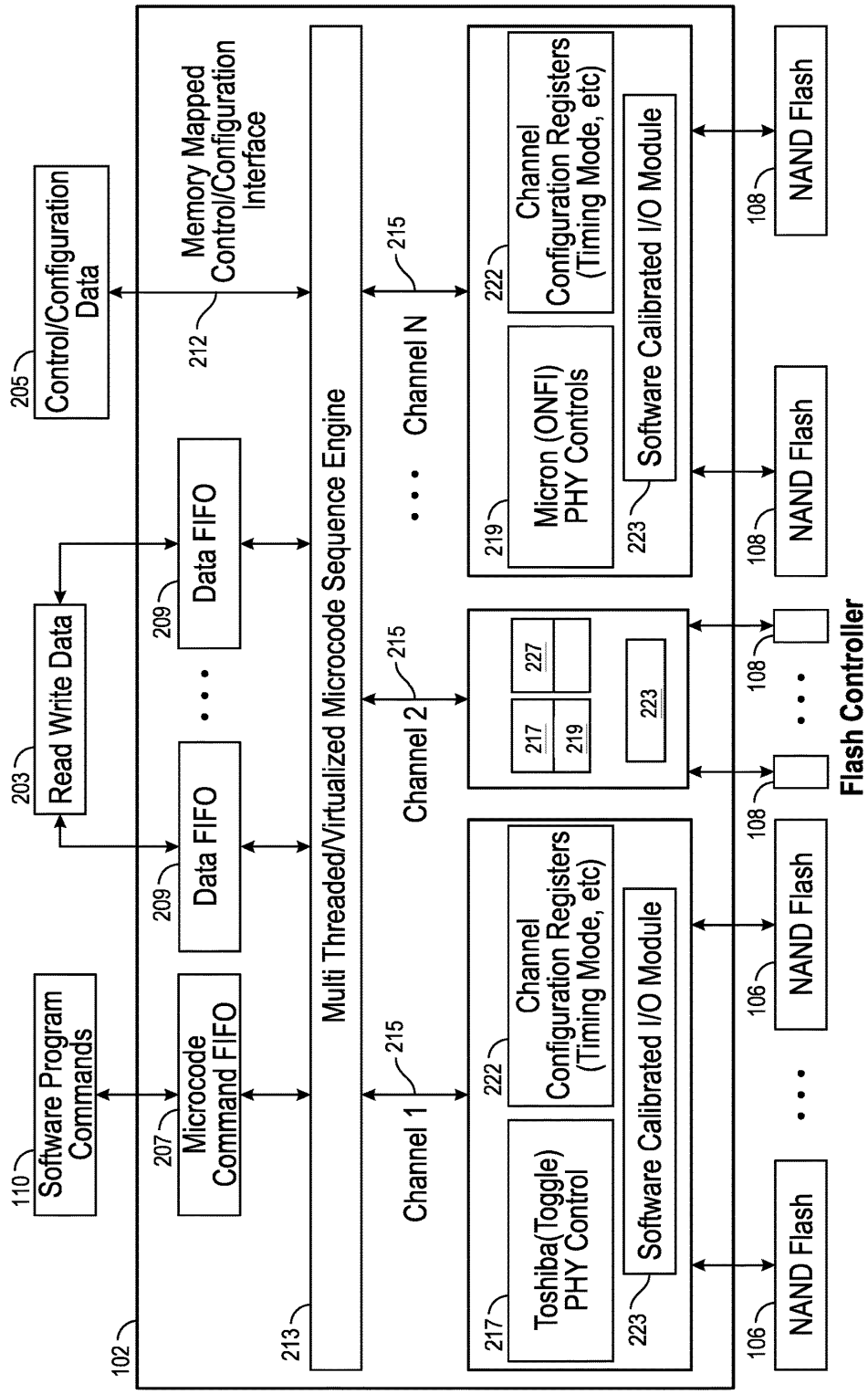

110
Software Program Commands
203
Read Write Data
205
Control/Configuration Data
102
207
Microcode Command FIFO
209
Data FIFO
209
Data FIFO
212
Memory Mapped Control/Configuration Interface
213
Multi Threaded/Virtualized Microcode Sequence Engine
215
Channel 1
215
Channel 2
215
Channel N
217
Toshiba(Toggle) PHY Control
222
Channel Configuration Registers (Timing Mode, etc)
223
Software Calibrated I/O Module
217
219
227
223
219
Micron (ONFI) PHY Controls
222
Channel Configuration Registers (Timing Mode, etc)
223
Software Calibrated I/O Module
106
NAND Flash
106
NAND Flash
108
NAND Flash
108
NAND Flash
108
108
Flash Controller

FIG. 7A

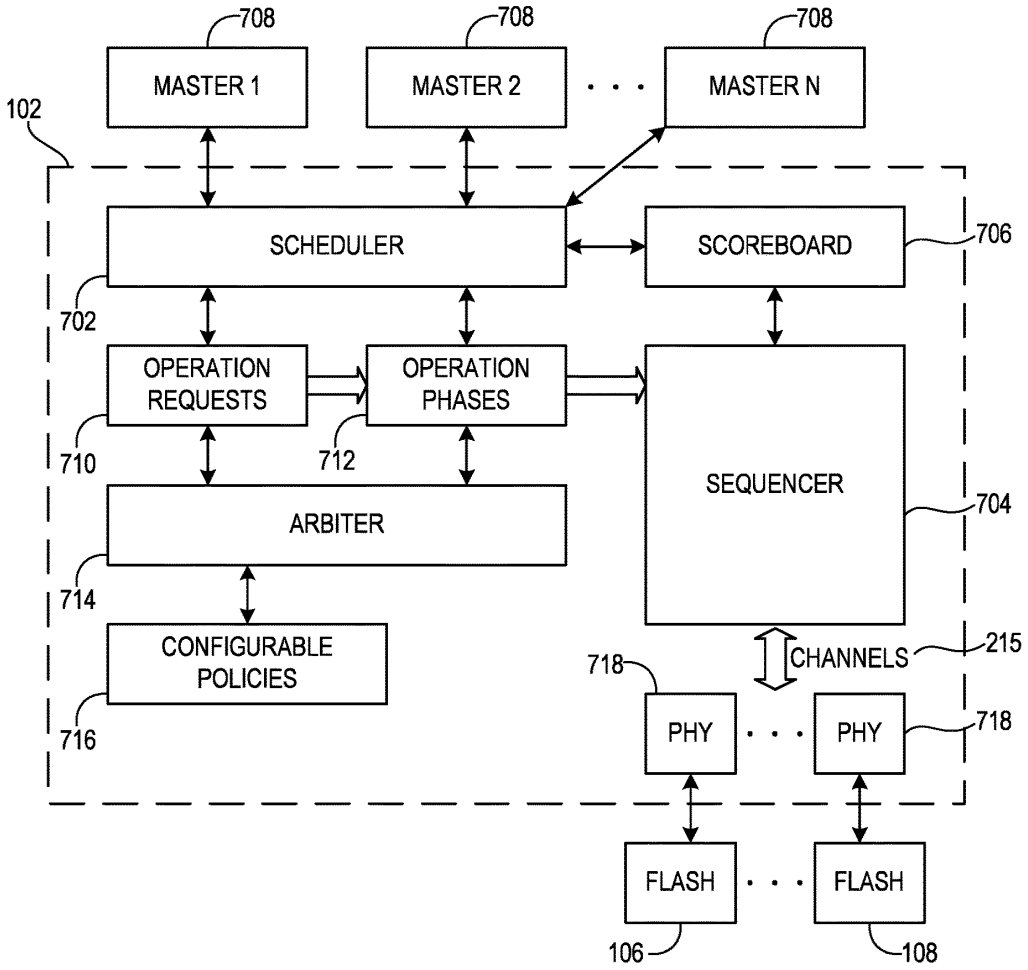

708
708
708
MASTER 1
MASTER 2
MASTER N
102
SCHEDULER
SCOREBOARD
706
702
OPERATION REQUESTS
OPERATION PHASES
710
712
SEQUENCER
704
ARBITER
714
CONFIGURABLE POLICIES
716
CHANNELS
215
718
PHY
PHY
718
FLASH
FLASH
106
108

FIG. 7B

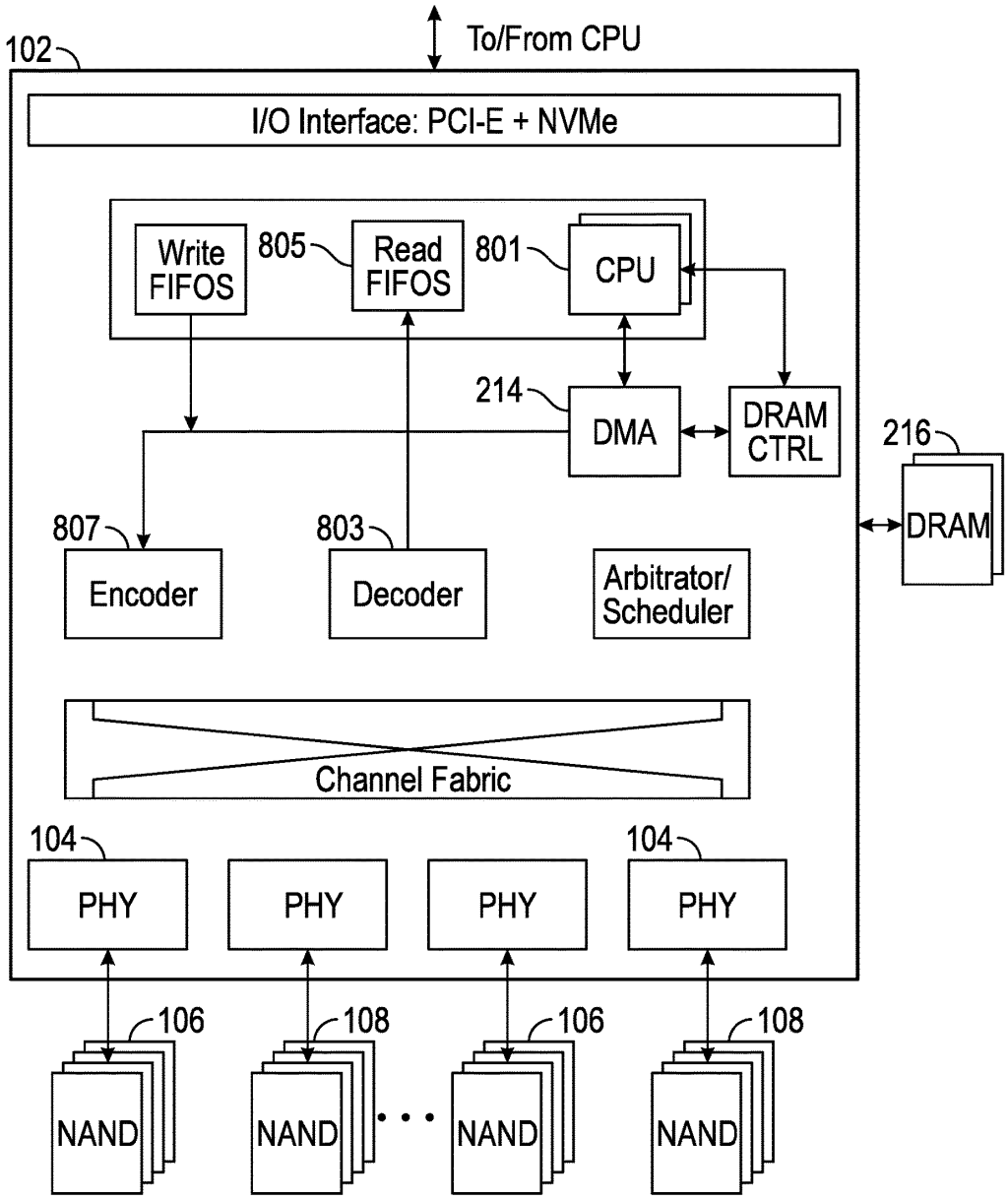
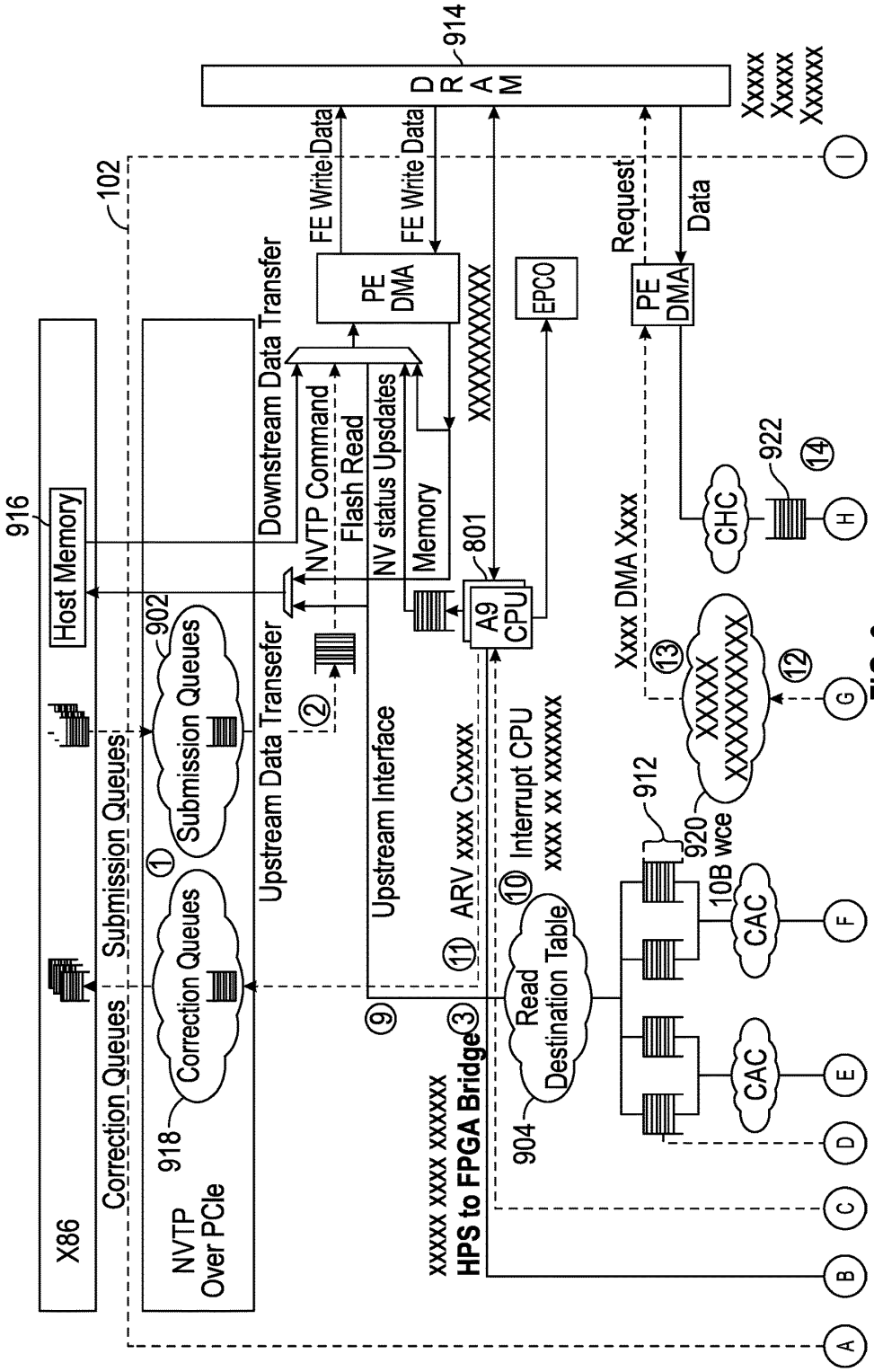
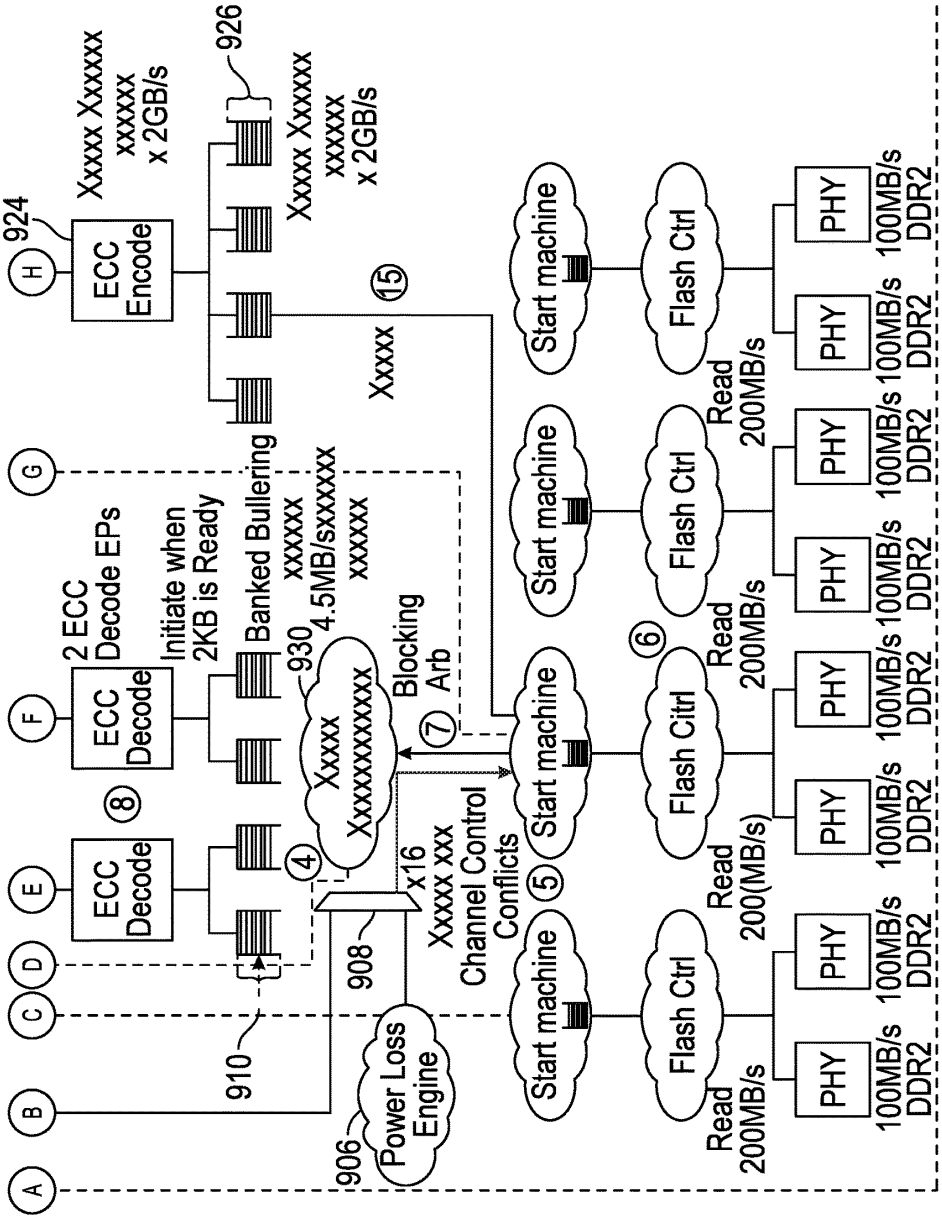

To/From CPU
102
I/O Interface: PCI-E + NVMe
Write FIFOS
805
Read FIFOS
801
CPU
214
DMA
DRAM CTRL
216
DRAM
807
Encoder
803
Decoder
Arbitrator/ Scheduler
Channel Fabric
104
PHY
PHY
PHY
104
PHY
106
108
106
108
NAND
NAND
NAND
NAND

FIG. 8


914
DRAM
Xxxxx
Xxxxx
Xxxxxx
102
FE Write Data
FE Write Data
Request
Data
PE DMA
EPCO
PE DMA
Downstream Data Transfer
NVTP Command
Flash Read
NV status Upsdates
Memory
XXXXXXXXXX
916
Host Memory
902
Submission Queues
Submission Queues
Correction Queues
Correction Queues
Upstream Data Transfer
Upstream Interface
801
A9 CPU
Xxxx DMA Xxxx
CHC
922
XXXXX XXXXXXXXX
912
920
10B wce
CAC
CAC
ARV xxxx Cxxxxx
Interrupt CPU
xxxx xx xxxxxxx
Read Destination Table
904
918
NVTP Over PCIe
X86
xxxxx xxxx xxxxxx
HPS to FPGA Bridge
FIG. 9


A
B
C
D
E
F
G
H
I
924
ECC Encode
Xxxxx Xxxxxx
xxxxxx
x 2GB/s
926
Xxxxx Xxxxxx
xxxxxx
x 2GB/s
Xxxxx
15
2 ECC Decode EPs
ECC Decode
8
ECC Decode
Initiate when 2KB is Ready
Banked Bullering
xxxxxx
4.5MB/sxxxxxx
xxxxxx
930
Xxxxx
Xxxxxxxxxxx
4
Blocking Arb
7
910
908
x16
Xxxxx xxx
Channel Control Conflicts
906
Power Loss Engine
Start machine
5
Start machine
Start machine
Flash Ctrl
Read 200(MB/s)
Flash Citrl
6
Flash Ctrl
Read 200MB/s
Read 200MB/s
Read 200MB/s
PHY
100MB/s DDR2
FIG. 9 (Continued)

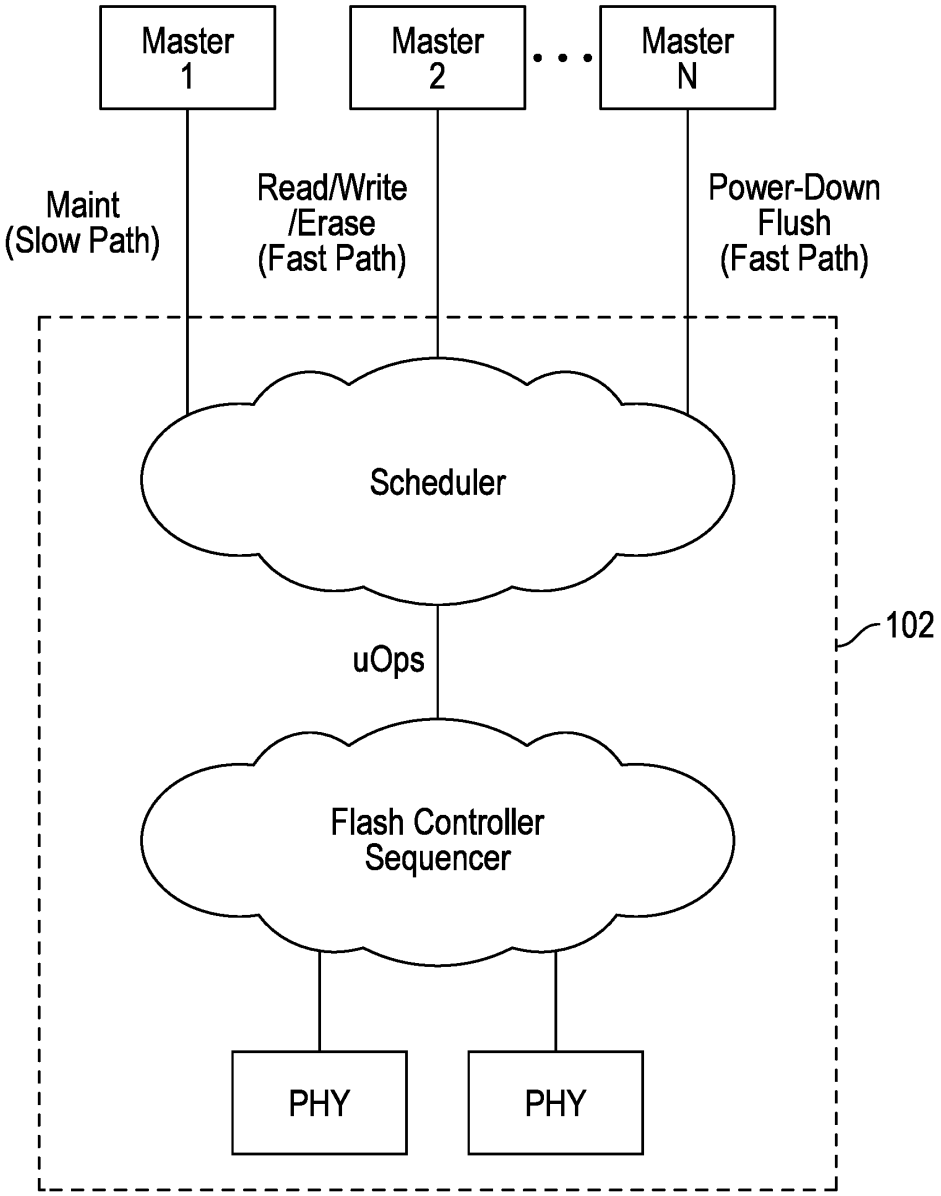

Master 1
Master 2
Master N
Maint (Slow Path)
Read/Write /Erase (Fast Path)
Power-Down Flush (Fast Path)
Scheduler
uOps
Flash Controller Sequencer
PHY
PHY
102

FIG. 10

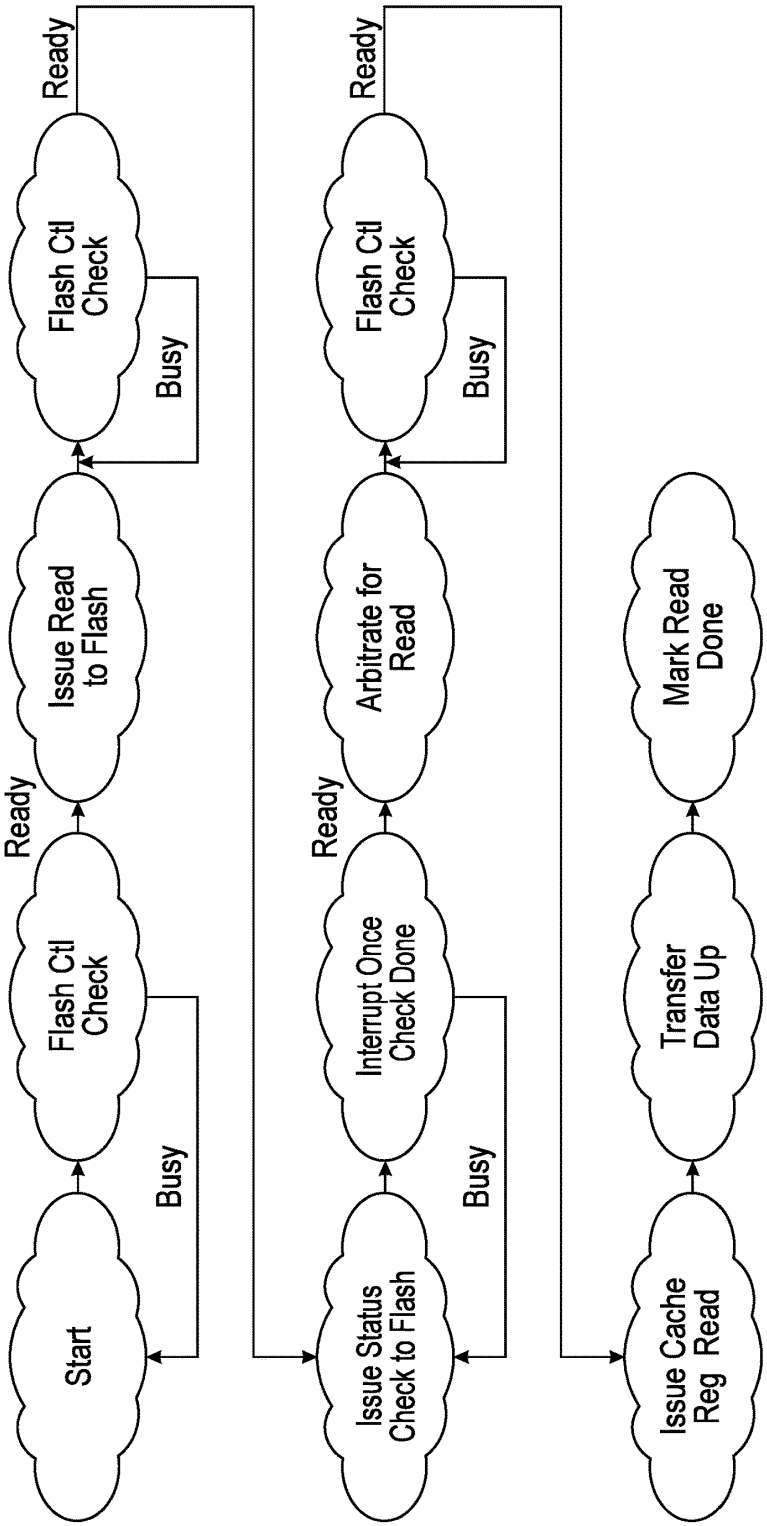

Start
Busy
Flash Ctl Check
Ready
Issue Read to Flash
Busy
Flash Ctl Check
Ready
Issue Status Check to Flash
Busy
Interrupt Once Check Done
Ready
Arbitrate for Read
Busy
Flash Ctl Check
Ready
Issue Cache Reg Read
Transfer Data Up
Mark Read Done

FIG. 11

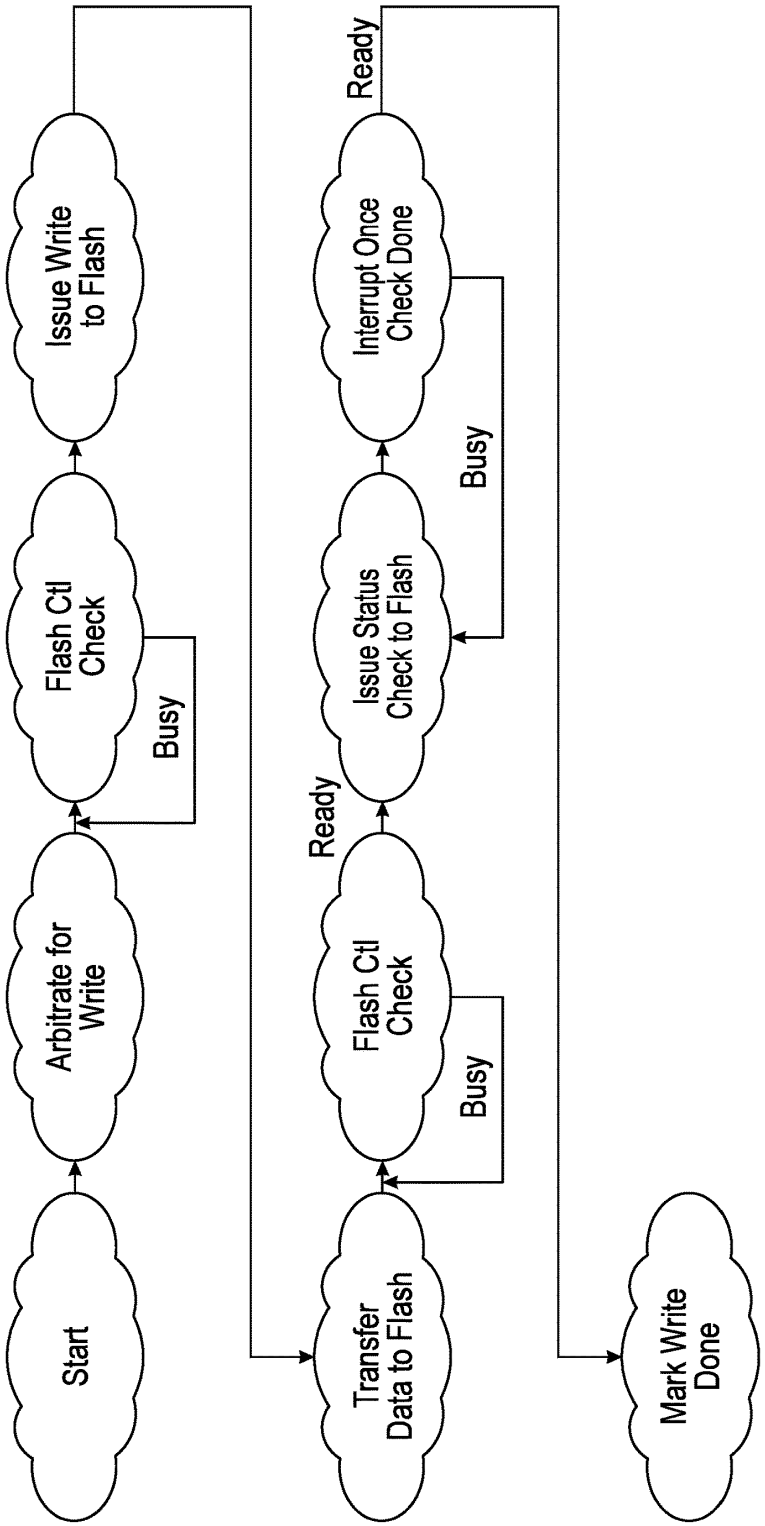

Start
Arbitrate for Write
Flash Ctl Check
Busy
Issue Write to Flash
Transfer Data to Flash
Flash Ctl Check
Busy
Ready
Issue Status Check to Flash
Busy
Interrupt Once Check Done
Ready
Mark Write Done

FIG. 12

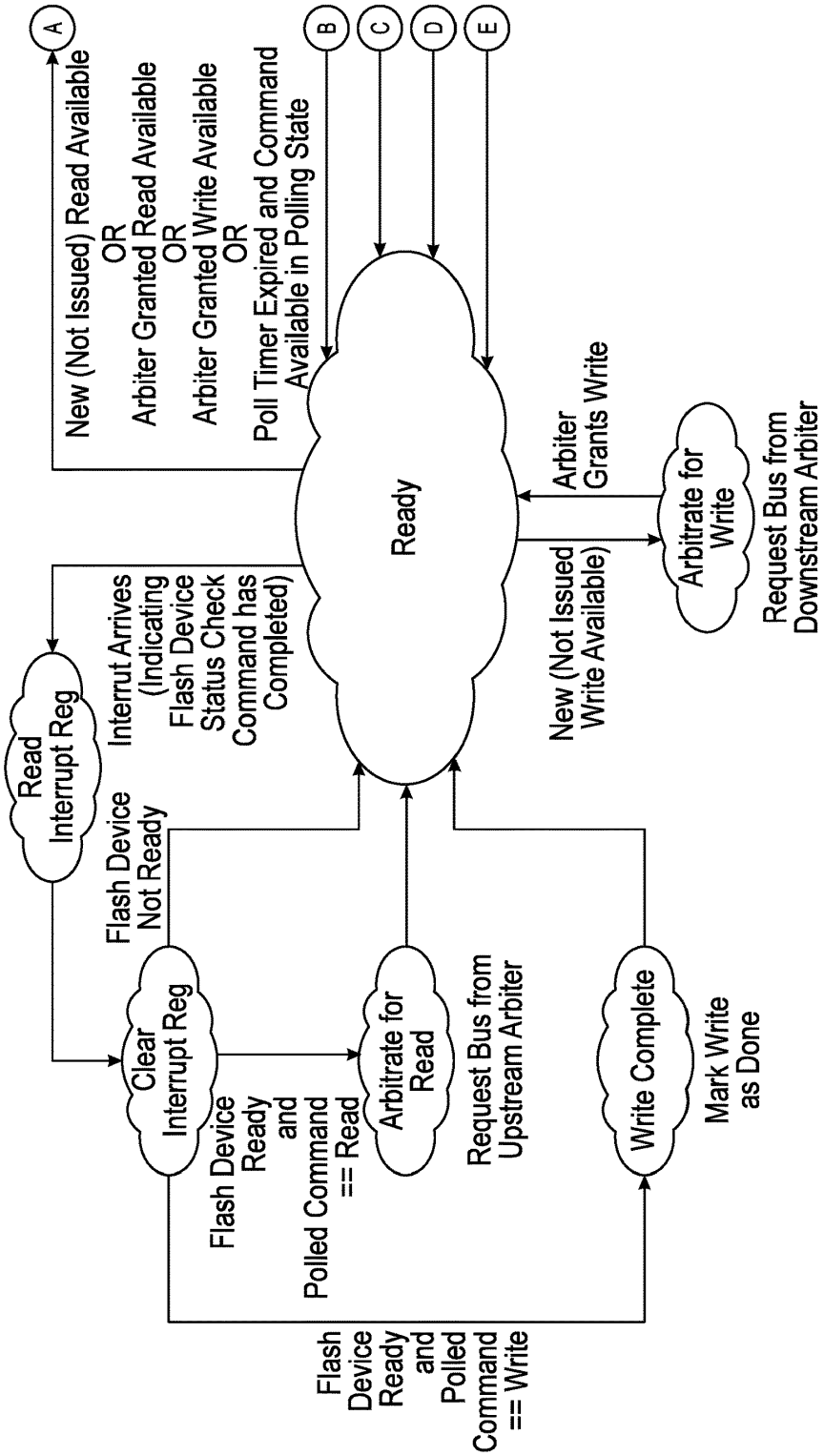

A
New (Not Issued) Read Available
OR
Arbiter Granted Read Available
OR
Arbiter Granted Write Available
OR
Poll Timer Expired and Command Available in Polling State
B
C
D
E
Ready
Arbiter Grants Write
Arbitrate for Write
Request Bus from Downstream Arbiter
New (Not Issued Write Available)
Read Interrupt Reg
Interrut Arrives (Indicating Flash Device Status Check Command has Completed)
Flash Device Not Ready
Clear Interrupt Reg
Flash Device Ready and Polled Command == Read
Arbitrate for Read
Request Bus from Upstream Arbiter
Write Complete
Mark Write as Done
Flash Device Ready and Polled Command == Write

FIG. 13A

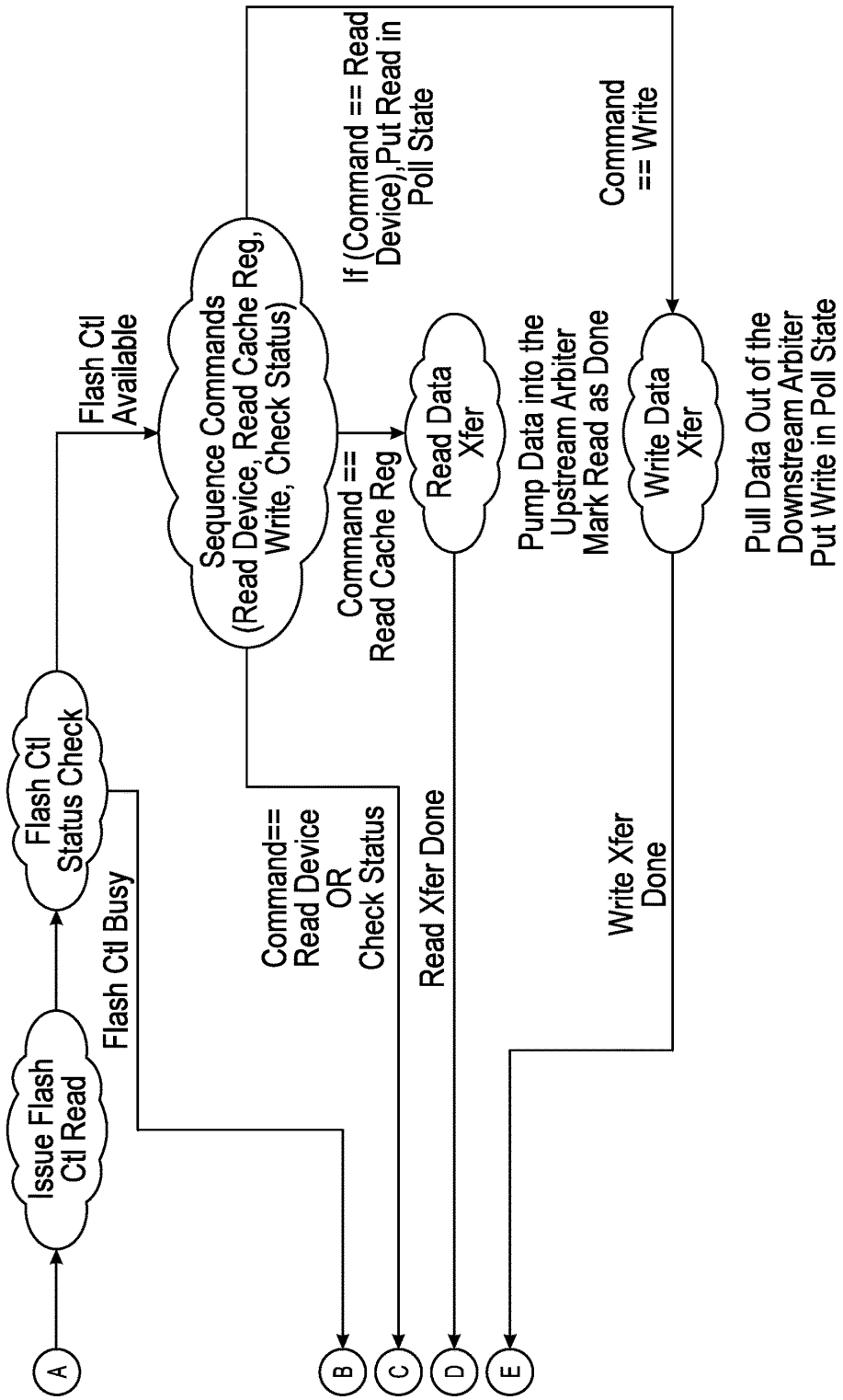


FIG. 13A (Continued)

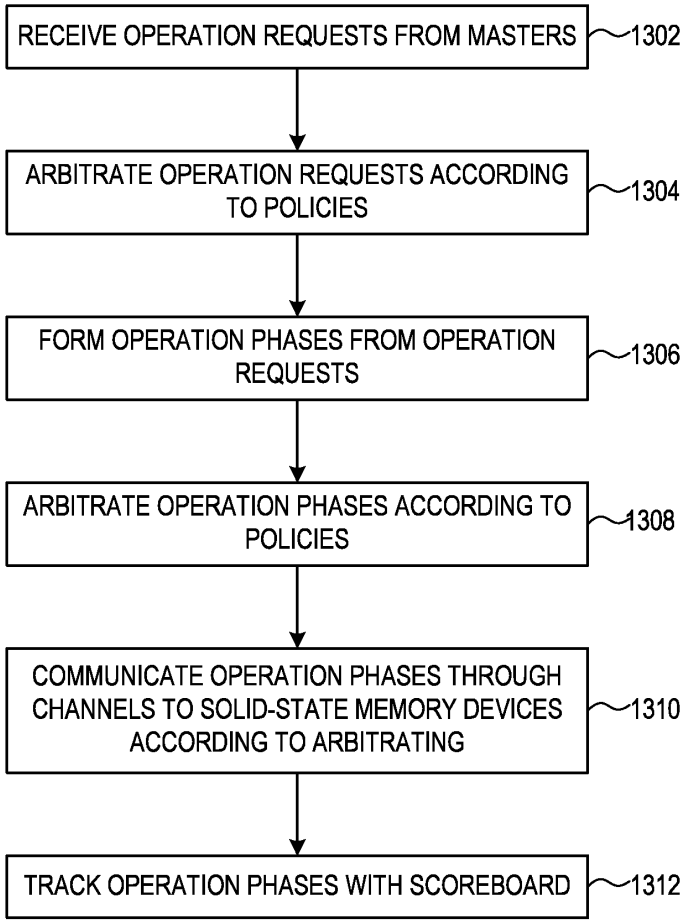

RECEIVE OPERATION REQUESTS FROM MASTERS
1302
ARBITRATE OPERATION REQUESTS ACCORDING TO POLICIES
1304
FORM OPERATION PHASES FROM OPERATION REQUESTS
1306
ARBITRATE OPERATION PHASES ACCORDING TO POLICIES
1308
COMMUNICATE OPERATION PHASES THROUGH CHANNELS TO SOLID-STATE MEMORY DEVICES ACCORDING TO ARBITRATING
1310
TRACK OPERATION PHASES WITH SCOREBOARD
1312

FIG. 13B

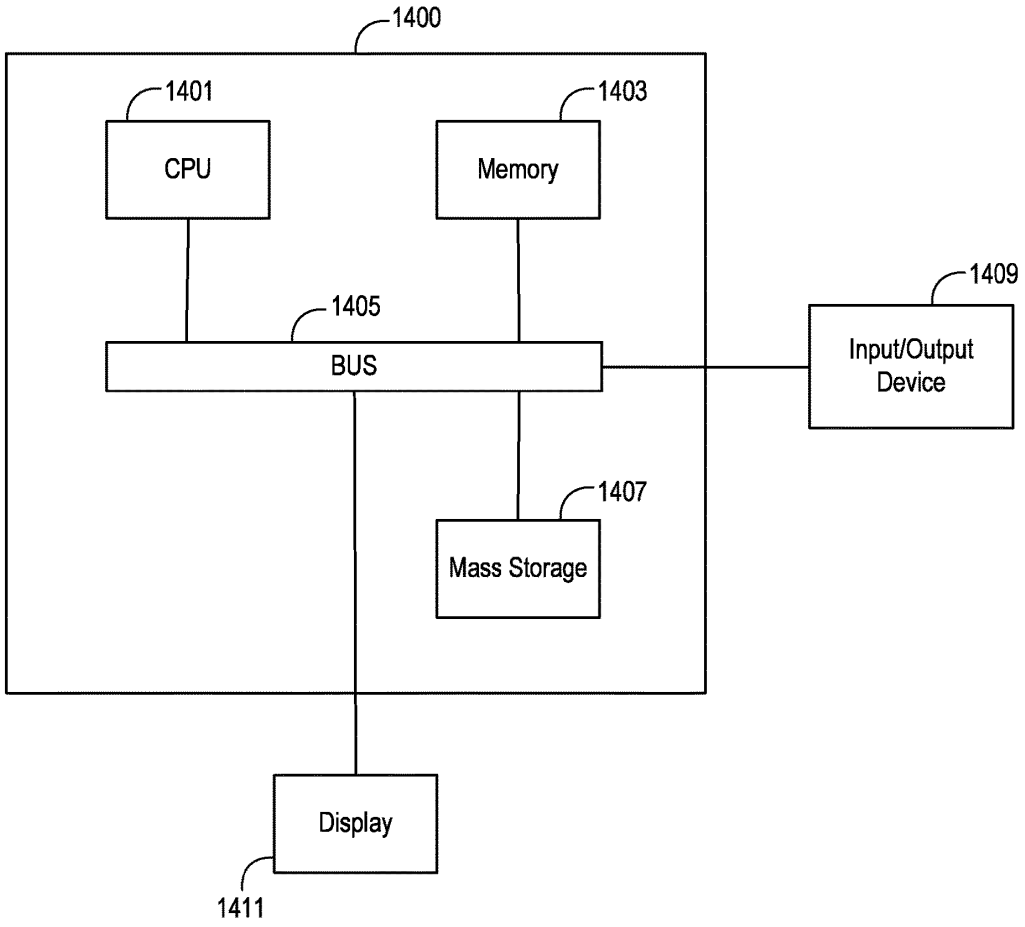

1400
1401
CPU
1403
Memory
1405
BUS
1409
Input/Output
Device
1407
Mass Storage
Display
1411

FIG. 14

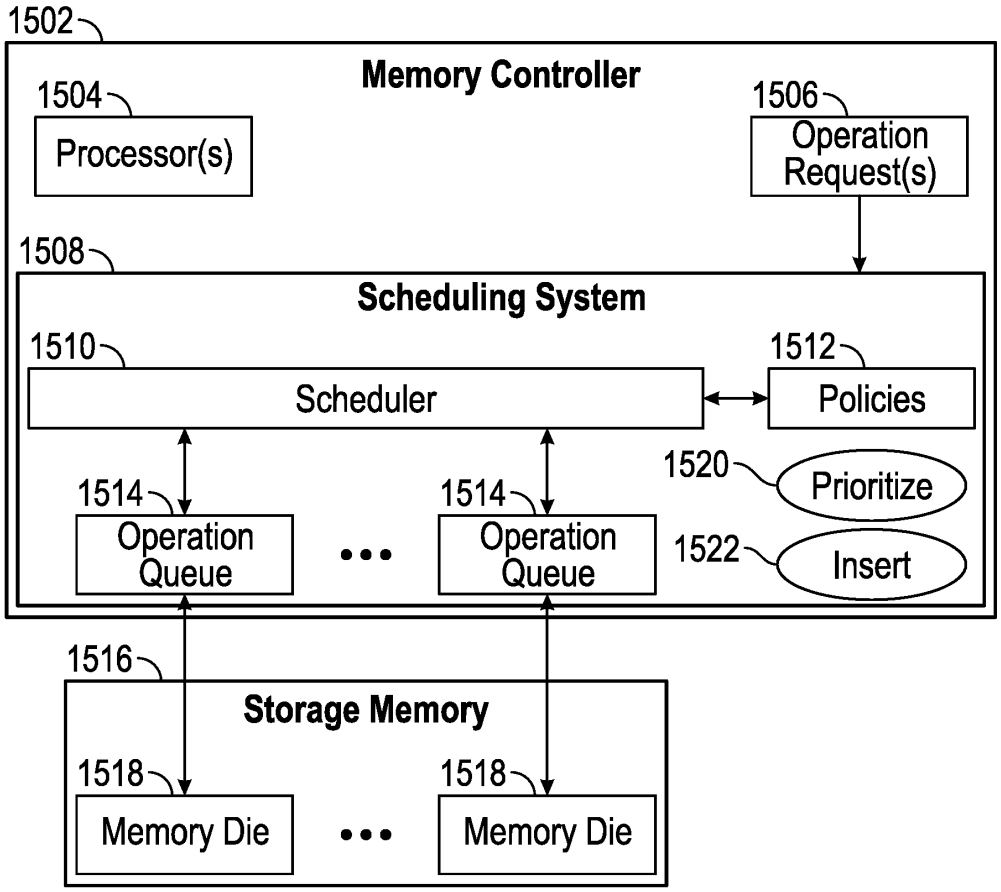

1502
Memory Controller
1504
Processor(s)
1506
Operation Request(s)
1508
Scheduling System
1510
Scheduler
1512
Policies
1520
Prioritize
1514
Operation Queue
1514
Operation Queue
1522
Insert
1516
Storage Memory
1518
Memory Die
1518
Memory Die

FIG. 15

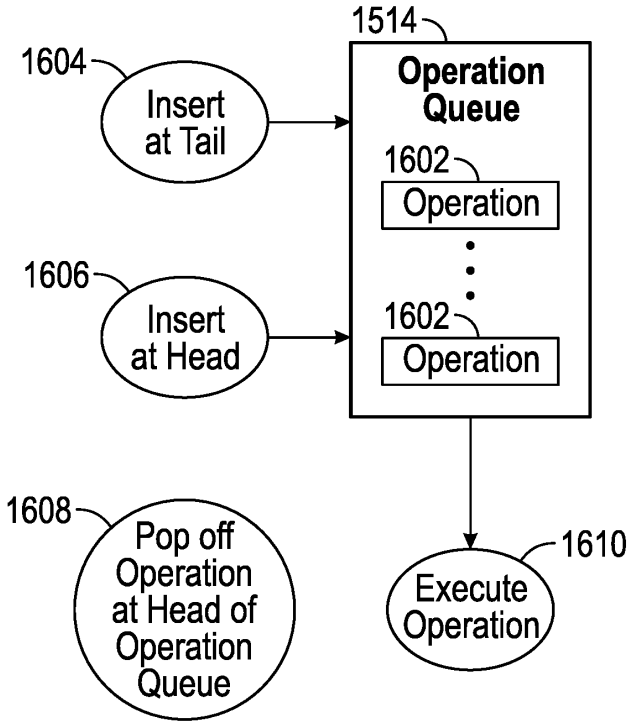

1514
Operation Queue
1604
Insert at Tail
1602
Operation
1606
Insert at Head
1602
Operation
1608
Pop off Operation at Head of Operation Queue
1610
Execute Operation

FIG. 16

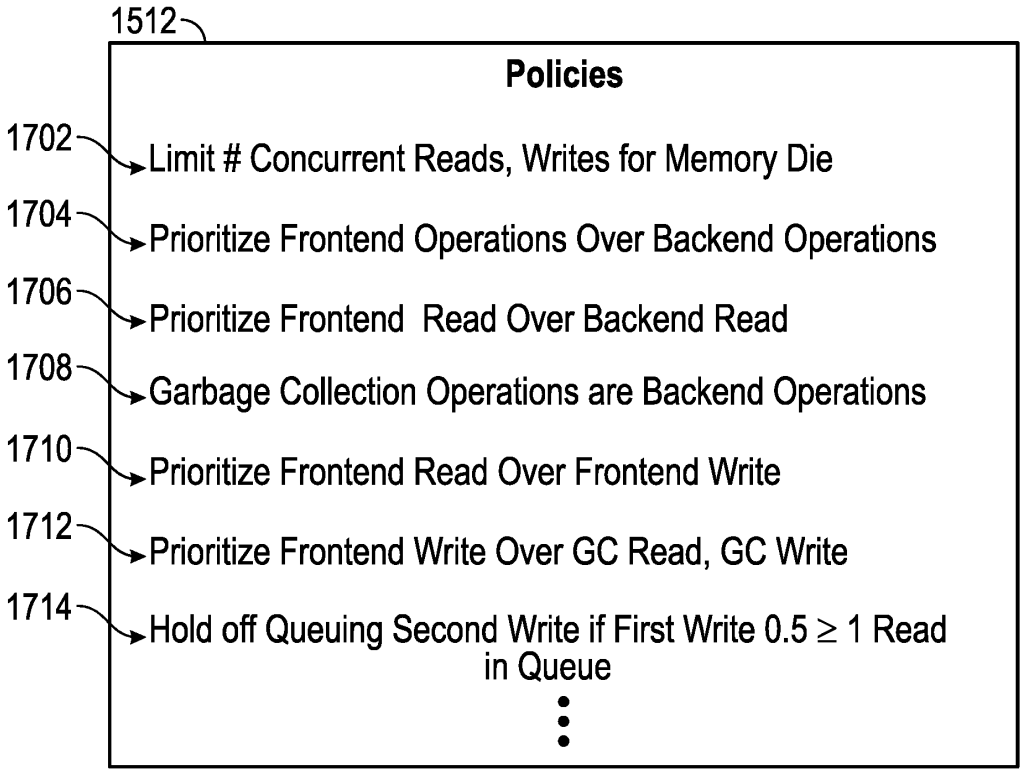

1512
Policies
1702
Limit # Concurrent Reads, Writes for Memory Die
1704
Prioritize Frontend Operations Over Backend Operations
1706
Prioritize Frontend Read Over Backend Read
1708
Garbage Collection Operations are Backend Operations
1710
Prioritize Frontend Read Over Frontend Write
1712
Prioritize Frontend Write Over GC Read, GC Write
1714
Hold off Queuing Second Write if First Write 0.5 ≥ 1 Read in Queue

FIG. 17

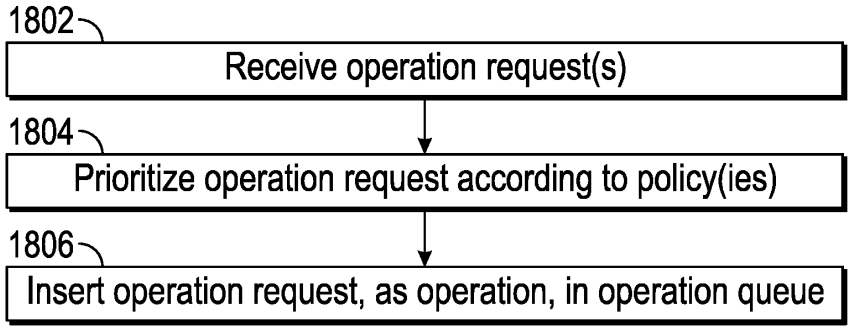

1802
Receive operation request(s)
1804
Prioritize operation request according to policy(ies)
1806
Insert operation request, as operation, in operation queue

FIG. 18

INTELLIGENT DIE AWARE STORAGE DEVICE SCHEDULER

BACKGROUND

[0001] NAND flash memory is available from different vendors, with different flash memory device interfaces and protocols. These protocols include asynchronous SDR (single data rate), synchronous DDR (double data rate), Toggle Mode (also a type of DDR or double data rate, in various release versions and from various manufacturers) and ONFI (Open NAND Flash Interface Working Group Standard, also a type of DDR or double data rate, in various release versions and from various manufacturers), and others may be developed. The proliferation of flash memory device interfaces and protocols poses a problem to designers of flash controllers for various storage devices, who generally choose one flash memory device interface and one protocol, and design the flash controller according to the selected device. However, even if a device had the ability to support different channels in a single flash controller, it would be desirable for the device to support scheduling parallel operations across the different channels/flash devices. In addition, with multiple types of memory in a system the differences in latencies among the differing types of memory can lead to contention. It is within this context that the embodiments arise.

BRIEF DESCRIPTION OF THE DRAWINGS

[0002] The described embodiments and the advantages thereof may best be understood by reference to the following description taken in conjunction with the accompanying drawings. These drawings in no way limit any changes in form and detail that may be made to the described embodiments by one skilled in the art without departing from the spirit and scope of the described embodiments.

[0003] FIG. **1**A illustrates a first example system for data storage in accordance with some implementations.

[0004] FIG. **1**B illustrates a second example system for data storage in accordance with some implementations.

[0005] FIG. **1**C illustrates a third example system for data storage in accordance with some implementations.

[0006] FIG. **1**D illustrates a fourth example system for data storage in accordance with some implementations.

[0007] FIG. **2**A is a perspective view of a storage cluster with multiple storage nodes and internal storage coupled to each storage node to provide network attached storage, in accordance with some embodiments.

[0008] FIG. **2**B is a block diagram showing an interconnect switch coupling multiple storage nodes in accordance with some embodiments.

[0009] FIG. **2**C is a multiple level block diagram, showing contents of a storage node and contents of one of the non-volatile solid state storage units in accordance with some embodiments.

[0010] FIG. **2**D shows a storage server environment, which uses embodiments of the storage nodes and storage units of some previous figures in accordance with some embodiments.

[0011] FIG. **2**E is a blade hardware block diagram, showing a control plane, compute and storage planes, and authorities interacting with underlying physical resources, in accordance with some embodiments.

[0012] FIG. **2**F depicts elasticity software layers in blades of a storage cluster, in accordance with some embodiments.

[0013] FIG. **2**G depicts authorities and storage resources in blades of a storage cluster, in accordance with some embodiments.

[0014] FIG. **3**A sets forth a diagram of a storage system that is coupled for data communications with a cloud services provider in accordance with some embodiments of the present disclosure.

[0015] FIG. **3**B sets forth a diagram of a storage system in accordance with some embodiments of the present disclosure.

[0016] FIG. **4** sets forth an example of a cloud-based storage system in accordance with some embodiments of the present disclosure.

[0017] FIG. **5** illustrates an exemplary computing device **350** that may be specifically configured to perform one or more of the processes described herein.

[0018] FIG. **6** depicts a flash controller that is configurable to couple to flash memories with differing flash memory device interfaces in accordance with some embodiments.

[0019] FIG. **7**A is a block diagram showing structural details of an embodiment of the flash controller of FIG. **6**, including a multithreaded/virtualized microcode sequence engine and multiple channels, each with phy controls, channel configuration registers and a software calibrated I/O module in accordance with some embodiments.

[0020] FIG. **7**B is a block diagram showing further structural details of an embodiment of the flash controller of FIG. **6**, including a scheduler with a scoreboard, an arbiter with configurable policies, and a sequencer with multiple channels each with a phy and flash memory.

[0021] FIG. **8** is a high level block diagram of the flash controller in accordance with some embodiments.

[0022] FIG. **9** is a microarchitecture diagram of the flash controller in accordance with some embodiments.

[0023] FIG. **10** is a high level overview of the flash controller handling operation requests from multiple masters in accordance with some embodiments.

[0024] FIG. **11** is an action diagram illustrating the read flow for the flash controller state machine in accordance with some embodiments.

[0025] FIG. **12** is an action diagram illustrating the write flow for the flash controller state machine in accordance with some embodiments.

[0026] FIG. **13**A illustrates a flash controller state machine transition diagram in accordance with some embodiments.

[0027] FIG. **13**B is a flow diagram of a method for scheduling in a memory controller.

[0028] FIG. **14** is an illustration showing an exemplary computing device which may implement the embodiments described herein.

[0029] FIG. **15** illustrates a memory controller with a scheduling system that prioritizes operations in accordance with present embodiments.

[0030] FIG. **16** illustrates an operation queue, with operations, that is suitable for embodiments of the scheduling system of FIG. **15**.

[0031] FIG. **17** illustrates example policies that are suitable for the scheduling system of FIG. **15**.

[0032] FIG. **18** is a flow diagram of a method that is practiced on and by embodiments of the scheduling system and memory controller of FIG. **15**.

DETAILED DESCRIPTION

[0033] Various storage systems described herein, and further storage systems, can be optimized for distribution of selected data, according to various criteria, in flash or other solid-state memory. The embodiments for the distributed flash wear leveling system are optimized for faster read access to the flash or other solid-state memory. Flash memory that is worn, i.e., that has a large number of program/erase cycles, often or usually has a greater error rate during read accesses, and this adds to read latency for data bits as a result of the processing time overhead to perform error correction. Various embodiments of the storage system track program/erase cycles, or track read errors or error rates, for example on a page, block, die, package, board, storage unit or storage node basis, are aware of faster and slower types or designs of flash memory or portions of flash memory, or otherwise determine relative access speeds for flash memory. The storage system then places data selectively in faster access or slower access locations in or portions of flash memory (or other solid-state memory). One embodiments of the storage system writes data bits to faster access portions of flash memory and parity bits to slower access portions of flash memory. Another embodiment takes advantage of faster and slower access pages of triple level cell flash memory. Principles of operation, variations, and implementation details for distributed flash wear leveling are further discussed below, with reference to FIGS. **6-13**, following description of embodiments of a storage cluster with storage nodes, suitable for distributed flash wear leveling. FIGS. **1-14** are followed by FIGS. **15-18**, which set forth embodiments of a memory controller with a scheduling system that prioritizes operations in operation queues for memory dies of storage memory.

[0034] Example methods, apparatus, and products for a memory controller scheduling system n accordance with embodiments of the present disclosure are described with reference to the accompanying drawings, beginning with FIG. **1**A. FIG. **1**A illustrates an example system for data storage, in accordance with some implementations. System **100** (also referred to as "storage system" herein) includes numerous elements for purposes of illustration rather than limitation. It may be noted that system **100** may include the same, more, or fewer elements configured in the same or different manner in other implementations.

[0035] System **100** includes a number of computing devices **164**A-B. Computing devices (also referred to as "client devices" herein) may be embodied, for example, a server in a data center, a workstation, a personal computer, a notebook, or the like. Computing devices **164**A-B may be coupled for data communications to one or more storage arrays **102**A-B through a storage area network ('SAN') **158** or a local area network ('LAN') **160**.

[0036] The SAN **158** may be implemented with a variety of data communications fabrics, devices, and protocols. For example, the fabrics for SAN **158** may include Fibre Channel, Ethernet, Infiniband, Serial Attached Small Computer System Interface ('SAS'), or the like. Data communications protocols for use with SAN **158** may include Advanced Technology Attachment ('ATA'), Fibre Channel Protocol, Small Computer System Interface ('SCSI'), Internet Small Computer System Interface ('iSCSI'), HyperSCSI, Non-Volatile Memory Express ('NVMe') over Fabrics, or the like. It may be noted that SAN **158** is provided for illustration, rather than limitation. Other data communication couplings may be implemented between computing devices **164**A-B and storage arrays **102**A-B.

[0037] The LAN **160** may also be implemented with a variety of fabrics, devices, and protocols. For example, the fabrics for LAN **160** may include Ethernet (802.3), wireless (802.11), or the like. Data communication protocols for use in LAN **160** may include Transmission Control Protocol ('TCP'), User Datagram Protocol ('UDP'), Internet Protocol ('IP'), HyperText Transfer Protocol ('HTTP'), Wireless Access Protocol ('WAP'), Handheld Device Transport Protocol ('HDTP'), Session Initiation Protocol ('SIP'), Real Time Protocol ('RTP'), or the like.

[0038] Storage arrays **102**A-B may provide persistent data storage for the computing devices **164**A-B. Storage array **102**A may be contained in a chassis (not shown), and storage array **102**B may be contained in another chassis (not shown), in implementations. Storage array **102**A and **102**B may include one or more storage array controllers **110**A-D (also referred to as "controller" herein). A storage array controller **110**A-D may be embodied as a module of automated computing machinery comprising computer hardware, computer software, or a combination of computer hardware and software. In some implementations, the storage array controllers **110**A-D may be configured to carry out various storage tasks. Storage tasks may include writing data received from the computing devices **164**A-B to storage array **102**A-B, erasing data from storage array **102**A-B, retrieving data from storage array **102**A-B and providing data to computing devices **164**A-B, monitoring and reporting of disk utilization and performance, performing redundancy operations, such as Redundant Array of Independent Drives (RAID') or RAID-like data redundancy operations, compressing data, encrypting data, and so forth.

[0039] Storage array controller **110**A-D may be implemented in a variety of ways, including as a Field Programmable Gate Array ('FPGA'), a Programmable Logic Chip ('PLC'), an Application Specific Integrated Circuit ('ASIC'), System-on-Chip ('SOC'), or any computing device that includes discrete components such as a processing device, central processing unit, computer memory, or various adapters. Storage array controller **110**A-D may include, for example, a data communications adapter configured to support communications via the SAN **158** or LAN **160**. In some implementations, storage array controller **110**A-D may be independently coupled to the LAN **160**. In implementations, storage array controller **110**A-D may include an I/O controller or the like that couples the storage array controller **110**A-D for data communications, through a midplane (not shown), to a persistent storage resource **170**A-B (also referred to as a "storage resource" herein). The persistent storage resource **170**A-B main include any number of storage drives **171**A-F (also referred to as "storage devices" herein) and any number of non-volatile Random Access Memory ('NVRAM') devices (not shown).

[0040] In some implementations, the NVRAM devices of a persistent storage resource **170**A-B may be configured to receive, from the storage array controller **110**A-D, data to be stored in the storage drives **171**A-F. In some examples, the data may originate from computing devices **164**A-B. In some examples, writing data to the NVRAM device may be carried out more quickly than directly writing data to the storage drive **171**A-F. In implementations, the storage array controller **110**A-D may be configured to utilize the NVRAM devices as a quickly accessible buffer for data destined to be

written to the storage drives **171**A-F. Latency for write requests using NVRAM devices as a buffer may be improved relative to a system in which a storage array controller **110**A-D writes data directly to the storage drives **171**A-F. In some implementations, the NVRAM devices may be implemented with computer memory in the form of high bandwidth, low latency RAM. The NVRAM device is referred to as "non-volatile" because the NVRAM device may receive or include a unique power source that maintains the state of the RAM after main power loss to the NVRAM device. Such a power source may be a battery, one or more capacitors, or the like. In response to a power loss, the NVRAM device may be configured to write the contents of the RAM to a persistent storage, such as the storage drives **171**A-F.

[0041] In implementations, storage drive **171**A-F may refer to any device configured to record data persistently, where "persistently" or "persistent" refers as to a device's ability to maintain recorded data after loss of power. In some implementations, storage drive **171**A-F may correspond to non-disk storage media. For example, the storage drive **171**A-F may be one or more solid-state drives ('SSDs'), flash memory based storage, any type of solid-state non-volatile memory, or any other type of non-mechanical storage device. In other implementations, storage drive **171**A-F may include mechanical or spinning hard disk, such as hard-disk drives ('HDD').

[0042] In some implementations, the storage array controllers **110**A-D may be configured for offloading device management responsibilities from storage drive **171**A-F in storage array **102**A-B. For example, storage array controllers **110**A-D may manage control information that may describe the state of one or more memory blocks in the storage drives **171**A-F. The control information may indicate, for example, that a particular memory block has failed and should no longer be written to, that a particular memory block contains boot code for a storage array controller **110**A-D, the number of program-erase ('P/E') cycles that have been performed on a particular memory block, the age of data stored in a particular memory block, the type of data that is stored in a particular memory block, and so forth. In some implementations, the control information may be stored with an associated memory block as metadata. In other implementations, the control information for the storage drives **171**A-F may be stored in one or more particular memory blocks of the storage drives **171**A-F that are selected by the storage array controller **110**A-D. The selected memory blocks may be tagged with an identifier indicating that the selected memory block contains control information. The identifier may be utilized by the storage array controllers **110**A-D in conjunction with storage drives **171**A-F to quickly identify the memory blocks that contain control information. For example, the storage controllers **110**A-D may issue a command to locate memory blocks that contain control information. It may be noted that control information may be so large that parts of the control information may be stored in multiple locations, that the control information may be stored in multiple locations for purposes of redundancy, for example, or that the control information may otherwise be distributed across multiple memory blocks in the storage drive **171**A-F.

[0043] In implementations, storage array controllers **110**A-D may offload device management responsibilities from storage drives **171**A-F of storage array **102**A-B by retrieving, from the storage drives **171**A-F, control information describing the state of one or more memory blocks in the storage drives **171**A-F. Retrieving the control information from the storage drives **171**A-F may be carried out, for example, by the storage array controller **110**A-D querying the storage drives **171**A-F for the location of control information for a particular storage drive **171**A-F. The storage drives **171**A-F may be configured to execute instructions that enable the storage drive **171**A-F to identify the location of the control information. The instructions may be executed by a controller (not shown) associated with or otherwise located on the storage drive **171**A-F and may cause the storage drive **171**A-F to scan a portion of each memory block to identify the memory blocks that store control information for the storage drives **171**A-F. The storage drives **171**A-F may respond by sending a response message to the storage array controller **110**A-D that includes the location of control information for the storage drive **171**A-F. Responsive to receiving the response message, storage array controllers **110**A-D may issue a request to read data stored at the address associated with the location of control information for the storage drives **171**A-F.

[0044] In other implementations, the storage array controllers **110**A-D may further offload device management responsibilities from storage drives **171**A-F by performing, in response to receiving the control information, a storage drive management operation. A storage drive management operation may include, for example, an operation that is typically performed by the storage drive **171**A-F (e.g., the controller (not shown) associated with a particular storage drive **171**A-F). A storage drive management operation may include, for example, ensuring that data is not written to failed memory blocks within the storage drive **171**A-F, ensuring that data is written to memory blocks within the storage drive **171**A-F in such a way that adequate wear leveling is achieved, and so forth.

[0045] In implementations, storage array **102**A-B may implement two or more storage array controllers **110**A-D. For example, storage array **102**A may include storage array controllers **110**A and storage array controllers **110**B. At a given instance, a single storage array controller **110**A-D (e.g., storage array controller **110**A) of a storage system **100** may be designated with primary status (also referred to as "primary controller" herein), and other storage array controllers **110**A-D (e.g., storage array controller **110**A) may be designated with secondary status (also referred to as "secondary controller" herein). The primary controller may have particular rights, such as permission to alter data in persistent storage resource **170**A-B (e.g., writing data to persistent storage resource **170**A-B). At least some of the rights of the primary controller may supersede the rights of the secondary controller. For instance, the secondary controller may not have permission to alter data in persistent storage resource **170**A-B when the primary controller has the right. The status of storage array controllers **110**A-D may change. For example, storage array controller **110**A may be designated with secondary status, and storage array controller **110**B may be designated with primary status.

[0046] In some implementations, a primary controller, such as storage array controller **110**A, may serve as the primary controller for one or more storage arrays **102**A-B, and a second controller, such as storage array controller **110**B, may serve as the secondary controller for the one or more storage arrays **102**A-B. For example, storage array

controller **110**A may be the primary controller for storage array **102**A and storage array **102**B, and storage array controller **110**B may be the secondary controller for storage array **102**A and **102**B. In some implementations, storage array controllers **110**C and **110**D (also referred to as "storage processing modules") may neither have primary or secondary status. Storage array controllers **110**C and **110**D, implemented as storage processing modules, may act as a communication interface between the primary and secondary controllers (e.g., storage array controllers **110**A and **110**B, respectively) and storage array **102**B. For example, storage array controller **110**A of storage array **102**A may send a write request, via SAN **158**, to storage array **102**B. The write request may be received by both storage array controllers **110**C and **110**D of storage array **102**B. Storage array controllers **110**C and **110**D facilitate the communication, e.g., send the write request to the appropriate storage drive **171**A-F. It may be noted that in some implementations storage processing modules may be used to increase the number of storage drives controlled by the primary and secondary controllers.

[0047] In implementations, storage array controllers **110**A-D are communicatively coupled, via a midplane (not shown), to one or more storage drives **171**A-F and to one or more NVRAM devices (not shown) that are included as part of a storage array **102**A-B. The storage array controllers **110**A-D may be coupled to the midplane via one or more data communication links and the midplane may be coupled to the storage drives **171**A-F and the NVRAM devices via one or more data communications links. The data communications links described herein are collectively illustrated by data communications links **108**A-D and may include a Peripheral Component Interconnect Express ('PCIe') bus, for example.

[0048] FIG. **1**B illustrates an example system for data storage, in accordance with some implementations. Storage array controller **101** illustrated in FIG. **1**B may similar to the storage array controllers **110**A-D described with respect to FIG. **1**A. In one example, storage array controller **101** may be similar to storage array controller **110**A or storage array controller **110**B. Storage array controller **101** includes numerous elements for purposes of illustration rather than limitation. It may be noted that storage array controller **101** may include the same, more, or fewer elements configured in the same or different manner in other implementations. It may be noted that elements of FIG. **1**A may be included below to help illustrate features of storage array controller **101**.

[0049] Storage array controller **101** may include one or more processing devices **104** and random access memory ('RAM') **111**. Processing device **104** (or controller **101**) represents one or more general-purpose processing devices such as a microprocessor, central processing unit, or the like. More particularly, the processing device **104** (or controller **101**) may be a complex instruction set computing ('CISC') microprocessor, reduced instruction set computing (RISC') microprocessor, very long instruction word ('VLIW') microprocessor, or a processor implementing other instruction sets or processors implementing a combination of instruction sets. The processing device **104** (or controller **101**) may also be one or more special-purpose processing devices such as an ASIC, an FPGA, a digital signal processor ('DSP'), network processor, or the like.

[0050] The processing device **104** may be connected to the RAM **111** via a data communications link **106**, which may be embodied as a high speed memory bus such as a Double-Data Rate 4 ('DDR4') bus. Stored in RAM **111** is an operating system **112**. In some implementations, instructions **113** are stored in RAM **111**. Instructions **113** may include computer program instructions for performing operations in in a direct-mapped flash storage system. In one embodiment, a direct-mapped flash storage system is one that that addresses data blocks within flash drives directly and without an address translation performed by the storage controllers of the flash drives.

[0051] In implementations, storage array controller **101** includes one or more host bus adapters **103**A-C that are coupled to the processing device **104** via a data communications link **105**A-C. In implementations, host bus adapters **103**A-C may be computer hardware that connects a host system (e.g., the storage array controller) to other network and storage arrays. In some examples, host bus adapters **103**A-C may be a Fibre Channel adapter that enables the storage array controller **101** to connect to a SAN, an Ethernet adapter that enables the storage array controller **101** to connect to a LAN, or the like. Host bus adapters **103**A-C may be coupled to the processing device **104** via a data communications link **105**A-C such as, for example, a PCIe bus.

[0052] In implementations, storage array controller **101** may include a host bus adapter **114** that is coupled to an expander **115**. The expander **115** may be used to attach a host system to a larger number of storage drives. The expander **115** may, for example, be a SAS expander utilized to enable the host bus adapter **114** to attach to storage drives in an implementation where the host bus adapter **114** is embodied as a SAS controller.

[0053] In implementations, storage array controller **101** may include a switch **116** coupled to the processing device **104** via a data communications link **109**. The switch **116** may be a computer hardware device that can create multiple endpoints out of a single endpoint, thereby enabling multiple devices to share a single endpoint. The switch **116** may, for example, be a PCIe switch that is coupled to a PCIe bus (e.g., data communications link **109**) and presents multiple PCIe connection points to the midplane.

[0054] In implementations, storage array controller **101** includes a data communications link **107** for coupling the storage array controller **101** to other storage array controllers. In some examples, data communications link **107** may be a QuickPath Interconnect (QPI) interconnect.

[0055] A traditional storage system that uses traditional flash drives may implement a process across the flash drives that are part of the traditional storage system. For example, a higher level process of the storage system may initiate and control a process across the flash drives. However, a flash drive of the traditional storage system may include its own storage controller that also performs the process. Thus, for the traditional storage system, a higher level process (e.g., initiated by the storage system) and a lower level process (e.g., initiated by a storage controller of the storage system) may both be performed.

[0056] To resolve various deficiencies of a traditional storage system, operations may be performed by higher level processes and not by the lower level processes. For example, the flash storage system may include flash drives that do not include storage controllers that provide the

process. Thus, the operating system of the flash storage system itself may initiate and control the process. This may be accomplished by a direct-mapped flash storage system that addresses data blocks within the flash drives directly and without an address translation performed by the storage controllers of the flash drives.

[0057] In implementations, storage drive **171**A-F may be one or more zoned storage devices. In some implementations, the one or more zoned storage devices may be a shingled HDD. In implementations, the one or more storage devices may be a flash-based SSD. In a zoned storage device, a zoned namespace on the zoned storage device can be addressed by groups of blocks that are grouped and aligned by a natural size, forming a number of addressable zones. In implementations utilizing an SSD, the natural size may be based on the erase block size of the SSD.

[0058] The mapping from a zone to an erase block (or to a shingled track in an HDD) may be arbitrary, dynamic, and hidden from view. The process of opening a zone may be an operation that allows a new zone to be dynamically mapped to underlying storage of the zoned storage device, and then allows data to be written through appending writes into the zone until the zone reaches capacity. The zone can be finished at any point, after which further data may not be written into the zone. When the data stored at the zone is no longer needed, the zone can be reset which effectively deletes the zone's content from the zoned storage device, making the physical storage held by that zone available for the subsequent storage of data. Once a zone has been written and finished, the zoned storage device ensures that the data stored at the zone is not lost until the zone is reset. In the time between writing the data to the zone and the resetting of the zone, the zone may be moved around between shingle tracks or erase blocks as part of maintenance operations within the zoned storage device, such as by copying data to keep the data refreshed or to handle memory cell aging in an SSD.

[0059] In implementations utilizing an HDD, the resetting of the zone may allow the shingle tracks to be allocated to a new, opened zone that may be opened at some point in the future. In implementations utilizing an SSD, the resetting of the zone may cause the associated physical erase block(s) of the zone to be erased and subsequently reused for the storage of data. In some implementations, the zoned storage device may have a limit on the number of open zones at a point in time to reduce the amount of overhead dedicated to keeping zones open.

[0060] The operating system of the flash storage system may identify and maintain a list of allocation units across multiple flash drives of the flash storage system. The allocation units may be entire erase blocks or multiple erase blocks. The operating system may maintain a map or address range that directly maps addresses to erase blocks of the flash drives of the flash storage system.

[0061] Direct mapping to the erase blocks of the flash drives may be used to rewrite data and erase data. For example, the operations may be performed on one or more allocation units that include a first data and a second data where the first data is to be retained and the second data is no longer being used by the flash storage system. The operating system may initiate the process to write the first data to new locations within other allocation units and erasing the second data and marking the allocation units as being available for use for subsequent data. Thus, the process may only be performed by the higher level operating system of the flash storage system without an additional lower level process being performed by controllers of the flash drives.

[0062] Advantages of the process being performed only by the operating system of the flash storage system include increased reliability of the flash drives of the flash storage system as unnecessary or redundant write operations are not being performed during the process. One possible point of novelty here is the concept of initiating and controlling the process at the operating system of the flash storage system. In addition, the process can be controlled by the operating system across multiple flash drives. This is contrast to the process being performed by a storage controller of a flash drive.

[0063] A storage system can consist of two storage array controllers that share a set of drives for failover purposes, or it could consist of a single storage array controller that provides a storage service that utilizes multiple drives, or it could consist of a distributed network of storage array controllers each with some number of drives or some amount of Flash storage where the storage array controllers in the network collaborate to provide a complete storage service and collaborate on various aspects of a storage service including storage allocation and garbage collection.

[0064] FIG. **1**C illustrates a third example system **117** for data storage in accordance with some implementations. System **117** (also referred to as "storage system" herein) includes numerous elements for purposes of illustration rather than limitation. It may be noted that system **117** may include the same, more, or fewer elements configured in the same or different manner in other implementations.

[0065] In one embodiment, system **117** includes a dual Peripheral Component Interconnect ('PCI') flash storage device **118** with separately addressable fast write storage. System **117** may include a storage controller **119**. In one embodiment, storage controller **119**A-D may be a CPU, ASIC, FPGA, or any other circuitry that may implement control structures necessary according to the present disclosure. In one embodiment, system **117** includes flash memory devices (e.g., including flash memory devices **120***a-n*), operatively coupled to various channels of the storage device controller **119**. Flash memory devices **120***a-n*, may be presented to the controller **119**A-D as an addressable collection of Flash pages, erase blocks, and/or control elements sufficient to allow the storage device controller **119**A-D to program and retrieve various aspects of the Flash. In one embodiment, storage device controller **119**A-D may perform operations on flash memory devices **120***a-n* including storing and retrieving data content of pages, arranging and erasing any blocks, tracking statistics related to the use and reuse of Flash memory pages, erase blocks, and cells, tracking and predicting error codes and faults within the Flash memory, controlling voltage levels associated with programming and retrieving contents of Flash cells, etc.

[0066] In one embodiment, system **117** may include RAM **121** to store separately addressable fast-write data. In one embodiment, RAM **121** may be one or more separate discrete devices. In another embodiment, RAM **121** may be integrated into storage device controller **119**A-D or multiple storage device controllers. The RAM **121** may be utilized for other purposes as well, such as temporary program memory for a processing device (e.g., a CPU) in the storage device controller **119**.

[0067] In one embodiment, system **117** may include a stored energy device **122**, such as a rechargeable battery or a capacitor. Stored energy device **122** may store energy sufficient to power the storage device controller **119**, some amount of the RAM (e.g., RAM **121**), and some amount of Flash memory (e.g., Flash memory **120***a*-**120***n*) for sufficient time to write the contents of RAM to Flash memory. In one embodiment, storage device controller **119**A-D may write the contents of RAM to Flash Memory if the storage device controller detects loss of external power.

[0068] In one embodiment, system **117** includes two data communications links **123***a*, **123***b*. In one embodiment, data communications links **123***a*, **123***b* may be PCI interfaces. In another embodiment, data communications links **123***a*, **123***b* may be based on other communications standards (e.g., HyperTransport, InfiniBand, etc.). Data communications links **123***a*, **123***b* may be based on non-volatile memory express ('NVMe') or NVMe over fabrics ('NVMf') specifications that allow external connection to the storage device controller **119**A-D from other components in the storage system **117**. It should be noted that data communications links may be interchangeably referred to herein as PCI buses for convenience.

[0069] System **117** may also include an external power source (not shown), which may be provided over one or both data communications links **123***a*, **123***b*, or which may be provided separately. An alternative embodiment includes a separate Flash memory (not shown) dedicated for use in storing the content of RAM **121**. The storage device controller **119**A-D may present a logical device over a PCI bus which may include an addressable fast-write logical device, or a distinct part of the logical address space of the storage device **118**, which may be presented as PCI memory or as persistent storage. In one embodiment, operations to store into the device are directed into the RAM **121**. On power failure, the storage device controller **119**A-D may write stored content associated with the addressable fast-write logical storage to Flash memory (e.g., Flash memory **120***a*-*n*) for long-term persistent storage.

[0070] In one embodiment, the logical device may include some presentation of some or all of the content of the Flash memory devices **120***a*-*n*, where that presentation allows a storage system including a storage device **118** (e.g., storage system **117**) to directly address Flash memory pages and directly reprogram erase blocks from storage system components that are external to the storage device through the PCI bus. The presentation may also allow one or more of the external components to control and retrieve other aspects of the Flash memory including some or all of: tracking statistics related to use and reuse of Flash memory pages, erase blocks, and cells across all the Flash memory devices; tracking and predicting error codes and faults within and across the Flash memory devices; controlling voltage levels associated with programming and retrieving contents of Flash cells; etc.

[0071] In one embodiment, the stored energy device **122** may be sufficient to ensure completion of in-progress operations to the Flash memory devices **120***a*-**120***n* stored energy device **122** may power storage device controller **119**A-D and associated Flash memory devices (e.g., **120***a*-*n*) for those operations, as well as for the storing of fast-write RAM to Flash memory. Stored energy device **122** may be used to store accumulated statistics and other parameters kept and tracked by the Flash memory devices **120***a*-*n* and/or the storage device controller **119**. Separate capacitors or stored energy devices (such as smaller capacitors near or embedded within the Flash memory devices themselves) may be used for some or all of the operations described herein.

[0072] Various schemes may be used to track and optimize the life span of the stored energy component, such as adjusting voltage levels over time, partially discharging the storage energy device **122** to measure corresponding discharge characteristics, etc. If the available energy decreases over time, the effective available capacity of the addressable fast-write storage may be decreased to ensure that it can be written safely based on the currently available stored energy.

[0073] FIG. **1**D illustrates a third example system **124** for data storage in accordance with some implementations. In one embodiment, system **124** includes storage controllers **125***a*, **125***b*. In one embodiment, storage controllers **125***a*, **125***b* are operatively coupled to Dual PCI storage devices **119***a*, **119***b* and **119***c*, **119***d*, respectively. Storage controllers **125***a*, **125***b* may be operatively coupled (e.g., via a storage network **130**) to some number of host computers **127***a*-*n*.

[0074] In one embodiment, two storage controllers (e.g., **125***a* and **125***b*) provide storage services, such as a SCS) block storage array, a file server, an object server, a database or data analytics service, etc. The storage controllers **125***a*, **125***b* may provide services through some number of network interfaces (e.g., **126***a*-*d*) to host computers **127***a*-*n* outside of the storage system **124**. Storage controllers **125***a*, **125***b* may provide integrated services or an application entirely within the storage system **124**, forming a converged storage and compute system. The storage controllers **125***a*, **125***b* may utilize the fast write memory within or across storage devices **119***a*-*d* to journal in progress operations to ensure the operations are not lost on a power failure, storage controller removal, storage controller or storage system shutdown, or some fault of one or more software or hardware components within the storage system **124**.

[0075] In one embodiment, controllers **125***a*, **125***b* operate as PCI masters to one or the other PCI buses **128***a*, **128***b*. In another embodiment, **128***a* and **128***b* may be based on other communications standards (e.g., HyperTransport, InfiniBand, etc.). Other storage system embodiments may operate storage controllers **125***a*, **125***b* as multi-masters for both PCI buses **128***a*, **128***b*. Alternately, a PCI/NVMe/NVMf switching infrastructure or fabric may connect multiple storage controllers. Some storage system embodiments may allow storage devices to communicate with each other directly rather than communicating only with storage controllers. In one embodiment, a storage device controller **119***a* may be operable under direction from a storage controller **125***a* to synthesize and transfer data to be stored into Flash memory devices from data that has been stored in RAM (e.g., RAM **121** of FIG. **1**C). For example, a recalculated version of RAM content may be transferred after a storage controller has determined that an operation has fully committed across the storage system, or when fast-write memory on the device has reached a certain used capacity, or after a certain amount of time, to ensure improve safety of the data or to release addressable fast-write capacity for reuse. This mechanism may be used, for example, to avoid a second transfer over a bus (e.g., **128***a*, **128***b*) from the storage controllers **125***a*, **125***b*. In one embodiment, a recalculation may include compressing data, attaching indexing or other metadata, combining multiple data segments together, performing erasure code calculations, etc.

[0076] In one embodiment, under direction from a storage controller **125***a*, **125***b*, a storage device controller **119***a*, **119***b* may be operable to calculate and transfer data to other storage devices from data stored in RAM (e.g., RAM **121** of FIG. **1**C) without involvement of the storage controllers **125***a*, **125***b*. This operation may be used to mirror data stored in one controller **125***a* to another controller **125***b*, or it could be used to offload compression, data aggregation, and/or erasure coding calculations and transfers to storage devices to reduce load on storage controllers or the storage controller interface **129***a*, **129***b* to the PCI bus **128***a*, **128***b*.

[0077] A storage device controller **119**A-D may include mechanisms for implementing high availability primitives for use by other parts of a storage system external to the Dual PCI storage device **118**. For example, reservation or exclusion primitives may be provided so that, in a storage system with two storage controllers providing a highly available storage service, one storage controller may prevent the other storage controller from accessing or continuing to access the storage device. This could be used, for example, in cases where one controller detects that the other controller is not functioning properly or where the interconnect between the two storage controllers may itself not be functioning properly.

[0078] In one embodiment, a storage system for use with Dual PCI direct mapped storage devices with separately addressable fast write storage includes systems that manage erase blocks or groups of erase blocks as allocation units for storing data on behalf of the storage service, or for storing metadata (e.g., indexes, logs, etc.) associated with the storage service, or for proper management of the storage system itself. Flash pages, which may be a few kilobytes in size, may be written as data arrives or as the storage system is to persist data for long intervals of time (e.g., above a defined threshold of time). To commit data more quickly, or to reduce the number of writes to the Flash memory devices, the storage controllers may first write data into the separately addressable fast write storage on one more storage devices.

[0079] In one embodiment, the storage controllers **125***a*, **125***b* may initiate the use of erase blocks within and across storage devices (e.g., 118) in accordance with an age and expected remaining lifespan of the storage devices, or based on other statistics. The storage controllers **125***a*, **125***b* may initiate garbage collection and data migration data between storage devices in accordance with pages that are no longer needed as well as to manage Flash page and erase block lifespans and to manage overall system performance.

[0080] In one embodiment, the storage system **124** may utilize mirroring and/or erasure coding schemes as part of storing data into addressable fast write storage and/or as part of writing data into allocation units associated with erase blocks. Erasure codes may be used across storage devices, as well as within erase blocks or allocation units, or within and across Flash memory devices on a single storage device, to provide redundancy against single or multiple storage device failures or to protect against internal corruptions of Flash memory pages resulting from Flash memory operations or from degradation of Flash memory cells. Mirroring and erasure coding at various levels may be used to recover from multiple types of failures that occur separately or in combination.

[0081] The embodiments depicted with reference to FIGS. **2**A-G illustrate a storage cluster that stores user data, such as user data originating from one or more user or client systems or other sources external to the storage cluster. The storage cluster distributes user data across storage nodes housed within a chassis, or across multiple chassis, using erasure coding and redundant copies of metadata. Erasure coding refers to a method of data protection or reconstruction in which data is stored across a set of different locations, such as disks, storage nodes or geographic locations. Flash memory is one type of solid-state memory that may be integrated with the embodiments, although the embodiments may be extended to other types of solid-state memory or other storage medium, including non-solid state memory. Control of storage locations and workloads are distributed across the storage locations in a clustered peer-to-peer system. Tasks such as mediating communications between the various storage nodes, detecting when a storage node has become unavailable, and balancing I/Os (inputs and outputs) across the various storage nodes, are all handled on a distributed basis. Data is laid out or distributed across multiple storage nodes in data fragments or stripes that support data recovery in some embodiments. Ownership of data can be reassigned within a cluster, independent of input and output patterns. This architecture described in more detail below allows a storage node in the cluster to fail, with the system remaining operational, since the data can be reconstructed from other storage nodes and thus remain available for input and output operations. In various embodiments, a storage node may be referred to as a cluster node, a blade, or a server.

[0082] The storage cluster may be contained within a chassis, i.e., an enclosure housing one or more storage nodes. A mechanism to provide power to each storage node, such as a power distribution bus, and a communication mechanism, such as a communication bus that enables communication between the storage nodes are included within the chassis. The storage cluster can run as an independent system in one location according to some embodiments. In one embodiment, a chassis contains at least two instances of both the power distribution and the communication bus which may be enabled or disabled independently. The internal communication bus may be an Ethernet bus, however, other technologies such as PCIe, InfiniBand, and others, are equally suitable. The chassis provides a port for an external communication bus for enabling communication between multiple chassis, directly or through a switch, and with client systems. The external communication may use a technology such as Ethernet, InfiniBand, Fibre Channel, etc. In some embodiments, the external communication bus uses different communication bus technologies for inter-chassis and client communication. If a switch is deployed within or between chassis, the switch may act as a translation between multiple protocols or technologies. When multiple chassis are connected to define a storage cluster, the storage cluster may be accessed by a client using either proprietary interfaces or standard interfaces such as network file system ('NFS'), common internet file system (CIFS'), small computer system interface (SCSI') or hypertext transfer protocol ('HTTP'). Translation from the client protocol may occur at the switch, chassis external communication bus or within each storage node. In some embodiments, multiple chassis may be coupled or connected to each other through an aggregator switch. A portion and/or all of the coupled or connected chassis may be designated as a storage cluster. As discussed above, each chassis can have multiple blades, each

blade has a media access control ('MAC') address, but the storage cluster is presented to an external network as having a single cluster IP address and a single MAC address in some embodiments.

[0083] Each storage node may be one or more storage servers and each storage server is connected to one or more non-volatile solid state memory units, which may be referred to as storage units or storage devices. One embodiment includes a single storage server in each storage node and between one to eight non-volatile solid state memory units, however this one example is not meant to be limiting. The storage server may include a processor, DRAM and interfaces for the internal communication bus and power distribution for each of the power buses. Inside the storage node, the interfaces and storage unit share a communication bus, e.g., PCI Express, in some embodiments. The non-volatile solid state memory units may directly access the internal communication bus interface through a storage node communication bus, or request the storage node to access the bus interface. The non-volatile solid state memory unit contains an embedded CPU, solid state storage controller, and a quantity of solid state mass storage, e.g., between 2-32 terabytes ('TB') in some embodiments. An embedded volatile storage medium, such as DRAM, and an energy reserve apparatus are included in the non-volatile solid state memory unit. In some embodiments, the energy reserve apparatus is a capacitor, super-capacitor, or battery that enables transferring a subset of DRAM contents to a stable storage medium in the case of power loss. In some embodiments, the non-volatile solid state memory unit is constructed with a storage class memory, such as phase change or magnetoresistive random access memory ('MUM') that substitutes for DRAM and enables a reduced power hold-up apparatus.

[0084] One of many features of the storage nodes and non-volatile solid state storage is the ability to proactively rebuild data in a storage cluster. The storage nodes and non-volatile solid state storage can determine when a storage node or non-volatile solid state storage in the storage cluster is unreachable, independent of whether there is an attempt to read data involving that storage node or non-volatile solid state storage. The storage nodes and non-volatile solid state storage then cooperate to recover and rebuild the data in at least partially new locations. This constitutes a proactive rebuild, in that the system rebuilds data without waiting until the data is needed for a read access initiated from a client system employing the storage cluster. These and further details of the storage memory and operation thereof are discussed below.

[0085] FIG. **2**A is a perspective view of a storage cluster **161**, with multiple storage nodes **150** and internal solid-state memory coupled to each storage node to provide network attached storage or storage area network, in accordance with some embodiments. A network attached storage, storage area network, or a storage cluster, or other storage memory, could include one or more storage clusters **161**, each having one or more storage nodes **150**, in a flexible and reconfigurable arrangement of both the physical components and the amount of storage memory provided thereby. The storage cluster **161** is designed to fit in a rack, and one or more racks can be set up and populated as desired for the storage memory. The storage cluster **161** has a chassis **138** having multiple slots **142**. It should be appreciated that chassis **138** may be referred to as a housing, enclosure, or rack unit. In one embodiment, the chassis **138** has fourteen slots **142**, although other numbers of slots are readily devised. For example, some embodiments have four slots, eight slots, sixteen slots, thirty-two slots, or other suitable number of slots. Each slot **142** can accommodate one storage node **150** in some embodiments. Chassis **138** includes flaps **148** that can be utilized to mount the chassis **138** on a rack. Fans **144** provide air circulation for cooling of the storage nodes **150** and components thereof, although other cooling components could be used, or an embodiment could be devised without cooling components. A switch fabric **146** couples storage nodes **150** within chassis **138** together and to a network for communication to the memory. In an embodiment depicted in herein, the slots **142** to the left of the switch fabric **146** and fans **144** are shown occupied by storage nodes **150**, while the slots **142** to the right of the switch fabric **146** and fans **144** are empty and available for insertion of storage node **150** for illustrative purposes. This configuration is one example, and one or more storage nodes **150** could occupy the slots **142** in various further arrangements. The storage node arrangements need not be sequential or adjacent in some embodiments. Storage nodes **150** are hot pluggable, meaning that a storage node **150** can be inserted into a slot **142** in the chassis **138**, or removed from a slot **142**, without stopping or powering down the system. Upon insertion or removal of storage node **150** from slot **142**, the system automatically reconfigures in order to recognize and adapt to the change. Reconfiguration, in some embodiments, includes restoring redundancy and/or rebalancing data or load.

[0086] Each storage node **150** can have multiple components. In the embodiment shown here, the storage node **150** includes a printed circuit board **159** populated by a CPU **156**, i.e., processor, a memory **154** coupled to the CPU **156**, and a non-volatile solid state storage **152** coupled to the CPU **156**, although other mountings and/or components could be used in further embodiments. The memory **154** has instructions which are executed by the CPU **156** and/or data operated on by the CPU **156**. As further explained below, the non-volatile solid state storage **152** includes flash or, in further embodiments, other types of solid-state memory.

[0087] Referring to FIG. **2**A, storage cluster **161** is scalable, meaning that storage capacity with non-uniform storage sizes is readily added, as described above. One or more storage nodes **150** can be plugged into or removed from each chassis and the storage cluster self-configures in some embodiments. Plug-in storage nodes **150**, whether installed in a chassis as delivered or later added, can have different sizes. For example, in one embodiment a storage node **150** can have any multiple of 4 TB, e.g., 8 TB, 12 TB, 16 TB, 32 TB, etc. In further embodiments, a storage node **150** could have any multiple of other storage amounts or capacities. Storage capacity of each storage node **150** is broadcast, and influences decisions of how to stripe the data. For maximum storage efficiency, an embodiment can self-configure as wide as possible in the stripe, subject to a predetermined requirement of continued operation with loss of up to one, or up to two, non-volatile solid state storage units **152** or storage nodes **150** within the chassis.

[0088] FIG. **2**B is a block diagram showing a communications interconnect **173** and power distribution bus **172** coupling multiple storage nodes **150**. Referring back to FIG. **2**A, the communications interconnect **173** can be included in or implemented with the switch fabric **146** in some embodi-

ments. Where multiple storage clusters **161** occupy a rack, the communications interconnect **173** can be included in or implemented with a top of rack switch, in some embodiments. As illustrated in FIG. **2**B, storage cluster **161** is enclosed within a single chassis **138**. External port **176** is coupled to storage nodes **150** through communications interconnect **173**, while external port **174** is coupled directly to a storage node. External power port **178** is coupled to power distribution bus **172**. Storage nodes **150** may include varying amounts and differing capacities of non-volatile solid state storage **152** as described with reference to FIG. **2**A. In addition, one or more storage nodes **150** may be a compute only storage node as illustrated in FIG. **2**B. Authorities **168** are implemented on the non-volatile solid state storages **152**, for example as lists or other data structures stored in memory. In some embodiments the authorities are stored within the non-volatile solid state storage **152** and supported by software executing on a controller or other processor of the non-volatile solid state storage **152**. In a further embodiment, authorities **168** are implemented on the storage nodes **150**, for example as lists or other data structures stored in the memory **154** and supported by software executing on the CPU **156** of the storage node **150**. Authorities **168** control how and where data is stored in the non-volatile solid state storages **152** in some embodiments. This control assists in determining which type of erasure coding scheme is applied to the data, and which storage nodes **150** have which portions of the data. Each authority **168** may be assigned to a non-volatile solid state storage **152**. Each authority may control a range of inode numbers, segment numbers, or other data identifiers which are assigned to data by a file system, by the storage nodes **150**, or by the non-volatile solid state storage **152**, in various embodiments.

[0089] Every piece of data, and every piece of metadata, has redundancy in the system in some embodiments. In addition, every piece of data and every piece of metadata has an owner, which may be referred to as an authority. If that authority is unreachable, for example through failure of a storage node, there is a plan of succession for how to find that data or that metadata. In various embodiments, there are redundant copies of authorities **168**. Authorities **168** have a relationship to storage nodes **150** and non-volatile solid state storage **152** in some embodiments. Each authority **168**, covering a range of data segment numbers or other identifiers of the data, may be assigned to a specific non-volatile solid state storage **152**. In some embodiments the authorities **168** for all of such ranges are distributed over the non-volatile solid state storages **152** of a storage cluster. Each storage node **150** has a network port that provides access to the non-volatile solid state storage(s) **152** of that storage node **150**. Data can be stored in a segment, which is associated with a segment number and that segment number is an indirection for a configuration of a RAID (redundant array of independent disks) stripe in some embodiments. The assignment and use of the authorities **168** thus establishes an indirection to data. Indirection may be referred to as the ability to reference data indirectly, in this case via an authority **168**, in accordance with some embodiments. A segment identifies a set of non-volatile solid state storage **152** and a local identifier into the set of non-volatile solid state storage **152** that may contain data. In some embodiments, the local identifier is an offset into the device and may be reused sequentially by multiple segments. In other embodiments the local identifier is unique for a specific segment and never reused. The offsets in the non-volatile solid state storage **152** are applied to locating data for writing to or reading from the non-volatile solid state storage **152** (in the form of a RAID stripe). Data is striped across multiple units of non-volatile solid state storage **152**, which may include or be different from the non-volatile solid state storage **152** having the authority **168** for a particular data segment.

[0090] If there is a change in where a particular segment of data is located, e.g., during a data move or a data reconstruction, the authority **168** for that data segment should be consulted, at that non-volatile solid state storage **152** or storage node **150** having that authority **168**. In order to locate a particular piece of data, embodiments calculate a hash value for a data segment or apply an inode number or a data segment number. The output of this operation points to a non-volatile solid state storage **152** having the authority **168** for that particular piece of data. In some embodiments there are two stages to this operation. The first stage maps an entity identifier (ID), e.g., a segment number, inode number, or directory number to an authority identifier. This mapping may include a calculation such as a hash or a bit mask. The second stage is mapping the authority identifier to a particular non-volatile solid state storage **152**, which may be done through an explicit mapping. The operation is repeatable, so that when the calculation is performed, the result of the calculation repeatably and reliably points to a particular non-volatile solid state storage **152** having that authority **168**. The operation may include the set of reachable storage nodes as input. If the set of reachable non-volatile solid state storage units changes the optimal set changes. In some embodiments, the persisted value is the current assignment (which is always true) and the calculated value is the target assignment the cluster will attempt to reconfigure towards. This calculation may be used to determine the optimal non-volatile solid state storage **152** for an authority in the presence of a set of non-volatile solid state storage **152** that are reachable and constitute the same cluster. The calculation also determines an ordered set of peer non-volatile solid state storage **152** that will also record the authority to non-volatile solid state storage mapping so that the authority may be determined even if the assigned non-volatile solid state storage is unreachable. A duplicate or substitute authority **168** may be consulted if a specific authority **168** is unavailable in some embodiments.

[0091] With reference to FIGS. **2**A and **2**B, two of the many tasks of the CPU **156** on a storage node **150** are to break up write data, and reassemble read data. When the system has determined that data is to be written, the authority **168** for that data is located as above. When the segment ID for data is already determined the request to write is forwarded to the non-volatile solid state storage **152** currently determined to be the host of the authority **168** determined from the segment. The host CPU **156** of the storage node **150**, on which the non-volatile solid state storage **152** and corresponding authority **168** reside, then breaks up or shards the data and transmits the data out to various non-volatile solid state storage **152**. The transmitted data is written as a data stripe in accordance with an erasure coding scheme. In some embodiments, data is requested to be pulled, and in other embodiments, data is pushed. In reverse, when data is read, the authority **168** for the segment ID containing the data is located as described above. The host CPU **156** of the storage node **150** on which the non-volatile

solid state storage **152** and corresponding authority **168** reside requests the data from the non-volatile solid state storage and corresponding storage nodes pointed to by the authority. In some embodiments the data is read from flash storage as a data stripe. The host CPU **156** of storage node **150** then reassembles the read data, correcting any errors (if present) according to the appropriate erasure coding scheme, and forwards the reassembled data to the network. In further embodiments, some or all of these tasks can be handled in the non-volatile solid state storage **152**. In some embodiments, the segment host requests the data be sent to storage node **150** by requesting pages from storage and then sending the data to the storage node making the original request.

[0092] In embodiments, authorities **168** operate to determine how operations will proceed against particular logical elements. Each of the logical elements may be operated on through a particular authority across a plurality of storage controllers of a storage system. The authorities **168** may communicate with the plurality of storage controllers so that the plurality of storage controllers collectively perform operations against those particular logical elements.

[0093] In embodiments, logical elements could be, for example, files, directories, object buckets, individual objects, delineated parts of files or objects, other forms of key-value pair databases, or tables. In embodiments, performing an operation can involve, for example, ensuring consistency, structural integrity, and/or recoverability with other operations against the same logical element, reading metadata and data associated with that logical element, determining what data should be written durably into the storage system to persist any changes for the operation, or where metadata and data can be determined to be stored across modular storage devices attached to a plurality of the storage controllers in the storage system.

[0094] In some embodiments the operations are token based transactions to efficiently communicate within a distributed system. Each transaction may be accompanied by or associated with a token, which gives permission to execute the transaction. The authorities **168** are able to maintain a pre-transaction state of the system until completion of the operation in some embodiments. The token based communication may be accomplished without a global lock across the system, and also enables restart of an operation in case of a disruption or other failure.

[0095] In some systems, for example in UNIX-style file systems, data is handled with an index node or inode, which specifies a data structure that represents an object in a file system. The object could be a file or a directory, for example. Metadata may accompany the object, as attributes such as permission data and a creation timestamp, among other attributes. A segment number could be assigned to all or a portion of such an object in a file system. In other systems, data segments are handled with a segment number assigned elsewhere. For purposes of discussion, the unit of distribution is an entity, and an entity can be a file, a directory or a segment. That is, entities are units of data or metadata stored by a storage system. Entities are grouped into sets called authorities. Each authority has an authority owner, which is a storage node that has the exclusive right to update the entities in the authority. In other words, a storage node contains the authority, and that the authority, in turn, contains entities.

[0096] A segment is a logical container of data in accordance with some embodiments. A segment is an address space between medium address space and physical flash locations, i.e., the data segment number, are in this address space. Segments may also contain meta-data, which enable data redundancy to be restored (rewritten to different flash locations or devices) without the involvement of higher level software. In one embodiment, an internal format of a segment contains client data and medium mappings to determine the position of that data. Each data segment is protected, e.g., from memory and other failures, by breaking the segment into a number of data and parity shards, where applicable. The data and parity shards are distributed, i.e., striped, across non-volatile solid state storage **152** coupled to the host CPUs **156** (See FIGS. **2**E and **2**G) in accordance with an erasure coding scheme. Usage of the term segments refers to the container and its place in the address space of segments in some embodiments. Usage of the term stripe refers to the same set of shards as a segment and includes how the shards are distributed along with redundancy or parity information in accordance with some embodiments.

[0097] A series of address-space transformations takes place across an entire storage system. At the top are the directory entries (file names) which link to an inode. Modes point into medium address space, where data is logically stored. Medium addresses may be mapped through a series of indirect mediums to spread the load of large files, or implement data services like deduplication or snapshots. Medium addresses may be mapped through a series of indirect mediums to spread the load of large files, or implement data services like deduplication or snapshots. Segment addresses are then translated into physical flash locations. Physical flash locations have an address range bounded by the amount of flash in the system in accordance with some embodiments. Medium addresses and segment addresses are logical containers, and in some embodiments use a 128 bit or larger identifier so as to be practically infinite, with a likelihood of reuse calculated as longer than the expected life of the system. Addresses from logical containers are allocated in a hierarchical fashion in some embodiments. Initially, each non-volatile solid state storage unit **152** may be assigned a range of address space. Within this assigned range, the non-volatile solid state storage **152** is able to allocate addresses without synchronization with other non-volatile solid state storage **152**.

[0098] Data and metadata is stored by a set of underlying storage layouts that are optimized for varying workload patterns and storage devices. These layouts incorporate multiple redundancy schemes, compression formats and index algorithms. Some of these layouts store information about authorities and authority masters, while others store file metadata and file data. The redundancy schemes include error correction codes that tolerate corrupted bits within a single storage device (such as a NAND flash chip), erasure codes that tolerate the failure of multiple storage nodes, and replication schemes that tolerate data center or regional failures. In some embodiments, low density parity check (IDPC') code is used within a single storage unit. Reed-Solomon encoding is used within a storage cluster, and mirroring is used within a storage grid in some embodiments. Metadata may be stored using an ordered log structured index (such as a Log Structured Merge Tree), and large data may not be stored in a log structured layout.

[0099] In order to maintain consistency across multiple copies of an entity, the storage nodes agree implicitly on two things through calculations: (1) the authority that contains

the entity, and (2) the storage node that contains the authority. The assignment of entities to authorities can be done by pseudo randomly assigning entities to authorities, by splitting entities into ranges based upon an externally produced key, or by placing a single entity into each authority. Examples of pseudorandom schemes are linear hashing and the Replication Under Scalable Hashing ('RUSH') family of hashes, including Controlled Replication Under Scalable Hashing ('CRUSH'). In some embodiments, pseudo-random assignment is utilized only for assigning authorities to nodes because the set of nodes can change. The set of authorities cannot change so any subjective function may be applied in these embodiments. Some placement schemes automatically place authorities on storage nodes, while other placement schemes rely on an explicit mapping of authorities to storage nodes. In some embodiments, a pseudorandom scheme is utilized to map from each authority to a set of candidate authority owners. A pseudorandom data distribution function related to CRUSH may assign authorities to storage nodes and create a list of where the authorities are assigned. Each storage node has a copy of the pseudorandom data distribution function, and can arrive at the same calculation for distributing, and later finding or locating an authority. Each of the pseudorandom schemes requires the reachable set of storage nodes as input in some embodiments in order to conclude the same target nodes. Once an entity has been placed in an authority, the entity may be stored on physical devices so that no expected failure will lead to unexpected data loss. In some embodiments, rebalancing algorithms attempt to store the copies of all entities within an authority in the same layout and on the same set of machines.

[0100] Examples of expected failures include device failures, stolen machines, datacenter fires, and regional disasters, such as nuclear or geological events. Different failures lead to different levels of acceptable data loss. In some embodiments, a stolen storage node impacts neither the security nor the reliability of the system, while depending on system configuration, a regional event could lead to no loss of data, a few seconds or minutes of lost updates, or even complete data loss.

[0101] In the embodiments, the placement of data for storage redundancy is independent of the placement of authorities for data consistency. In some embodiments, storage nodes that contain authorities do not contain any persistent storage. Instead, the storage nodes are connected to non-volatile solid state storage units that do not contain authorities. The communications interconnect between storage nodes and non-volatile solid state storage units consists of multiple communication technologies and has non-uniform performance and fault tolerance characteristics. In some embodiments, as mentioned above, non-volatile solid state storage units are connected to storage nodes via PCI express, storage nodes are connected together within a single chassis using Ethernet backplane, and chassis are connected together to form a storage cluster. Storage clusters are connected to clients using Ethernet or fiber channel in some embodiments. If multiple storage clusters are configured into a storage grid, the multiple storage clusters are connected using the Internet or other long-distance networking links, such as a "metro scale" link or private link that does not traverse the internet.

[0102] Authority owners have the exclusive right to modify entities, to migrate entities from one non-volatile solid state storage unit to another non-volatile solid state storage unit, and to add and remove copies of entities. This allows for maintaining the redundancy of the underlying data. When an authority owner fails, is going to be decommissioned, or is overloaded, the authority is transferred to a new storage node. Transient failures make it non-trivial to ensure that all non-faulty machines agree upon the new authority location. The ambiguity that arises due to transient failures can be achieved automatically by a consensus protocol such as Paxos, hot-warm failover schemes, via manual intervention by a remote system administrator, or by a local hardware administrator (such as by physically removing the failed machine from the cluster, or pressing a button on the failed machine). In some embodiments, a consensus protocol is used, and failover is automatic. If too many failures or replication events occur in too short a time period, the system goes into a self-preservation mode and halts replication and data movement activities until an administrator intervenes in accordance with some embodiments.

[0103] As authorities are transferred between storage nodes and authority owners update entities in their authorities, the system transfers messages between the storage nodes and non-volatile solid state storage units. With regard to persistent messages, messages that have different purposes are of different types. Depending on the type of the message, the system maintains different ordering and durability guarantees. As the persistent messages are being processed, the messages are temporarily stored in multiple durable and non-durable storage hardware technologies. In some embodiments, messages are stored in RAM, NVRAM and on NAND flash devices, and a variety of protocols are used in order to make efficient use of each storage medium. Latency-sensitive client requests may be persisted in replicated NVRAM, and then later NAND, while background rebalancing operations are persisted directly to NAND.

[0104] Persistent messages are persistently stored prior to being transmitted. This allows the system to continue to serve client requests despite failures and component replacement. Although many hardware components contain unique identifiers that are visible to system administrators, manufacturer, hardware supply chain and ongoing monitoring quality control infrastructure, applications running on top of the infrastructure address virtualize addresses. These virtualized addresses do not change over the lifetime of the storage system, regardless of component failures and replacements. This allows each component of the storage system to be replaced over time without reconfiguration or disruptions of client request processing, i.e., the system supports non-disruptive upgrades.

[0105] In some embodiments, the virtualized addresses are stored with sufficient redundancy. A continuous monitoring system correlates hardware and software status and the hardware identifiers. This allows detection and prediction of failures due to faulty components and manufacturing details. The monitoring system also enables the proactive transfer of authorities and entities away from impacted devices before failure occurs by removing the component from the critical path in some embodiments.

[0106] FIG. **2**C is a multiple level block diagram, showing contents of a storage node **150** and contents of a non-volatile solid state storage **152** of the storage node **150**. Data is communicated to and from the storage node **150** by a network interface controller ('NIC') **202** in some embodi-

ments. Each storage node **150** has a CPU **156**, and one or more non-volatile solid state storage **152**, as discussed above. Moving down one level in FIG. **2**C, each non-volatile solid state storage **152** has a relatively fast non-volatile solid state memory, such as nonvolatile random access memory ('NVRAM') **204**, and flash memory **206**. In some embodiments, NVRAM **204** may be a component that does not require program/erase cycles (DRAM, MRAM, PCM), and can be a memory that can support being written vastly more often than the memory is read from. Moving down another level in FIG. **2**C, the NVRAM **204** is implemented in one embodiment as high speed volatile memory, such as dynamic random access memory (DRAM) **216**, backed up by energy reserve **218**. Energy reserve **218** provides sufficient electrical power to keep the DRAM **216** powered long enough for contents to be transferred to the flash memory **206** in the event of power failure. In some embodiments, energy reserve **218** is a capacitor, super-capacitor, battery, or other device, that supplies a suitable supply of energy sufficient to enable the transfer of the contents of DRAM **216** to a stable storage medium in the case of power loss. The flash memory **206** is implemented as multiple flash dies **222**, which may be referred to as packages of flash dies **222** or an array of flash dies **222**. It should be appreciated that the flash dies **222** could be packaged in any number of ways, with a single die per package, multiple dies per package (i.e. multichip packages), in hybrid packages, as bare dies on a printed circuit board or other substrate, as encapsulated dies, etc. In the embodiment shown, the non-volatile solid state storage **152** has a controller **212** or other processor, and an input output (I/O) port **210** coupled to the controller **212**. I/O port **210** is coupled to the CPU **156** and/or the network interface controller **202** of the flash storage node **150**. Flash input output (I/O) port **220** is coupled to the flash dies **222**, and a direct memory access unit (DMA) **214** is coupled to the controller **212**, the DRAM **216** and the flash dies **222**. In the embodiment shown, the I/O port **210**, controller **212**, DMA unit **214** and flash I/O port **220** are implemented on a programmable logic device ('PLD') **208**, e.g., an FPGA. In this embodiment, each flash die **222** has pages, organized as sixteen kB (kilobyte) pages **224**, and a register **226** through which data can be written to or read from the flash die **222**. In further embodiments, other types of solid-state memory are used in place of, or in addition to flash memory illustrated within flash die **222**.

[0107] Storage clusters **161**, in various embodiments as disclosed herein, can be contrasted with storage arrays in general. The storage nodes **150** are part of a collection that creates the storage cluster **161**. Each storage node **150** owns a slice of data and computing required to provide the data. Multiple storage nodes **150** cooperate to store and retrieve the data. Storage memory or storage devices, as used in storage arrays in general, are less involved with processing and manipulating the data. Storage memory or storage devices in a storage array receive commands to read, write, or erase data. The storage memory or storage devices in a storage array are not aware of a larger system in which they are embedded, or what the data means. Storage memory or storage devices in storage arrays can include various types of storage memory, such as RAM, solid state drives, hard disk drives, etc. The storage units **152** described herein have multiple interfaces active simultaneously and serving multiple purposes. In some embodiments, some of the functionality of a storage node **150** is shifted into a storage unit **152**, transforming the storage unit **152** into a combination of storage unit **152** and storage node **150**. Placing computing (relative to storage data) into the storage unit **152** places this computing closer to the data itself. The various system embodiments have a hierarchy of storage node layers with different capabilities. By contrast, in a storage array, a controller owns and knows everything about all of the data that the controller manages in a shelf or storage devices. In a storage cluster **161**, as described herein, multiple controllers in multiple storage units **152** and/or storage nodes **150** cooperate in various ways (e.g., for erasure coding, data sharding, metadata communication and redundancy, storage capacity expansion or contraction, data recovery, and so on).

[0108] FIG. **2**D shows a storage server environment, which uses embodiments of the storage nodes **150** and storage units **152** of FIGS. **2**A-C. In this version, each storage unit **152** has a processor such as controller **212** (see FIG. **2**C), an FPGA, flash memory **206**, and NVRAM **204** (which is super-capacitor backed DRAM **216**, see FIGS. **2**B and **2**C) on a PCIe (peripheral component interconnect express) board in a chassis **138** (see FIG. **2**A). The storage unit **152** may be implemented as a single board containing storage, and may be the largest tolerable failure domain inside the chassis. In some embodiments, up to two storage units **152** may fail and the device will continue with no data loss.

[0109] The physical storage is divided into named regions based on application usage in some embodiments. The NVRAM **204** is a contiguous block of reserved memory in the storage unit **152** DRAM **216**, and is backed by NAND flash. NVRAM **204** is logically divided into multiple memory regions written for two as spool (e.g., spool region). Space within the NVRAM **204** spools is managed by each authority **168** independently. Each device provides an amount of storage space to each authority **168**. That authority **168** further manages lifetimes and allocations within that space. Examples of a spool include distributed transactions or notions. When the primary power to a storage unit **152** fails, onboard super-capacitors provide a short duration of power hold up. During this holdup interval, the contents of the NVRAM **204** are flushed to flash memory **206**. On the next power-on, the contents of the NVRAM **204** are recovered from the flash memory **206**.

[0110] As for the storage unit controller, the responsibility of the logical "controller" is distributed across each of the blades containing authorities **168**. This distribution of logical control is shown in FIG. **2**D as a host controller **242**, mid-tier controller **244** and storage unit controller(s) **246**. Management of the control plane and the storage plane are treated independently, although parts may be physically co-located on the same blade. Each authority **168** effectively serves as an independent controller. Each authority **168** provides its own data and metadata structures, its own background workers, and maintains its own lifecycle.

[0111] FIG. **2**E is a blade **252** hardware block diagram, showing a control plane **254**, compute and storage planes **256**, **258**, and authorities **168** interacting with underlying physical resources, using embodiments of the storage nodes **150** and storage units **152** of FIGS. **2**A-C in the storage server environment of FIG. **2**D. The control plane **254** is partitioned into a number of authorities **168** which can use the compute resources in the compute plane **256** to run on any of the blades **252**. The storage plane **258** is partitioned into a set of devices, each of which provides access to flash

206 and NVRAM **204** resources. In one embodiment, the compute plane **256** may perform the operations of a storage array controller, as described herein, on one or more devices of the storage plane **258** (e.g., a storage array).

[0112] In the compute and storage planes **256**, **258** of FIG. **2**E, the authorities **168** interact with the underlying physical resources (i.e., devices). From the point of view of an authority **168**, its resources are striped over all of the physical devices. From the point of view of a device, it provides resources to all authorities **168**, irrespective of where the authorities happen to run. Each authority **168** has allocated or has been allocated one or more partitions **260** of storage memory in the storage units **152**, e.g. partitions **260** in flash memory **206** and NVRAM **204**. Each authority **168** uses those allocated partitions **260** that belong to it, for writing or reading user data. Authorities can be associated with differing amounts of physical storage of the system. For example, one authority **168** could have a larger number of partitions **260** or larger sized partitions **260** in one or more storage units **152** than one or more other authorities **168**.

[0113] FIG. **2**F depicts elasticity software layers in blades **252** of a storage cluster, in accordance with some embodiments. In the elasticity structure, elasticity software is symmetric, i.e., each blade's compute module **270** runs the three identical layers of processes depicted in FIG. **2**F. Storage managers **274** execute read and write requests from other blades **252** for data and metadata stored in local storage unit **152** NVRAM **204** and flash **206**. Authorities **168** fulfill client requests by issuing the necessary reads and writes to the blades **252** on whose storage units **152** the corresponding data or metadata resides. Endpoints **272** parse client connection requests received from switch fabric **146** supervisory software, relay the client connection requests to the authorities **168** responsible for fulfillment, and relay the authorities' **168** responses to clients. The symmetric three-layer structure enables the storage system's high degree of concurrency. Elasticity scales out efficiently and reliably in these embodiments. In addition, elasticity implements a unique scale-out technique that balances work evenly across all resources regardless of client access pattern, and maximizes concurrency by eliminating much of the need for inter-blade coordination that typically occurs with conventional distributed locking.

[0114] Still referring to FIG. **2**F, authorities **168** running in the compute modules **270** of a blade **252** perform the internal operations required to fulfill client requests. One feature of elasticity is that authorities **168** are stateless, i.e., they cache active data and metadata in their own blades' **252** DRAMs for fast access, but the authorities store every update in their NVRAM **204** partitions on three separate blades **252** until the update has been written to flash **206**. All the storage system writes to NVRAM **204** are in triplicate to partitions on three separate blades **252** in some embodiments. With triple-mirrored NVRAM **204** and persistent storage protected by parity and Reed-Solomon RAID checksums, the storage system can survive concurrent failure of two blades **252** with no loss of data, metadata, or access to either.

[0115] Because authorities **168** are stateless, they can migrate between blades **252**. Each authority **168** has a unique identifier. NVRAM **204** and flash **206** partitions are associated with authorities' **168** identifiers, not with the blades **252** on which they are running in some. Thus, when an authority **168** migrates, the authority **168** continues to manage the same storage partitions from its new location. When a new blade **252** is installed in an embodiment of the storage cluster, the system automatically rebalances load by: partitioning the new blade's **252** storage for use by the system's authorities **168**, migrating selected authorities **168** to the new blade **252**, starting endpoints **272** on the new blade **252** and including them in the switch fabric's **146** client connection distribution algorithm.

[0116] From their new locations, migrated authorities **168** persist the contents of their NVRAM **204** partitions on flash **206**, process read and write requests from other authorities **168**, and fulfill the client requests that endpoints **272** direct to them. Similarly, if a blade **252** fails or is removed, the system redistributes its authorities **168** among the system's remaining blades **252**. The redistributed authorities **168** continue to perform their original functions from their new locations.

[0117] FIG. **2**G depicts authorities **168** and storage resources in blades **252** of a storage cluster, in accordance with some embodiments. Each authority **168** is exclusively responsible for a partition of the flash **206** and NVRAM **204** on each blade **252**. The authority **168** manages the content and integrity of its partitions independently of other authorities **168**. Authorities **168** compress incoming data and preserve it temporarily in their NVRAM **204** partitions, and then consolidate, RAID-protect, and persist the data in segments of the storage in their flash **206** partitions. As the authorities **168** write data to flash **206**, storage managers **274** perform the necessary flash translation to optimize write performance and maximize media longevity. In the background, authorities **168** "garbage collect," or reclaim space occupied by data that clients have made obsolete by overwriting the data. It should be appreciated that since authorities' **168** partitions are disjoint, there is no need for distributed locking to execute client and writes or to perform background functions.

[0118] The embodiments described herein may utilize various software, communication and/or networking protocols. In addition, the configuration of the hardware and/or software may be adjusted to accommodate various protocols. For example, the embodiments may utilize Active Directory, which is a database based system that provides authentication, directory, policy, and other services in a WINDOWS™ environment. In these embodiments, LDAP (Lightweight Directory Access Protocol) is one example application protocol for querying and modifying items in directory service providers such as Active Directory. In some embodiments, a network lock manager ('NLM') is utilized as a facility that works in cooperation with the Network File System ('NFS') to provide a System V style of advisory file and record locking over a network. The Server Message Block ('SMB') protocol, one version of which is also known as Common Internet File System ('CIFS'), may be integrated with the storage systems discussed herein. SMP operates as an application-layer network protocol typically used for providing shared access to files, printers, and serial ports and miscellaneous communications between nodes on a network. SMB also provides an authenticated inter-process communication mechanism. AMAZON™ S3 (Simple Storage Service) is a web service offered by Amazon Web Services, and the systems described herein may interface with Amazon S3 through web services interfaces (REST (representational state transfer), SOAP (simple object access protocol), and BitTorrent). A RESTful API (application programming interface) breaks down a trans-

action to create a series of small modules. Each module addresses a particular underlying part of the transaction. The control or permissions provided with these embodiments, especially for object data, may include utilization of an access control list ('ACL'). The ACL is a list of permissions attached to an object and the ACL specifies which users or system processes are granted access to objects, as well as what operations are allowed on given objects. The systems may utilize Internet Protocol version 6 ('IPv6'), as well as IPv4, for the communications protocol that provides an identification and location system for computers on networks and routes traffic across the Internet. The routing of packets between networked systems may include Equal-cost multi-path routing ('ECMP'), which is a routing strategy where next-hop packet forwarding to a single destination can occur over multiple "best paths" which tie for top place in routing metric calculations. Multi-path routing can be used in conjunction with most routing protocols, because it is a per-hop decision limited to a single router. The software may support Multi-tenancy, which is an architecture in which a single instance of a software application serves multiple customers. Each customer may be referred to as a tenant. Tenants may be given the ability to customize some parts of the application, but may not customize the application's code, in some embodiments. The embodiments may maintain audit logs. An audit log is a document that records an event in a computing system. In addition to documenting what resources were accessed, audit log entries typically include destination and source addresses, a timestamp, and user login information for compliance with various regulations. The embodiments may support various key management policies, such as encryption key rotation. In addition, the system may support dynamic root passwords or some variation dynamically changing passwords.

[0119] FIG. **3**A sets forth a diagram of a storage system **306** that is coupled for data communications with a cloud services provider **302** in accordance with some embodiments of the present disclosure. Although depicted in less detail, the storage system **306** depicted in FIG. **3**A may be similar to the storage systems described above with reference to FIGS. **1**A-**1**D and FIGS. **2**A-**2**G. In some embodiments, the storage system **306** depicted in FIG. **3**A may be embodied as a storage system that includes imbalanced active/active controllers, as a storage system that includes balanced active/active controllers, as a storage system that includes active/active controllers where less than all of each controller's resources are utilized such that each controller has reserve resources that may be used to support failover, as a storage system that includes fully active/active controllers, as a storage system that includes dataset-segregated controllers, as a storage system that includes dual-layer architectures with front-end controllers and back-end integrated storage controllers, as a storage system that includes scale-out clusters of dual-controller arrays, as well as combinations of such embodiments.

[0120] In the example depicted in FIG. **3**A, the storage system **306** is coupled to the cloud services provider **302** via a data communications link **304**. The data communications link **304** may be embodied as a dedicated data communications link, as a data communications pathway that is provided through the use of one or data communications networks such as a wide area network ('WAN') or LAN, or as some other mechanism capable of transporting digital information between the storage system **306** and the cloud services provider **302**. Such a data communications link **304** may be fully wired, fully wireless, or some aggregation of wired and wireless data communications pathways. In such an example, digital information may be exchanged between the storage system **306** and the cloud services provider **302** via the data communications link **304** using one or more data communications protocols. For example, digital information may be exchanged between the storage system **306** and the cloud services provider **302** via the data communications link **304** using the handheld device transfer protocol ('HDTP'), hypertext transfer protocol ('HTTP'), internet protocol ('IP'), real-time transfer protocol ('RTP'), transmission control protocol ('TCP'), user datagram protocol ('UDP'), wireless application protocol ('WAP'), or other protocol.

[0121] The cloud services provider **302** depicted in FIG. **3**A may be embodied, for example, as a system and computing environment that provides a vast array of services to users of the cloud services provider **302** through the sharing of computing resources via the data communications link **304**. The cloud services provider **302** may provide on-demand access to a shared pool of configurable computing resources such as computer networks, servers, storage, applications and services, and so on. The shared pool of configurable resources may be rapidly provisioned and released to a user of the cloud services provider **302** with minimal management effort. Generally, the user of the cloud services provider **302** is unaware of the exact computing resources utilized by the cloud services provider **302** to provide the services. Although in many cases such a cloud services provider **302** may be accessible via the Internet, readers of skill in the art will recognize that any system that abstracts the use of shared resources to provide services to a user through any data communications link may be considered a cloud services provider **302**.

[0122] In the example depicted in FIG. **3**A, the cloud services provider **302** may be configured to provide a variety of services to the storage system **306** and users of the storage system **306** through the implementation of various service models. For example, the cloud services provider **302** may be configured to provide services through the implementation of an infrastructure as a service ('IaaS') service model, through the implementation of a platform as a service ('PaaS') service model, through the implementation of a software as a service ('SaaS') service model, through the implementation of an authentication as a service ('AaaS') service model, through the implementation of a storage as a service model where the cloud services provider **302** offers access to its storage infrastructure for use by the storage system **306** and users of the storage system **306**, and so on. Readers will appreciate that the cloud services provider **302** may be configured to provide additional services to the storage system **306** and users of the storage system **306** through the implementation of additional service models, as the service models described above are included only for explanatory purposes and in no way represent a limitation of the services that may be offered by the cloud services provider **302** or a limitation as to the service models that may be implemented by the cloud services provider **302**.

[0123] In the example depicted in FIG. **3**A, the cloud services provider **302** may be embodied, for example, as a private cloud, as a public cloud, or as a combination of a private cloud and public cloud. In an embodiment in which the cloud services provider **302** is embodied as a private

cloud, the cloud services provider **302** may be dedicated to providing services to a single organization rather than providing services to multiple organizations. In an embodiment where the cloud services provider **302** is embodied as a public cloud, the cloud services provider **302** may provide services to multiple organizations. In still alternative embodiments, the cloud services provider **302** may be embodied as a mix of a private and public cloud services with a hybrid cloud deployment.

[0124] Although not explicitly depicted in FIG. **3**A, readers will appreciate that a vast amount of additional hardware components and additional software components may be necessary to facilitate the delivery of cloud services to the storage system **306** and users of the storage system **306**. For example, the storage system **306** may be coupled to (or even include) a cloud storage gateway. Such a cloud storage gateway may be embodied, for example, as hardware-based or software-based appliance that is located on premise with the storage system **306**. Such a cloud storage gateway may operate as a bridge between local applications that are executing on the storage array **306** and remote, cloud-based storage that is utilized by the storage array **306**. Through the use of a cloud storage gateway, organizations may move primary iSCSI or NAS to the cloud services provider **302**, thereby enabling the organization to save space on their on-premises storage systems. Such a cloud storage gateway may be configured to emulate a disk array, a block-based device, a file server, or other storage system that can translate the SCSI commands, file server commands, or other appropriate command into REST-space protocols that facilitate communications with the cloud services provider **302**.

[0125] In order to enable the storage system **306** and users of the storage system **306** to make use of the services provided by the cloud services provider **302**, a cloud migration process may take place during which data, applications, or other elements from an organization's local systems (or even from another cloud environment) are moved to the cloud services provider **302**. In order to successfully migrate data, applications, or other elements to the cloud services provider's **302** environment, middleware such as a cloud migration tool may be utilized to bridge gaps between the cloud services provider's **302** environment and an organization's environment. Such cloud migration tools may also be configured to address potentially high network costs and long transfer times associated with migrating large volumes of data to the cloud services provider **302**, as well as addressing security concerns associated with sensitive data to the cloud services provider **302** over data communications networks. In order to further enable the storage system **306** and users of the storage system **306** to make use of the services provided by the cloud services provider **302**, a cloud orchestrator may also be used to arrange and coordinate automated tasks in pursuit of creating a consolidated process or workflow. Such a cloud orchestrator may perform tasks such as configuring various components, whether those components are cloud components or on-premises components, as well as managing the interconnections between such components. The cloud orchestrator can simplify the inter-component communication and connections to ensure that links are correctly configured and maintained.

[0126] In the example depicted in FIG. **3**A, and as described briefly above, the cloud services provider **302** may be configured to provide services to the storage system **306** and users of the storage system **306** through the usage of a SaaS service model, eliminating the need to install and run the application on local computers, which may simplify maintenance and support of the application. Such applications may take many forms in accordance with various embodiments of the present disclosure. For example, the cloud services provider **302** may be configured to provide access to data analytics applications to the storage system **306** and users of the storage system **306**. Such data analytics applications may be configured, for example, to receive vast amounts of telemetry data phoned home by the storage system **306**. Such telemetry data may describe various operating characteristics of the storage system **306** and may be analyzed for a vast array of purposes including, for example, to determine the health of the storage system **306**, to identify workloads that are executing on the storage system **306**, to predict when the storage system **306** will run out of various resources, to recommend configuration changes, hardware or software upgrades, workflow migrations, or other actions that may improve the operation of the storage system **306**.

[0127] The cloud services provider **302** may also be configured to provide access to virtualized computing environments to the storage system **306** and users of the storage system **306**. Such virtualized computing environments may be embodied, for example, as a virtual machine or other virtualized computer hardware platforms, virtual storage devices, virtualized computer network resources, and so on. Examples of such virtualized environments can include virtual machines that are created to emulate an actual computer, virtualized desktop environments that separate a logical desktop from a physical machine, virtualized file systems that allow uniform access to different types of concrete file systems, and many others.

[0128] Although the example depicted in FIG. **3**A illustrates the storage system **306** being coupled for data communications with the cloud services provider **302**, in other embodiments the storage system **306** may be part of a hybrid cloud deployment in which private cloud elements (e.g., private cloud services, on-premises infrastructure, and so on) and public cloud elements (e.g., public cloud services, infrastructure, and so on that may be provided by one or more cloud services providers) are combined to form a single solution, with orchestration among the various platforms. Such a hybrid cloud deployment may leverage hybrid cloud management software such as, for example, Azure™ Arc from Microsoft™, that centralize the management of the hybrid cloud deployment to any infrastructure and enable the deployment of services anywhere. In such an example, the hybrid cloud management software may be configured to create, update, and delete resources (both physical and virtual) that form the hybrid cloud deployment, to allocate compute and storage to specific workloads, to monitor workloads and resources for performance, policy compliance, updates and patches, security status, or to perform a variety of other tasks.

[0129] Readers will appreciate that by pairing the storage systems described herein with one or more cloud services providers, various offerings may be enabled. For example, disaster recovery as a service ('DRaaS') may be provided where cloud resources are utilized to protect applications and data from disruption caused by disaster, including in embodiments where the storage systems may serve as the primary data store. In such embodiments, a total system

backup may be taken that allows for business continuity in the event of system failure. In such embodiments, cloud data backup techniques (by themselves or as part of a larger DRaaS solution) may also be integrated into an overall solution that includes the storage systems and cloud services providers described herein.

[0130] The storage systems described herein, as well as the cloud services providers, may be utilized to provide a wide array of security features. For example, the storage systems may encrypt data at rest (and data may be sent to and from the storage systems encrypted) and may make use of Key Management-as-a-Service ('KMaaS') to manage encryption keys, keys for locking and unlocking storage devices, and so on. Likewise, cloud data security gateways or similar mechanisms may be utilized to ensure that data stored within the storage systems does not improperly end up being stored in the cloud as part of a cloud data backup operation. Furthermore, microsegmentation or identity-based-segmentation may be utilized in a data center that includes the storage systems or within the cloud services provider, to create secure zones in data centers and cloud deployments that enables the isolation of workloads from one another.

[0131] For further explanation, FIG. **3**B sets forth a diagram of a storage system **306** in accordance with some embodiments of the present disclosure. Although depicted in less detail, the storage system **306** depicted in FIG. **3**B may be similar to the storage systems described above with reference to FIGS. **1**A-**1**D and FIGS. **2**A-**2**G as the storage system may include many of the components described above.

[0132] The storage system **306** depicted in FIG. **3**B may include a vast amount of storage resources **308**, which may be embodied in many forms. For example, the storage resources **308** can include nano-RAM or another form of nonvolatile random access memory that utilizes carbon nanotubes deposited on a substrate, 3D crosspoint non-volatile memory, flash memory including single-level cell ('SLC') NAND flash, multi-level cell ('MLC') NAND flash, triple-level cell ('TLC') NAND flash, quad-level cell ('QLC') NAND flash, or others. Likewise, the storage resources **308** may include non-volatile magnetoresistive random-access memory ('MRAM'), including spin transfer torque ('STT') MRAM. The example storage resources **308** may alternatively include non-volatile phase-change memory ('PCM'), quantum memory that allows for the storage and retrieval of photonic quantum information, resistive random-access memory ('ReRAM'), storage class memory ('SCM'), or other form of storage resources, including any combination of resources described herein. Readers will appreciate that other forms of computer memories and storage devices may be utilized by the storage systems described above, including DRAM, SRAM, EEPROM, universal memory, and many others. The storage resources **308** depicted in FIG. **3**A may be embodied in a variety of form factors, including but not limited to, dual in-line memory modules ('DIMMs'), non-volatile dual in-line memory modules (NVDIMMs'), M.2, U.2, and others.

[0133] The storage resources **308** depicted in FIG. **3**B may include various forms of SCM. SCM may effectively treat fast, non-volatile memory (e.g., NAND flash) as an extension of DRAM such that an entire dataset may be treated as an in-memory dataset that resides entirely in DRAM. SCM may include non-volatile media such as, for example, NAND flash. Such NAND flash may be accessed utilizing NVMe that can use the PCIe bus as its transport, providing for relatively low access latencies compared to older protocols. In fact, the network protocols used for SSDs in all-flash arrays can include NVMe using Ethernet (ROCE, NVME TCP), Fibre Channel (NVMe FC), InfiniBand (iWARP), and others that make it possible to treat fast, non-volatile memory as an extension of DRAM. In view of the fact that DRAM is often byte-addressable and fast, non-volatile memory such as NAND flash is block-addressable, a controller software/hardware stack may be needed to convert the block data to the bytes that are stored in the media. Examples of media and software that may be used as SCM can include, for example, 3D XPoint, Intel Memory Drive Technology, Samsung's Z-SSD, and others.

[0134] The storage resources **308** depicted in FIG. **3**B may also include racetrack memory (also referred to as domain-wall memory). Such racetrack memory may be embodied as a form of non-volatile, solid-state memory that relies on the intrinsic strength and orientation of the magnetic field created by an electron as it spins in addition to its electronic charge, in solid-state devices. Through the use of spin-coherent electric current to move magnetic domains along a nanoscopic permalloy wire, the domains may pass by magnetic read/write heads positioned near the wire as current is passed through the wire, which alter the domains to record patterns of bits. In order to create a racetrack memory device, many such wires and read/write elements may be packaged together.

[0135] The example storage system **306** depicted in FIG. **3**B may implement a variety of storage architectures. For example, storage systems in accordance with some embodiments of the present disclosure may utilize block storage where data is stored in blocks, and each block essentially acts as an individual hard drive. Storage systems in accordance with some embodiments of the present disclosure may utilize object storage, where data is managed as objects. Each object may include the data itself, a variable amount of metadata, and a globally unique identifier, where object storage can be implemented at multiple levels (e.g., device level, system level, interface level). Storage systems in accordance with some embodiments of the present disclosure utilize file storage in which data is stored in a hierarchical structure. Such data may be saved in files and folders, and presented to both the system storing it and the system retrieving it in the same format.

[0136] The example storage system **306** depicted in FIG. **3**B may be embodied as a storage system in which additional storage resources can be added through the use of a scale-up model, additional storage resources can be added through the use of a scale-out model, or through some combination thereof. In a scale-up model, additional storage may be added by adding additional storage devices. In a scale-out model, however, additional storage nodes may be added to a cluster of storage nodes, where such storage nodes can include additional processing resources, additional networking resources, and so on.

[0137] The example storage system **306** depicted in FIG. **3**B may leverage the storage resources described above in a variety of different ways. For example, some portion of the storage resources may be utilized to serve as a write cache where data is initially written to storage resources with relatively fast write latencies, relatively high write bandwidth, or similar characteristics. In such an example, data

that is written to the storage resources that serve as a write cache may later be written to other storage resources that may be characterized by slower write latencies, lower write bandwidth, or similar characteristics than the storage resources that are utilized to serve as a write cache. In a similar manner, storage resources within the storage system may be utilized as a read cache, where the read cache is populated in accordance with a set of predetermined rules or heuristics. In other embodiments, tiering may be achieved within the storage systems by placing data within the storage system in accordance with one or more policies such that, for example, data that is accessed frequently is stored in faster storage tiers while data that is accessed infrequently is stored in slower storage tiers.

[0138] The storage system **306** depicted in FIG. **3**B also includes communications resources **310** that may be useful in facilitating data communications between components within the storage system **306**, as well as data communications between the storage system **306** and computing devices that are outside of the storage system **306**, including embodiments where those resources are separated by a relatively vast expanse. The communications resources **310** may be configured to utilize a variety of different protocols and data communication fabrics to facilitate data communications between components within the storage systems as well as computing devices that are outside of the storage system. For example, the communications resources **310** can include fibre channel ('FC') technologies such as FC fabrics and FC protocols that can transport SCSI commands over FC network, FC over ethernet ('FCoE') technologies through which FC frames are encapsulated and transmitted over Ethernet networks, InfiniBand ('IB') technologies in which a switched fabric topology is utilized to facilitate transmissions between channel adapters, NVM Express ('NVMe') technologies and NVMe over fabrics ('NVMeoF') technologies through which non-volatile storage media attached via a PCI express ('PCIe') bus may be accessed, and others. In fact, the storage systems described above may, directly or indirectly, make use of neutrino communication technologies and devices through which information (including binary information) is transmitted using a beam of neutrinos.

[0139] The communications resources **310** can also include mechanisms for accessing storage resources **308** within the storage system **306** utilizing serial attached SCSI ('SAS'), serial ATA ('SATA') bus interfaces for connecting storage resources **308** within the storage system **306** to host bus adapters within the storage system **306**, internet small computer systems interface ('iSCSI') technologies to provide block-level access to storage resources **308** within the storage system **306**, and other communications resources that that may be useful in facilitating data communications between components within the storage system **306**, as well as data communications between the storage system **306** and computing devices that are outside of the storage system **306**.

[0140] The storage system **306** depicted in FIG. **3**B also includes processing resources **312** that may be useful in useful in executing computer program instructions and performing other computational tasks within the storage system **306**. The processing resources **312** may include one or more ASICs that are customized for some particular purpose as well as one or more CPUs. The processing resources **312** may also include one or more DSPs, one or more FPGAs, one or more systems on a chip ('SoCs'), or other form of processing resources **312**. The storage system **306** may utilize the storage resources **312** to perform a variety of tasks including, but not limited to, supporting the execution of software resources **314** that will be described in greater detail below.

[0141] The storage system **306** depicted in FIG. **3**B also includes software resources **314** that, when executed by processing resources **312** within the storage system **306**, may perform a vast array of tasks. The software resources **314** may include, for example, one or more modules of computer program instructions that when executed by processing resources **312** within the storage system **306** are useful in carrying out various data protection techniques to preserve the integrity of data that is stored within the storage systems. Readers will appreciate that such data protection techniques may be carried out, for example, by system software executing on computer hardware within the storage system, by a cloud services provider, or in other ways. Such data protection techniques can include, for example, data archiving techniques that cause data that is no longer actively used to be moved to a separate storage device or separate storage system for long-term retention, data backup techniques through which data stored in the storage system may be copied and stored in a distinct location to avoid data loss in the event of equipment failure or some other form of catastrophe with the storage system, data replication techniques through which data stored in the storage system is replicated to another storage system such that the data may be accessible via multiple storage systems, data snapshotting techniques through which the state of data within the storage system is captured at various points in time, data and database cloning techniques through which duplicate copies of data and databases may be created, and other data protection techniques.

[0142] The software resources **314** may also include software that is useful in implementing software-defined storage ('SDS'). In such an example, the software resources **314** may include one or more modules of computer program instructions that, when executed, are useful in policy-based provisioning and management of data storage that is independent of the underlying hardware. Such software resources **314** may be useful in implementing storage virtualization to separate the storage hardware from the software that manages the storage hardware.

[0143] The software resources **314** may also include software that is useful in facilitating and optimizing I/O operations that are directed to the storage resources **308** in the storage system **306**. For example, the software resources **314** may include software modules that perform carry out various data reduction techniques such as, for example, data compression, data deduplication, and others. The software resources **314** may include software modules that intelligently group together I/O operations to facilitate better usage of the underlying storage resource **308**, software modules that perform data migration operations to migrate from within a storage system, as well as software modules that perform other functions. Such software resources **314** may be embodied as one or more software containers or in many other ways.

[0144] For further explanation, FIG. **4** sets forth an example of a cloud-based storage system **318** in accordance with some embodiments of the present disclosure. In the example depicted in FIG. **4**, the cloud-based storage system **318** is created entirely in a cloud computing environment

316 such as, for example, Amazon Web Services ('AWS'), Microsoft Azure, Google Cloud Platform, IBM Cloud, Oracle Cloud, and others. The cloud-based storage system **318** may be used to provide services similar to the services that may be provided by the storage systems described above. For example, the cloud-based storage system **318** may be used to provide block storage services to users of the cloud-based storage system **318**, the cloud-based storage system **318** may be used to provide storage services to users of the cloud-based storage system **318** through the use of solid-state storage, and so on.

[0145] The cloud-based storage system **318** depicted in FIG. **4** includes two cloud computing instances **320**, **322** that each are used to support the execution of a storage controller application **324**, **326**. The cloud computing instances **320**, **322** may be embodied, for example, as instances of cloud computing resources (e.g., virtual machines) that may be provided by the cloud computing environment **316** to support the execution of software applications such as the storage controller application **324**, **326**. In one embodiment, the cloud computing instances **320**, **322** may be embodied as Amazon Elastic Compute Cloud ('EC2') instances. In such an example, an Amazon Machine Image ('AMI') that includes the storage controller application **324**, **326** may be booted to create and configure a virtual machine that may execute the storage controller application **324**, **326**.

[0146] In the example method depicted in FIG. **4**, the storage controller application **324**, **326** may be embodied as a module of computer program instructions that, when executed, carries out various storage tasks. For example, the storage controller application **324**, **326** may be embodied as a module of computer program instructions that, when executed, carries out the same tasks as the controllers **110**A, **110**B in FIG. **1**A described above such as writing data received from the users of the cloud-based storage system **318** to the cloud-based storage system **318**, erasing data from the cloud-based storage system **318**, retrieving data from the cloud-based storage system **318** and providing such data to users of the cloud-based storage system **318**, monitoring and reporting of disk utilization and performance, performing redundancy operations, such as RAID or RAID-like data redundancy operations, compressing data, encrypting data, deduplicating data, and so forth. Readers will appreciate that because there are two cloud computing instances **320**, **322** that each include the storage controller application **324**, **326**, in some embodiments one cloud computing instance **320** may operate as the primary controller as described above while the other cloud computing instance **322** may operate as the secondary controller as described above. Readers will appreciate that the storage controller application **324**, **326** depicted in FIG. **4** may include identical source code that is executed within different cloud computing instances **320**, **322**.

[0147] Consider an example in which the cloud computing environment **316** is embodied as AWS and the cloud computing instances are embodied as EC2 instances. In such an example, the cloud computing instance **320** that operates as the primary controller may be deployed on one of the instance types that has a relatively large amount of memory and processing power while the cloud computing instance **322** that operates as the secondary controller may be deployed on one of the instance types that has a relatively small amount of memory and processing power. In such an example, upon the occurrence of a failover event where the roles of primary and secondary are switched, a double failover may actually be carried out such that: 1) a first failover event where the cloud computing instance **322** that formerly operated as the secondary controller begins to operate as the primary controller, and 2) a third cloud computing instance (not shown) that is of an instance type that has a relatively large amount of memory and processing power is spun up with a copy of the storage controller application, where the third cloud computing instance begins operating as the primary controller while the cloud computing instance **322** that originally operated as the secondary controller begins operating as the secondary controller again. In such an example, the cloud computing instance **320** that formerly operated as the primary controller may be terminated. Readers will appreciate that in alternative embodiments, the cloud computing instance **320** that is operating as the secondary controller after the failover event may continue to operate as the secondary controller and the cloud computing instance **322** that operated as the primary controller after the occurrence of the failover event may be terminated once the primary role has been assumed by the third cloud computing instance (not shown).

[0148] Readers will appreciate that while the embodiments described above relate to embodiments where one cloud computing instance **320** operates as the primary controller and the second cloud computing instance **322** operates as the secondary controller, other embodiments are within the scope of the present disclosure. For example, each cloud computing instance **320**, **322** may operate as a primary controller for some portion of the address space supported by the cloud-based storage system **318**, each cloud computing instance **320**, **322** may operate as a primary controller where the servicing of I/O operations directed to the cloud-based storage system **318** are divided in some other way, and so on. In fact, in other embodiments where costs savings may be prioritized over performance demands, only a single cloud computing instance may exist that contains the storage controller application.

[0149] The cloud-based storage system **318** depicted in FIG. **4** includes cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338**. The cloud computing instances **340***a*, **340***b*, **340***n* depicted in FIG. **4** may be embodied, for example, as instances of cloud computing resources that may be provided by the cloud computing environment **316** to support the execution of software applications. The cloud computing instances **340***a*, **340***b*, **340***n* of FIG. **4** may differ from the cloud computing instances **320**, **322** described above as the cloud computing instances **340***a*, **340***b*, **340***n* of FIG. **4** have local storage **330**, **334**, **338** resources whereas the cloud computing instances **320**, **322** that support the execution of the storage controller application **324**, **326** need not have local storage resources. The cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338** may be embodied, for example, as EC2 M5 instances that include one or more SSDs, as EC2 R5 instances that include one or more SSDs, as EC2 I3 instances that include one or more SSDs, and so on. In some embodiments, the local storage **330**, **334**, **338** must be embodied as solid-state storage (e.g., SSDs) rather than storage that makes use of hard disk drives.

[0150] In the example depicted in FIG. **4**, each of the cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338** can include a software daemon **328**, **332**, **336** that, when executed by a cloud computing instance **340***a*,

340*b*, **340***n* can present itself to the storage controller applications **324**, **326** as if the cloud computing instance **340***a*, **340***b*, **340***n* were a physical storage device (e.g., one or more SSDs). In such an example, the software daemon **328**, **332**, **336** may include computer program instructions similar to those that would normally be contained on a storage device such that the storage controller applications **324**, **326** can send and receive the same commands that a storage controller would send to storage devices. In such a way, the storage controller applications **324**, **326** may include code that is identical to (or substantially identical to) the code that would be executed by the controllers in the storage systems described above. In these and similar embodiments, communications between the storage controller applications **324**, **326** and the cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338** may utilize iSCSI, NVMe over TCP, messaging, a custom protocol, or in some other mechanism.

[0151] In the example depicted in FIG. **4**, each of the cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338** may also be coupled to block-storage **342**, **344**, **346** that is offered by the cloud computing environment **316**. The block-storage **342**, **344**, **346** that is offered by the cloud computing environment **316** may be embodied, for example, as Amazon Elastic Block Store ('EBS') volumes. For example, a first EBS volume may be coupled to a first cloud computing instance **340***a*, a second EBS volume may be coupled to a second cloud computing instance **340***b*, and a third EBS volume may be coupled to a third cloud computing instance **340***n*. In such an example, the block-storage **342**, **344**, **346** that is offered by the cloud computing environment **316** may be utilized in a manner that is similar to how the NVRAM devices described above are utilized, as the software daemon **328**, **332**, **336** (or some other module) that is executing within a particular cloud comping instance **340***a*, **340***b*, **340***n* may, upon receiving a request to write data, initiate a write of the data to its attached EBS volume as well as a write of the data to its local storage **330**, **334**, **338** resources. In some alternative embodiments, data may only be written to the local storage **330**, **334**, **338** resources within a particular cloud comping instance **340***a*, **340***b*, **340***n*. In an alternative embodiment, rather than using the block-storage **342**, **344**, **346** that is offered by the cloud computing environment **316** as NVRAM, actual RAM on each of the cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338** may be used as NVRAM, thereby decreasing network utilization costs that would be associated with using an EBS volume as the NVRAM.

[0152] In the example depicted in FIG. **4**, the cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338** may be utilized, by cloud computing instances **320**, **322** that support the execution of the storage controller application **324**, **326** to service I/O operations that are directed to the cloud-based storage system **318**. Consider an example in which a first cloud computing instance **320** that is executing the storage controller application **324** is operating as the primary controller. In such an example, the first cloud computing instance **320** that is executing the storage controller application **324** may receive (directly or indirectly via the secondary controller) requests to write data to the cloud-based storage system **318** from users of the cloud-based storage system **318**. In such an example, the first cloud computing instance **320** that is executing the storage controller application **324** may perform various tasks such as, for example, deduplicating the data contained in the request, compressing the data contained in the request, determining where to the write the data contained in the request, and so on, before ultimately sending a request to write a deduplicated, encrypted, or otherwise possibly updated version of the data to one or more of the cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338**. Either cloud computing instance **320**, **322**, in some embodiments, may receive a request to read data from the cloud-based storage system **318** and may ultimately send a request to read data to one or more of the cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338**.

[0153] Readers will appreciate that when a request to write data is received by a particular cloud computing instance **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338**, the software daemon **328**, **332**, **336** or some other module of computer program instructions that is executing on the particular cloud computing instance **340***a*, **340***b*, **340***n* may be configured to not only write the data to its own local storage **330**, **334**, **338** resources and any appropriate block-storage **342**, **344**, **346** that are offered by the cloud computing environment **316**, but the software daemon **328**, **332**, **336** or some other module of computer program instructions that is executing on the particular cloud computing instance **340***a*, **340***b*, **340***n* may also be configured to write the data to cloud-based object storage **348** that is attached to the particular cloud computing instance **340***a*, **340***b*, **340***n*. The cloud-based object storage **348** that is attached to the particular cloud computing instance **340***a*, **340***b*, **340***n* may be embodied, for example, as Amazon Simple Storage Service ('S3') storage that is accessible by the particular cloud computing instance **340***a*, **340***b*, **340***n*. In other embodiments, the cloud computing instances **320**, **322** that each include the storage controller application **324**, **326** may initiate the storage of the data in the local storage **330**, **334**, **338** of the cloud computing instances **340***a*, **340***b*, **340***n* and the cloud-based object storage **348**.

[0154] Readers will appreciate that, as described above, the cloud-based storage system **318** may be used to provide block storage services to users of the cloud-based storage system **318**. While the local storage **330**, **334**, **338** resources and the block-storage **342**, **344**, **346** resources that are utilized by the cloud computing instances **340***a*, **340***b*, **340***n* may support block-level access, the cloud-based object storage **348** that is attached to the particular cloud computing instance **340***a*, **340***b*, **340***n* supports only object-based access. In order to address this, the software daemon **328**, **332**, **336** or some other module of computer program instructions that is executing on the particular cloud computing instance **340***a*, **340***b*, **340***n* may be configured to take blocks of data, package those blocks into objects, and write the objects to the cloud-based object storage **348** that is attached to the particular cloud computing instance **340***a*, **340***b*, **340***n*.

[0155] Consider an example in which data is written to the local storage **330**, **334**, **338** resources and the block-storage **342**, **344**, **346** resources that are utilized by the cloud computing instances **340***a*, **340***b*, **340***n* in 1 MB blocks. In such an example, assume that a user of the cloud-based storage system **318** issues a request to write data that, after being compressed and deduplicated by the storage controller application **324**, **326** results in the need to write 5 MB of data. In such an example, writing the data to the local storage **330**, **334**, **338** resources and the block-storage **342**, **344**, **346**

resources that are utilized by the cloud computing instances **340***a*, **340***b*, **340***n* is relatively straightforward as 5 blocks that are 1 MB in size are written to the local storage **330**, **334**, **338** resources and the block-storage **342**, **344**, **346** resources that are utilized by the cloud computing instances **340***a*, **340***b*, **340***n*. In such an example, the software daemon **328**, **332**, **336** or some other module of computer program instructions that is executing on the particular cloud computing instance **340***a*, **340***b*, **340***n* may be configured to: 1) create a first object that includes the first 1 MB of data and write the first object to the cloud-based object storage **348**, 2) create a second object that includes the second 1 MB of data and write the second object to the cloud-based object storage **348**, 3) create a third object that includes the third 1 MB of data and write the third object to the cloud-based object storage **348**, and so on. As such, in some embodiments, each object that is written to the cloud-based object storage **348** may be identical (or nearly identical) in size. Readers will appreciate that in such an example, metadata that is associated with the data itself may be included in each object (e.g., the first 1 MB of the object is data and the remaining portion is metadata associated with the data).

[0156] Readers will appreciate that the cloud-based object storage **348** may be incorporated into the cloud-based storage system **318** to increase the durability of the cloud-based storage system **318**. Continuing with the example described above where the cloud computing instances **340***a*, **340***b*, **340***n* are EC2 instances, readers will understand that EC2 instances are only guaranteed to have a monthly uptime of 99.9% and data stored in the local instance store only persists during the lifetime of the EC2 instance. As such, relying on the cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338** as the only source of persistent data storage in the cloud-based storage system **318** may result in a relatively unreliable storage system. Likewise, EBS volumes are designed for 99.999% availability. As such, even relying on EBS as the persistent data store in the cloud-based storage system **318** may result in a storage system that is not sufficiently durable. Amazon S3, however, is designed to provide 99.999999999% durability, meaning that a cloud-based storage system **318** that can incorporate S3 into its pool of storage is substantially more durable than various other options.

[0157] Readers will appreciate that while a cloud-based storage system **318** that can incorporate S3 into its pool of storage is substantially more durable than various other options, utilizing S3 as the primary pool of storage may result in storage system that has relatively slow response times and relatively long I/O latencies. As such, the cloud-based storage system **318** depicted in FIG. **4** not only stores data in S3 but the cloud-based storage system **318** also stores data in local storage **330**, **334**, **338** resources and block-storage **342**, **344**, **346** resources that are utilized by the cloud computing instances **340***a*, **340***b*, **340***n*, such that read operations can be serviced from local storage **330**, **334**, **338** resources and the block-storage **342**, **344**, **346** resources that are utilized by the cloud computing instances **340***a*, **340***b*, **340***n*, thereby reducing read latency when users of the cloud-based storage system **318** attempt to read data from the cloud-based storage system **318**.

[0158] In some embodiments, all data that is stored by the cloud-based storage system **318** may be stored in both: 1) the cloud-based object storage **348**, and 2) at least one of the local storage **330**, **334**, **338** resources or block-storage **342**, **344**, **346** resources that are utilized by the cloud computing instances **340***a*, **340***b*, **340***n*. In such embodiments, the local storage **330**, **334**, **338** resources and block-storage **342**, **344**, **346** resources that are utilized by the cloud computing instances **340***a*, **340***b*, **340***n* may effectively operate as cache that generally includes all data that is also stored in S3, such that all reads of data may be serviced by the cloud computing instances **340***a*, **340***b*, **340***n* without requiring the cloud computing instances **340***a*, **340***b*, **340***n* to access the cloud-based object storage **348**. Readers will appreciate that in other embodiments, however, all data that is stored by the cloud-based storage system **318** may be stored in the cloud-based object storage **348**, but less than all data that is stored by the cloud-based storage system **318** may be stored in at least one of the local storage **330**, **334**, **338** resources or block-storage **342**, **344**, **346** resources that are utilized by the cloud computing instances **340***a*, **340***b*, **340***n*. In such an example, various policies may be utilized to determine which subset of the data that is stored by the cloud-based storage system **318** should reside in both: 1) the cloud-based object storage **348**, and 2) at least one of the local storage **330**, **334**, **338** resources or block-storage **342**, **344**, **346** resources that are utilized by the cloud computing instances **340***a*, **340***b*, **340***n*.

[0159] As described above, when the cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338** are embodied as EC2 instances, the cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338** are only guaranteed to have a monthly uptime of 99.9% and data stored in the local instance store only persists during the lifetime of each cloud computing instance **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338**. As such, one or more modules of computer program instructions that are executing within the cloud-based storage system **318** (e.g., a monitoring module that is executing on its own EC2 instance) may be designed to handle the failure of one or more of the cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338**. In such an example, the monitoring module may handle the failure of one or more of the cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338** by creating one or more new cloud computing instances with local storage, retrieving data that was stored on the failed cloud computing instances **340***a*, **340***b*, **340***n* from the cloud-based object storage **348**, and storing the data retrieved from the cloud-based object storage **348** in local storage on the newly created cloud computing instances. Readers will appreciate that many variants of this process may be implemented.

[0160] Consider an example in which all cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338** failed. In such an example, the monitoring module may create new cloud computing instances with local storage, where high-bandwidth instances types are selected that allow for the maximum data transfer rates between the newly created high-bandwidth cloud computing instances with local storage and the cloud-based object storage **348**. Readers will appreciate that instances types are selected that allow for the maximum data transfer rates between the new cloud computing instances and the cloud-based object storage **348** such that the new high-bandwidth cloud computing instances can be rehydrated with data from the cloud-based object storage **348** as quickly as possible. Once the new high-bandwidth cloud computing instances are rehydrated with data from the cloud-based object storage **348**, less

expensive lower-bandwidth cloud computing instances may be created, data may be migrated to the less expensive lower-bandwidth cloud computing instances, and the high-bandwidth cloud computing instances may be terminated.

[0161] Readers will appreciate that in some embodiments, the number of new cloud computing instances that are created may substantially exceed the number of cloud computing instances that are needed to locally store all of the data stored by the cloud-based storage system **318**. The number of new cloud computing instances that are created may substantially exceed the number of cloud computing instances that are needed to locally store all of the data stored by the cloud-based storage system **318** in order to more rapidly pull data from the cloud-based object storage **348** and into the new cloud computing instances, as each new cloud computing instance can (in parallel) retrieve some portion of the data stored by the cloud-based storage system **318**. In such embodiments, once the data stored by the cloud-based storage system **318** has been pulled into the newly created cloud computing instances, the data may be consolidated within a subset of the newly created cloud computing instances and those newly created cloud computing instances that are excessive may be terminated.

[0162] Consider an example in which 1000 cloud computing instances are needed in order to locally store all valid data that users of the cloud-based storage system **318** have written to the cloud-based storage system **318**. In such an example, assume that all 1,000 cloud computing instances fail. In such an example, the monitoring module may cause 100,000 cloud computing instances to be created, where each cloud computing instance is responsible for retrieving, from the cloud-based object storage **348**, distinct 1/100,000th chunks of the valid data that users of the cloud-based storage system **318** have written to the cloud-based storage system **318** and locally storing the distinct chunk of the dataset that it retrieved. In such an example, because each of the 100,000 cloud computing instances can retrieve data from the cloud-based object storage **348** in parallel, the caching layer may be restored 100 times faster as compared to an embodiment where the monitoring module only create 1000 replacement cloud computing instances. In such an example, over time the data that is stored locally in the 100,000 could be consolidated into 1,000 cloud computing instances and the remaining 99,000 cloud computing instances could be terminated.

[0163] Readers will appreciate that various performance aspects of the cloud-based storage system **318** may be monitored (e.g., by a monitoring module that is executing in an EC2 instance) such that the cloud-based storage system **318** can be scaled-up or scaled-out as needed. Consider an example in which the monitoring module monitors the performance of the could-based storage system **318** via communications with one or more of the cloud computing instances **320**, **322** that each are used to support the execution of a storage controller application **324**, **326**, via monitoring communications between cloud computing instances **320**, **322**, **340***a*, **340***b*, **340***n*, via monitoring communications between cloud computing instances **320**, **322**, **340***a*, **340***b*, **340***n* and the cloud-based object storage **348**, or in some other way. In such an example, assume that the monitoring module determines that the cloud computing instances **320**, **322** that are used to support the execution of a storage controller application **324**, **326** are undersized and not sufficiently servicing the I/O requests that are issued by users of the cloud-based storage system **318**. In such an example, the monitoring module may create a new, more powerful cloud computing instance (e.g., a cloud computing instance of a type that includes more processing power, more memory, etc. . . .) that includes the storage controller application such that the new, more powerful cloud computing instance can begin operating as the primary controller. Likewise, if the monitoring module determines that the cloud computing instances **320**, **322** that are used to support the execution of a storage controller application **324**, **326** are oversized and that cost savings could be gained by switching to a smaller, less powerful cloud computing instance, the monitoring module may create a new, less powerful (and less expensive) cloud computing instance that includes the storage controller application such that the new, less powerful cloud computing instance can begin operating as the primary controller.

[0164] Consider, as an additional example of dynamically sizing the cloud-based storage system **318**, an example in which the monitoring module determines that the utilization of the local storage that is collectively provided by the cloud computing instances **340***a*, **340***b*, **340***n* has reached a predetermined utilization threshold (e.g., 95%). In such an example, the monitoring module may create additional cloud computing instances with local storage to expand the pool of local storage that is offered by the cloud computing instances. Alternatively, the monitoring module may create one or more new cloud computing instances that have larger amounts of local storage than the already existing cloud computing instances **340***a*, **340***b*, **340***n*, such that data stored in an already existing cloud computing instance **340***a*, **340***b*, **340***n* can be migrated to the one or more new cloud computing instances and the already existing cloud computing instance **340***a*, **340***b*, **340***n* can be terminated, thereby expanding the pool of local storage that is offered by the cloud computing instances. Likewise, if the pool of local storage that is offered by the cloud computing instances is unnecessarily large, data can be consolidated and some cloud computing instances can be terminated.

[0165] Readers will appreciate that the cloud-based storage system **318** may be sized up and down automatically by a monitoring module applying a predetermined set of rules that may be relatively simple of relatively complicated. In fact, the monitoring module may not only take into account the current state of the cloud-based storage system **318**, but the monitoring module may also apply predictive policies that are based on, for example, observed behavior (e.g., every night from 10 PM until 6 AM usage of the storage system is relatively light), predetermined fingerprints (e.g., every time a virtual desktop infrastructure adds 100 virtual desktops, the number of TOPS directed to the storage system increase by X), and so on. In such an example, the dynamic scaling of the cloud-based storage system **318** may be based on current performance metrics, predicted workloads, and many other factors, including combinations thereof.

[0166] Readers will further appreciate that because the cloud-based storage system **318** may be dynamically scaled, the cloud-based storage system **318** may even operate in a way that is more dynamic. Consider the example of garbage collection. In a traditional storage system, the amount of storage is fixed. As such, at some point the storage system may be forced to perform garbage collection as the amount of available storage has become so constrained that the storage system is on the verge of running out of storage. In

contrast, the cloud-based storage system **318** described here can always 'add' additional storage (e.g., by adding more cloud computing instances with local storage). Because the cloud-based storage system **318** described here can always 'add' additional storage, the cloud-based storage system **318** can make more intelligent decisions regarding when to perform garbage collection. For example, the cloud-based storage system **318** may implement a policy that garbage collection only be performed when the number of TOPS being serviced by the cloud-based storage system **318** falls below a certain level. In some embodiments, other system-level functions (e.g., deduplication, compression) may also be turned off and on in response to system load, given that the size of the cloud-based storage system **318** is not constrained in the same way that traditional storage systems are constrained.

[0167] Readers will appreciate that embodiments of the present disclosure resolve an issue with block-storage services offered by some cloud computing environments as some cloud computing environments only allow for one cloud computing instance to connect to a block-storage volume at a single time. For example, in Amazon AWS, only a single EC2 instance may be connected to an EBS volume. Through the use of EC2 instances with local storage, embodiments of the present disclosure can offer multi-connect capabilities where multiple EC2 instances can connect to another EC2 instance with local storage ('a drive instance'). In such embodiments, the drive instances may include software executing within the drive instance that allows the drive instance to support I/O directed to a particular volume from each connected EC2 instance. As such, some embodiments of the present disclosure may be embodied as multi-connect block storage services that may not include all of the components depicted in FIG. **4**.

[0168] In some embodiments, especially in embodiments where the cloud-based object storage **348** resources are embodied as Amazon S3, the cloud-based storage system **318** may include one or more modules (e.g., a module of computer program instructions executing on an EC2 instance) that are configured to ensure that when the local storage of a particular cloud computing instance is rehydrated with data from S3, the appropriate data is actually in S3. This issue arises largely because S3 implements an eventual consistency model where, when overwriting an existing object, reads of the object will eventually (but not necessarily immediately) become consistent and will eventually (but not necessarily immediately) return the overwritten version of the object. To address this issue, in some embodiments of the present disclosure, objects in S3 are never overwritten. Instead, a traditional 'overwrite' would result in the creation of the new object (that includes the updated version of the data) and the eventual deletion of the old object (that includes the previous version of the data).

[0169] In some embodiments of the present disclosure, as part of an attempt to never (or almost never) overwrite an object, when data is written to S3 the resultant object may be tagged with a sequence number. In some embodiments, these sequence numbers may be persisted elsewhere (e.g., in a database) such that at any point in time, the sequence number associated with the most up-to-date version of some piece of data can be known. In such a way, a determination can be made as to whether S3 has the most recent version of some piece of data by merely reading the sequence number associated with an object—and without actually reading the data from S3. The ability to make this determination may be particularly important when a cloud computing instance with local storage crashes, as it would be undesirable to rehydrate the local storage of a replacement cloud computing instance with out-of-date data. In fact, because the cloud-based storage system **318** does not need to access the data to verify its validity, the data can stay encrypted and access charges can be avoided.

[0170] The storage systems described above may carry out intelligent data backup techniques through which data stored in the storage system may be copied and stored in a distinct location to avoid data loss in the event of equipment failure or some other form of catastrophe. For example, the storage systems described above may be configured to examine each backup to avoid restoring the storage system to an undesirable state. Consider an example in which malware infects the storage system. In such an example, the storage system may include software resources **314** that can scan each backup to identify backups that were captured before the malware infected the storage system and those backups that were captured after the malware infected the storage system. In such an example, the storage system may restore itself from a backup that does not include the malware—or at least not restore the portions of a backup that contained the malware. In such an example, the storage system may include software resources **314** that can scan each backup to identify the presences of malware (or a virus, or some other undesirable), for example, by identifying write operations that were serviced by the storage system and originated from a network subnet that is suspected to have delivered the malware, by identifying write operations that were serviced by the storage system and originated from a user that is suspected to have delivered the malware, by identifying write operations that were serviced by the storage system and examining the content of the write operation against fingerprints of the malware, and in many other ways.

[0171] Readers will further appreciate that the backups (often in the form of one or more snapshots) may also be utilized to perform rapid recovery of the storage system. Consider an example in which the storage system is infected with ransomware that locks users out of the storage system. In such an example, software resources **314** within the storage system may be configured to detect the presence of ransomware and may be further configured to restore the storage system to a point-in-time, using the retained backups, prior to the point-in-time at which the ransomware infected the storage system. In such an example, the presence of ransomware may be explicitly detected through the use of software tools utilized by the system, through the use of a key (e.g., a USB drive) that is inserted into the storage system, or in a similar way. Likewise, the presence of ransomware may be inferred in response to system activity meeting a predetermined fingerprint such as, for example, no reads or writes coming into the system for a predetermined period of time.

[0172] Readers will appreciate that the various components described above may be grouped into one or more optimized computing packages as converged infrastructures. Such converged infrastructures may include pools of computers, storage and networking resources that can be shared by multiple applications and managed in a collective manner using policy-driven processes. Such converged infrastructures may be implemented with a converged infrastructure reference architecture, with standalone appliances, with a

software driven hyper-converged approach (e.g., hyper-converged infrastructures), or in other ways.

[0173] Readers will appreciate that the storage systems described above may be useful for supporting various types of software applications. For example, the storage system **306** may be useful in supporting artificial intelligence ('AI') applications, database applications, DevOps projects, electronic design automation tools, event-driven software applications, high performance computing applications, simulation applications, high-speed data capture and analysis applications, machine learning applications, media production applications, media serving applications, picture archiving and communication systems ('PACS') applications, software development applications, virtual reality applications, augmented reality applications, and many other types of applications by providing storage resources to such applications.

[0174] The storage systems described above may operate to support a wide variety of applications. In view of the fact that the storage systems include compute resources, storage resources, and a wide variety of other resources, the storage systems may be well suited to support applications that are resource intensive such as, for example, AI applications. AI applications may be deployed in a variety of fields, including: predictive maintenance in manufacturing and related fields, healthcare applications such as patient data & risk analytics, retail and marketing deployments (e.g., search advertising, social media advertising), supply chains solutions, fintech solutions such as business analytics & reporting tools, operational deployments such as real-time analytics tools, application performance management tools, IT infrastructure management tools, and many others.

[0175] Such AI applications may enable devices to perceive their environment and take actions that maximize their chance of success at some goal. Examples of such AI applications can include IBM Watson, Microsoft Oxford, Google DeepMind, Baidu Minwa, and others. The storage systems described above may also be well suited to support other types of applications that are resource intensive such as, for example, machine learning applications. Machine learning applications may perform various types of data analysis to automate analytical model building. Using algorithms that iteratively learn from data, machine learning applications can enable computers to learn without being explicitly programmed. One particular area of machine learning is referred to as reinforcement learning, which involves taking suitable actions to maximize reward in a particular situation. Reinforcement learning may be employed to find the best possible behavior or path that a particular software application or machine should take in a specific situation. Reinforcement learning differs from other areas of machine learning (e.g., supervised learning, unsupervised learning) in that correct input/output pairs need not be presented for reinforcement learning and sub-optimal actions need not be explicitly corrected.

[0176] In addition to the resources already described, the storage systems described above may also include graphics processing units ('GPUs'), occasionally referred to as visual processing unit ('VPUs'). Such GPUs may be embodied as specialized electronic circuits that rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device. Such GPUs may be included within any of the computing devices that are part of the storage systems described above, including as one of many individually scalable components of a storage system, where other examples of individually scalable components of such storage system can include storage components, memory components, compute components (e.g., CPUs, FPGAs, ASICs), networking components, software components, and others. In addition to GPUs, the storage systems described above may also include neural network processors ('NNPs') for use in various aspects of neural network processing. Such NNPs may be used in place of (or in addition to) GPUs and may be also be independently scalable.

[0177] As described above, the storage systems described herein may be configured to support artificial intelligence applications, machine learning applications, big data analytics applications, and many other types of applications. The rapid growth in these sort of applications is being driven by three technologies: deep learning (DL), GPU processors, and Big Data. Deep learning is a computing model that makes use of massively parallel neural networks inspired by the human brain. Instead of experts handcrafting software, a deep learning model writes its own software by learning from lots of examples. Such GPUs may include thousands of cores that are well-suited to run algorithms that loosely represent the parallel nature of the human brain.

[0178] Advances in deep neural networks, including the development of multi-layer neural networks, have ignited a new wave of algorithms and tools for data scientists to tap into their data with artificial intelligence (AI). With improved algorithms, larger data sets, and various frameworks (including open-source software libraries for machine learning across a range of tasks), data scientists are tackling new use cases like autonomous driving vehicles, natural language processing and understanding, computer vision, machine reasoning, strong AI, and many others. Applications of such techniques may include: machine and vehicular object detection, identification and avoidance; visual recognition, classification and tagging; algorithmic financial trading strategy performance management; simultaneous localization and mapping; predictive maintenance of high-value machinery; prevention against cyber security threats, expertise automation; image recognition and classification; question answering; robotics; text analytics (extraction, classification) and text generation and translation; and many others. Applications of AI techniques has materialized in a wide array of products include, for example, Amazon Echo's speech recognition technology that allows users to talk to their machines, Google Translate™ which allows for machine-based language translation, Spotify's Discover Weekly that provides recommendations on new songs and artists that a user may like based on the user's usage and traffic analysis, Quill's text generation offering that takes structured data and turns it into narrative stories, Chatbots that provide real-time, contextually specific answers to questions in a dialog format, and many others.

[0179] Data is the heart of modern AI and deep learning algorithms. Before training can begin, one problem that must be addressed revolves around collecting the labeled data that is crucial for training an accurate AI model. A full scale AI deployment may be required to continuously collect, clean, transform, label, and store large amounts of data. Adding additional high quality data points directly translates to more accurate models and better insights. Data samples may undergo a series of processing steps including, but not limited to: 1) ingesting the data from an external source into

the training system and storing the data in raw form, 2) cleaning and transforming the data in a format convenient for training, including linking data samples to the appropriate label, 3) exploring parameters and models, quickly testing with a smaller dataset, and iterating to converge on the most promising models to push into the production cluster, 4) executing training phases to select random batches of input data, including both new and older samples, and feeding those into production GPU servers for computation to update model parameters, and 5) evaluating including using a holdback portion of the data not used in training in order to evaluate model accuracy on the holdout data. This lifecycle may apply for any type of parallelized machine learning, not just neural networks or deep learning. For example, standard machine learning frameworks may rely on CPUs instead of GPUs but the data ingest and training workflows may be the same. Readers will appreciate that a single shared storage data hub creates a coordination point throughout the lifecycle without the need for extra data copies among the ingest, preprocessing, and training stages. Rarely is the ingested data used for only one purpose, and shared storage gives the flexibility to train multiple different models or apply traditional analytics to the data.

[0180] Readers will appreciate that each stage in the AI data pipeline may have varying requirements from the data hub (e.g., the storage system or collection of storage systems). Scale-out storage systems must deliver uncompromising performance for all manner of access types and patterns—from small, metadata-heavy to large files, from random to sequential access patterns, and from low to high concurrency. The storage systems described above may serve as an ideal AI data hub as the systems may service unstructured workloads. In the first stage, data is ideally ingested and stored on to the same data hub that following stages will use, in order to avoid excess data copying. The next two steps can be done on a standard compute server that optionally includes a GPU, and then in the fourth and last stage, full training production jobs are run on powerful GPU-accelerated servers. Often, there is a production pipeline alongside an experimental pipeline operating on the same dataset. Further, the GPU-accelerated servers can be used independently for different models or joined together to train on one larger model, even spanning multiple systems for distributed training. If the shared storage tier is slow, then data must be copied to local storage for each phase, resulting in wasted time staging data onto different servers. The ideal data hub for the AI training pipeline delivers performance similar to data stored locally on the server node while also having the simplicity and performance to enable all pipeline stages to operate concurrently.

[0181] In order for the storage systems described above to serve as a data hub or as part of an AI deployment, in some embodiments the storage systems may be configured to provide DMA between storage devices that are included in the storage systems and one or more GPUs that are used in an AI or big data analytics pipeline. The one or more GPUs may be coupled to the storage system, for example, via NVMe-over-Fabrics ('NVMe-oF') such that bottlenecks such as the host CPU can be bypassed and the storage system (or one of the components contained therein) can directly access GPU memory. In such an example, the storage systems may leverage API hooks to the GPUs to transfer data directly to the GPUs. For example, the GPUs may be embodied as Nvidia™ GPUs and the storage systems may support GPUDirect Storage ('GDS') software, or have similar proprietary software, that enables the storage system to transfer data to the GPUs via RDMA or similar mechanism. Readers will appreciate that in embodiments where the storage systems are embodied as cloud-based storage systems as described below, virtual drive or other components within such a cloud-based storage system may also be configured

[0182] Although the preceding paragraphs discuss deep learning applications, readers will appreciate that the storage systems described herein may also be part of a distributed deep learning ('DDL') platform to support the execution of DDL algorithms. The storage systems described above may also be paired with other technologies such as TensorFlow, an open-source software library for dataflow programming across a range of tasks that may be used for machine learning applications such as neural networks, to facilitate the development of such machine learning models, applications, and so on.

[0183] The storage systems described above may also be used in a neuromorphic computing environment. Neuromorphic computing is a form of computing that mimics brain cells. To support neuromorphic computing, an architecture of interconnected "neurons" replace traditional computing models with low-powered signals that go directly between neurons for more efficient computation. Neuromorphic computing may make use of very-large-scale integration (VLSI) systems containing electronic analog circuits to mimic neuro-biological architectures present in the nervous system, as well as analog, digital, mixed-mode analog/digital VLSI, and software systems that implement models of neural systems for perception, motor control, or multisensory integration.

[0184] Readers will appreciate that the storage systems described above may be configured to support the storage or use of (among other types of data) blockchains. In addition to supporting the storage and use of blockchain technologies, the storage systems described above may also support the storage and use of derivative items such as, for example, open source blockchains and related tools that are part of the IBM™ Hyperledger project, permissioned blockchains in which a certain number of trusted parties are allowed to access the block chain, blockchain products that enable developers to build their own distributed ledger projects, and others. Blockchains and the storage systems described herein may be leveraged to support on-chain storage of data as well as off-chain storage of data.

[0185] Off-chain storage of data can be implemented in a variety of ways and can occur when the data itself is not stored within the blockchain. For example, in one embodiment, a hash function may be utilized and the data itself may be fed into the hash function to generate a hash value. In such an example, the hashes of large pieces of data may be embedded within transactions, instead of the data itself. Readers will appreciate that, in other embodiments, alternatives to blockchains may be used to facilitate the decentralized storage of information. For example, one alternative to a blockchain that may be used is a blockweave. While conventional blockchains store every transaction to achieve validation, a blockweave permits secure decentralization without the usage of the entire chain, thereby enabling low cost on-chain storage of data. Such blockweaves may utilize a consensus mechanism that is based on proof of access (PoA) and proof of work (PoW).

[0186] The storage systems described above may, either alone or in combination with other computing devices, be used to support in-memory computing applications. In-memory computing involves the storage of information in RAM that is distributed across a cluster of computers. Readers will appreciate that the storage systems described above, especially those that are configurable with customizable amounts of processing resources, storage resources, and memory resources (e.g., those systems in which blades that contain configurable amounts of each type of resource), may be configured in a way so as to provide an infrastructure that can support in-memory computing. Likewise, the storage systems described above may include component parts (e.g., NVDIMMs, 3D crosspoint storage that provide fast random access memory that is persistent) that can actually provide for an improved in-memory computing environment as compared to in-memory computing environments that rely on RAM distributed across dedicated servers.

[0187] In some embodiments, the storage systems described above may be configured to operate as a hybrid in-memory computing environment that includes a universal interface to all storage media (e.g., RAM, flash storage, 3D crosspoint storage). In such embodiments, users may have no knowledge regarding the details of where their data is stored but they can still use the same full, unified API to address data. In such embodiments, the storage system may (in the background) move data to the fastest layer available—including intelligently placing the data in dependence upon various characteristics of the data or in dependence upon some other heuristic. In such an example, the storage systems may even make use of existing products such as Apache Ignite and GridGain to move data between the various storage layers, or the storage systems may make use of custom software to move data between the various storage layers. The storage systems described herein may implement various optimizations to improve the performance of in-memory computing such as, for example, having computations occur as close to the data as possible.

[0188] Readers will further appreciate that in some embodiments, the storage systems described above may be paired with other resources to support the applications described above. For example, one infrastructure could include primary compute in the form of servers and workstations which specialize in using General-purpose computing on graphics processing units ('GPGPU') to accelerate deep learning applications that are interconnected into a computation engine to train parameters for deep neural networks. Each system may have Ethernet external connectivity, InfiniBand external connectivity, some other form of external connectivity, or some combination thereof. In such an example, the GPUs can be grouped for a single large training or used independently to train multiple models. The infrastructure could also include a storage system such as those described above to provide, for example, a scale-out all-flash file or object store through which data can be accessed via high-performance protocols such as NFS, S3, and so on. The infrastructure can also include, for example, redundant top-of-rack Ethernet switches connected to storage and compute via ports in MLAG port channels for redundancy. The infrastructure could also include additional compute in the form of whitebox servers, optionally with GPUs, for data ingestion, pre-processing, and model debugging. Readers will appreciate that additional infrastructures are also be possible.

[0189] Readers will appreciate that the storage systems described above, either alone or in coordination with other computing machinery may be configured to support other AI related tools. For example, the storage systems may make use of tools like ONXX or other open neural network exchange formats that make it easier to transfer models written in different AI frameworks. Likewise, the storage systems may be configured to support tools like Amazon's Gluon that allow developers to prototype, build, and train deep learning models. In fact, the storage systems described above may be part of a larger platform, such as IBM™ Cloud Private for Data, that includes integrated data science, data engineering and application building services.

[0190] Readers will further appreciate that the storage systems described above may also be deployed as an edge solution. Such an edge solution may be in place to optimize cloud computing systems by performing data processing at the edge of the network, near the source of the data. Edge computing can push applications, data and computing power (i.e., services) away from centralized points to the logical extremes of a network. Through the use of edge solutions such as the storage systems described above, computational tasks may be performed using the compute resources provided by such storage systems, data may be storage using the storage resources of the storage system, and cloud-based services may be accessed through the use of various resources of the storage system (including networking resources). By performing computational tasks on the edge solution, storing data on the edge solution, and generally making use of the edge solution, the consumption of expensive cloud-based resources may be avoided and, in fact, performance improvements may be experienced relative to a heavier reliance on cloud-based resources.

[0191] While many tasks may benefit from the utilization of an edge solution, some particular uses may be especially suited for deployment in such an environment. For example, devices like drones, autonomous cars, robots, and others may require extremely rapid processing—so fast, in fact, that sending data up to a cloud environment and back to receive data processing support may simply be too slow. As an additional example, some IoT devices such as connected video cameras may not be well-suited for the utilization of cloud-based resources as it may be impractical (not only from a privacy perspective, security perspective, or a financial perspective) to send the data to the cloud simply because of the pure volume of data that is involved. As such, many tasks that really on data processing, storage, or communications may be better suited by platforms that include edge solutions such as the storage systems described above.

[0192] The storage systems described above may alone, or in combination with other computing resources, serves as a network edge platform that combines compute resources, storage resources, networking resources, cloud technologies and network virtualization technologies, and so on. As part of the network, the edge may take on characteristics similar to other network facilities, from the customer premise and backhaul aggregation facilities to Points of Presence (PoPs) and regional data centers. Readers will appreciate that network workloads, such as Virtual Network Functions (VNFs) and others, will reside on the network edge platform. Enabled by a combination of containers and virtual machines, the network edge platform may rely on controllers and schedulers that are no longer geographically co-located with the data processing resources. The functions, as

microservices, may split into control planes, user and data planes, or even state machines, allowing for independent optimization and scaling techniques to be applied. Such user and data planes may be enabled through increased accelerators, both those residing in server platforms, such as FPGAs and Smart NICs, and through SDN-enabled merchant silicon and programmable ASICs.

[0193] The storage systems described above may also be optimized for use in big data analytics. Big data analytics may be generally described as the process of examining large and varied data sets to uncover hidden patterns, unknown correlations, market trends, customer preferences and other useful information that can help organizations make more-informed business decisions. As part of that process, semi-structured and unstructured data such as, for example, internet clickstream data, web server logs, social media content, text from customer emails and survey responses, mobile-phone call-detail records, IoT sensor data, and other data may be converted to a structured form.

[0194] The storage systems described above may also support (including implementing as a system interface) applications that perform tasks in response to human speech. For example, the storage systems may support the execution intelligent personal assistant applications such as, for example, Amazon's Alexa, Apple Siri, Google Voice, Samsung Bixby, Microsoft Cortana, and others. While the examples described in the previous sentence make use of voice as input, the storage systems described above may also support chatbots, talkbots, chatterbots, or artificial conversational entities or other applications that are configured to conduct a conversation via auditory or textual methods. Likewise, the storage system may actually execute such an application to enable a user such as a system administrator to interact with the storage system via speech. Such applications are generally capable of voice interaction, music playback, making to-do lists, setting alarms, streaming podcasts, playing audiobooks, and providing weather, traffic, and other real time information, such as news, although in embodiments in accordance with the present disclosure, such applications may be utilized as interfaces to various system management operations.

[0195] The storage systems described above may also implement AI platforms for delivering on the vision of self-driving storage. Such AI platforms may be configured to deliver global predictive intelligence by collecting and analyzing large amounts of storage system telemetry data points to enable effortless management, analytics and support. In fact, such storage systems may be capable of predicting both capacity and performance, as well as generating intelligent advice on workload deployment, interaction and optimization. Such AI platforms may be configured to scan all incoming storage system telemetry data against a library of issue fingerprints to predict and resolve incidents in real-time, before they impact customer environments, and captures hundreds of variables related to performance that are used to forecast performance load.

[0196] The storage systems described above may support the serialized or simultaneous execution of artificial intelligence applications, machine learning applications, data analytics applications, data transformations, and other tasks that collectively may form an AI ladder. Such an AI ladder may effectively be formed by combining such elements to form a complete data science pipeline, where exist dependencies between elements of the AI ladder. For example, AI may require that some form of machine learning has taken place, machine learning may require that some form of analytics has taken place, analytics may require that some form of data and information architecting has taken place, and so on. As such, each element may be viewed as a rung in an AI ladder that collectively can form a complete and sophisticated AI solution.

[0197] The storage systems described above may also, either alone or in combination with other computing environments, be used to deliver an AI everywhere experience where AI permeates wide and expansive aspects of business and life. For example, AI may play an important role in the delivery of deep learning solutions, deep reinforcement learning solutions, artificial general intelligence solutions, autonomous vehicles, cognitive computing solutions, commercial UAVs or drones, conversational user interfaces, enterprise taxonomies, ontology management solutions, machine learning solutions, smart dust, smart robots, smart workplaces, and many others.

[0198] The storage systems described above may also, either alone or in combination with other computing environments, be used to deliver a wide range of transparently immersive experiences (including those that use digital twins of various "things" such as people, places, processes, systems, and so on) where technology can introduce transparency between people, businesses, and things. Such transparently immersive experiences may be delivered as augmented reality technologies, connected homes, virtual reality technologies, brain—computer interfaces, human augmentation technologies, nanotube electronics, volumetric displays, 4D printing technologies, or others.

[0199] The storage systems described above may also, either alone or in combination with other computing environments, be used to support a wide variety of digital platforms. Such digital platforms can include, for example, 5G wireless systems and platforms, digital twin platforms, edge computing platforms, IoT platforms, quantum computing platforms, serverless PaaS, software-defined security, neuromorphic computing platforms, and so on.

[0200] The storage systems described above may also be part of a multi-cloud environment in which multiple cloud computing and storage services are deployed in a single heterogeneous architecture. In order to facilitate the operation of such a multi-cloud environment, DevOps tools may be deployed to enable orchestration across clouds. Likewise, continuous development and continuous integration tools may be deployed to standardize processes around continuous integration and delivery, new feature rollout and provisioning cloud workloads. By standardizing these processes, a multi-cloud strategy may be implemented that enables the utilization of the best provider for each workload.

[0201] The storage systems described above may be used as a part of a platform to enable the use of crypto-anchors that may be used to authenticate a product's origins and contents to ensure that it matches a blockchain record associated with the product. Similarly, as part of a suite of tools to secure data stored on the storage system, the storage systems described above may implement various encryption technologies and schemes, including lattice cryptography. Lattice cryptography can involve constructions of cryptographic primitives that involve lattices, either in the construction itself or in the security proof. Unlike public-key schemes such as the RSA, Diffie-Hellman or Elliptic-Curve cryptosystems, which are easily attacked by a quantum

computer, some lattice-based constructions appear to be resistant to attack by both classical and quantum computers.

[0202] A quantum computer is a device that performs quantum computing. Quantum computing is computing using quantum-mechanical phenomena, such as superposition and entanglement. Quantum computers differ from traditional computers that are based on transistors, as such traditional computers require that data be encoded into binary digits (bits), each of which is always in one of two definite states (0 or 1). In contrast to traditional computers, quantum computers use quantum bits, which can be in superpositions of states. A quantum computer maintains a sequence of qubits, where a single qubit can represent a one, a zero, or any quantum superposition of those two qubit states. A pair of qubits can be in any quantum superposition of 4 states, and three qubits in any superposition of 8 states. A quantum computer with n qubits can generally be in an arbitrary superposition of up to 2^n different states simultaneously, whereas a traditional computer can only be in one of these states at any one time. A quantum Turing machine is a theoretical model of such a computer.

[0203] The storage systems described above may also be paired with FPGA-accelerated servers as part of a larger AI or ML infrastructure. Such FPGA-accelerated servers may reside near (e.g., in the same data center) the storage systems described above or even incorporated into an appliance that includes one or more storage systems, one or more FPGA-accelerated servers, networking infrastructure that supports communications between the one or more storage systems and the one or more FPGA-accelerated servers, as well as other hardware and software components. Alternatively, FPGA-accelerated servers may reside within a cloud computing environment that may be used to perform compute-related tasks for AI and ML jobs. Any of the embodiments described above may be used to collectively serve as a FPGA-based AI or ML platform. Readers will appreciate that, in some embodiments of the FPGA-based AI or ML platform, the FPGAs that are contained within the FPGA-accelerated servers may be reconfigured for different types of ML models (e.g., LSTMs, CNNs, GRUs). The ability to reconfigure the FPGAs that are contained within the FPGA-accelerated servers may enable the acceleration of a ML or AI application based on the most optimal numerical precision and memory model being used. Readers will appreciate that by treating the collection of FPGA-accelerated servers as a pool of FPGAs, any CPU in the data center may utilize the pool of FPGAs as a shared hardware microservice, rather than limiting a server to dedicated accelerators plugged into it.

[0204] The FPGA-accelerated servers and the GPU-accelerated servers described above may implement a model of computing where, rather than keeping a small amount of data in a CPU and running a long stream of instructions over it as occurred in more traditional computing models, the machine learning model and parameters are pinned into the high-bandwidth on-chip memory with lots of data streaming though the high-bandwidth on-chip memory. FPGAs may even be more efficient than GPUs for this computing model, as the FPGAs can be programmed with only the instructions needed to run this kind of computing model.

[0205] The storage systems described above may be configured to provide parallel storage, for example, through the use of a parallel file system such as BeeGFS. Such parallel files systems may include a distributed metadata architecture. For example, the parallel file system may include a plurality of metadata servers across which metadata is distributed, as well as components that include services for clients and storage servers.

[0206] The systems described above can support the execution of a wide array of software applications. Such software applications can be deployed in a variety of ways, including container-based deployment models. Containerized applications may be managed using a variety of tools. For example, containerized applications may be managed using Docker Swarm, Kubernetes, and others. Containerized applications may be used to facilitate a serverless, cloud native computing deployment and management model for software applications. In support of a serverless, cloud native computing deployment and management model for software applications, containers may be used as part of an event handling mechanisms (e.g., AWS Lambdas) such that various events cause a containerized application to be spun up to operate as an event handler.

[0207] The systems described above may be deployed in a variety of ways, including being deployed in ways that support fifth generation ('5G') networks. 5G networks may support substantially faster data communications than previous generations of mobile communications networks and, as a consequence may lead to the disaggregation of data and computing resources as modern massive data centers may become less prominent and may be replaced, for example, by more-local, micro data centers that are close to the mobile-network towers. The systems described above may be included in such local, micro data centers and may be part of or paired to multi-access edge computing ('MEC') systems. Such MEC systems may enable cloud computing capabilities and an IT service environment at the edge of the cellular network. By running applications and performing related processing tasks closer to the cellular customer, network congestion may be reduced and applications may perform better.

[0208] The storage systems described above may also be configured to implement NVMe Zoned Namespaces. Through the use of NVMe Zoned Namespaces, the logical address space of a namespace is divided into zones. Each zone provides a logical block address range that must be written sequentially and explicitly reset before rewriting, thereby enabling the creation of namespaces that expose the natural boundaries of the device and offload management of internal mapping tables to the host. In order to implement NVMe Zoned Name Spaces ('ZNS'), ZNS SSDs or some other form of zoned block devices may be utilized that expose a namespace logical address space using zones. With the zones aligned to the internal physical properties of the device, several inefficiencies in the placement of data can be eliminated. In such embodiments, each zone may be mapped, for example, to a separate application such that functions like wear levelling and garbage collection could be performed on a per-zone or per-application basis rather than across the entire device. In order to support ZNS, the storage controllers described herein may be configured with to interact with zoned block devices through the usage of, for example, the Linux™ kernel zoned block device interface or other tools.

[0209] The storage systems described above may also be configured to implement zoned storage in other ways such as, for example, through the usage of shingled magnetic recording (SMR) storage devices. In examples where zoned

storage is used, device-managed embodiments may be deployed where the storage devices hide this complexity by managing it in the firmware, presenting an interface like any other storage device. Alternatively, zoned storage may be implemented via a host-managed embodiment that depends on the operating system to know how to handle the drive, and only write sequentially to certain regions of the drive. Zoned storage may similarly be implemented using a host-aware embodiment in which a combination of a drive managed and host managed implementation is deployed.

[0210] For further explanation, FIG. **5** illustrates an exemplary computing device **350** that may be specifically configured to perform one or more of the processes described herein. As shown in FIG. **5**, computing device **350** may include a communication interface **352**, a processor **354**, a storage device **356**, and an input/output ("I/O") module **358** communicatively connected one to another via a communication infrastructure **360**. While an exemplary computing device **350** is shown in FIG. **5**, the components illustrated in FIG. **5** are not intended to be limiting. Additional or alternative components may be used in other embodiments. Components of computing device **350** shown in FIG. **5** will now be described in additional detail.

[0211] Communication interface **352** may be configured to communicate with one or more computing devices. Examples of communication interface **352** include, without limitation, a wired network interface (such as a network interface card), a wireless network interface (such as a wireless network interface card), a modem, an audio/video connection, and any other suitable interface.

[0212] Processor **354** generally represents any type or form of processing unit capable of processing data and/or interpreting, executing, and/or directing execution of one or more of the instructions, processes, and/or operations described herein. Processor **354** may perform operations by executing computer-executable instructions **362** (e.g., an application, software, code, and/or other executable data instance) stored in storage device **356**.

[0213] Storage device **356** may include one or more data storage media, devices, or configurations and may employ any type, form, and combination of data storage media and/or device. For example, storage device **356** may include, but is not limited to, any combination of the non-volatile media and/or volatile media described herein. Electronic data, including data described herein, may be temporarily and/or permanently stored in storage device **356**. For example, data representative of computer-executable instructions **362** configured to direct processor **354** to perform any of the operations described herein may be stored within storage device **356**. In some examples, data may be arranged in one or more databases residing within storage device **356**.

[0214] I/O module **358** may include one or more I/O modules configured to receive user input and provide user output. I/O module **358** may include any hardware, firmware, software, or combination thereof supportive of input and output capabilities. For example, I/O module **358** may include hardware and/or software for capturing user input, including, but not limited to, a keyboard or keypad, a touchscreen component (e.g., touchscreen display), a receiver (e.g., an RF or infrared receiver), motion sensors, and/or one or more input buttons.

[0215] I/O module **358** may include one or more devices for presenting output to a user, including, but not limited to, a graphics engine, a display (e.g., a display screen), one or more output drivers (e.g., display drivers), one or more audio speakers, and one or more audio drivers. In certain embodiments, I/O module **358** is configured to provide graphical data to a display for presentation to a user. The graphical data may be representative of one or more graphical user interfaces and/or any other graphical content as may serve a particular implementation. In some examples, any of the systems, computing devices, and/or other components described herein may be implemented by computing device **350**.

[0216] The storage systems described above may, either alone or in combination, by configured to serve as a continuous data protection store. A continuous data protection store is a feature of a storage system that records updates to a dataset in such a way that consistent images of prior contents of the dataset can be accessed with a low time granularity (often on the order of seconds, or even less), and stretching back for a reasonable period of time (often hours or days). These allow access to very recent consistent points in time for the dataset, and also allow access to access to points in time for a dataset that might have just preceded some event that, for example, caused parts of the dataset to be corrupted or otherwise lost, while retaining close to the maximum number of updates that preceded that event. Conceptually, they are like a sequence of snapshots of a dataset taken very frequently and kept for a long period of time, though continuous data protection stores are often implemented quite differently from snapshots. A storage system implementing a data continuous data protection store may further provide a means of accessing these points in time, accessing one or more of these points in time as snapshots or as cloned copies, or reverting the dataset back to one of those recorded points in time.

[0217] Over time, to reduce overhead, some points in the time held in a continuous data protection store can be merged with other nearby points in time, essentially deleting some of these points in time from the store. This can reduce the capacity needed to store updates. It may also be possible to convert a limited number of these points in time into longer duration snapshots. For example, such a store might keep a low granularity sequence of points in time stretching back a few hours from the present, with some points in time merged or deleted to reduce overhead for up to an additional day. Stretching back in the past further than that, some of these points in time could be converted to snapshots representing consistent point-in-time images from only every few hours.

[0218] Although some embodiments are described largely in the context of a storage system, readers of skill in the art will recognize that embodiments of the present disclosure may also take the form of a computer program product disposed upon computer readable storage media for use with any suitable processing system. Such computer readable storage media may be any storage medium for machine-readable information, including magnetic media, optical media, solid-state media, or other suitable media. Examples of such media include magnetic disks in hard drives or diskettes, compact disks for optical drives, magnetic tape, and others as will occur to those of skill in the art. Persons skilled in the art will immediately recognize that any computer system having suitable programming means will be capable of executing the steps described herein as embodied in a computer program product. Persons skilled in the art

will recognize also that, although some of the embodiments described in this specification are oriented to software installed and executing on computer hardware, nevertheless, alternative embodiments implemented as firmware or as hardware are well within the scope of the present disclosure.

[0219] In some examples, a non-transitory computer-readable medium storing computer-readable instructions may be provided in accordance with the principles described herein. The instructions, when executed by a processor of a computing device, may direct the processor and/or computing device to perform one or more operations, including one or more of the operations described herein. Such instructions may be stored and/or transmitted using any of a variety of known computer-readable media.

[0220] A non-transitory computer-readable medium as referred to herein may include any non-transitory storage medium that participates in providing data (e.g., instructions) that may be read and/or executed by a computing device (e.g., by a processor of a computing device). For example, a non-transitory computer-readable medium may include, but is not limited to, any combination of non-volatile storage media and/or volatile storage media. Exemplary non-volatile storage media include, but are not limited to, read-only memory, flash memory, a solid-state drive, a magnetic storage device (e.g. a hard disk, a floppy disk, magnetic tape, etc.), ferroelectric random-access memory ("RAM"), and an optical disc (e.g., a compact disc, a digital video disc, a Blu-ray disc, etc.). Exemplary volatile storage media include, but are not limited to, RAM (e.g., dynamic RAM).

[0221] FIGS. **6** and **7** describe a multithreaded multimode NAND flash controller that can be used with multiple flash memory devices having the same or different flash memory interfaces, protocols, operating frequencies and/or signal timing, in various combinations and in various storage devices and systems. The flash controller can select from among multiple protocols, adjust and calibrate operating frequency and signal timing relative to each selected protocol and associated flash memory device interface, independently for each of multiple channels, and dynamically adapt signal rates to varying conditions that flash memory devices experience over time. Such tasks and capabilities are accomplished with a multithreaded and virtualized microcode sequence engine, individual channel configuration, and software calibrated I/O (input/output). Some embodiments can recalibrate signal rates (e.g., signal frequency and/or timing) to compensate for short-term drift the flash memory devices may experience as a result of environmental conditions such as temperature variation, power supply variation, noise, etc. Some embodiments can recalibrate signal rates to compensate for long-term drift or errors that flash memory devices may experience as a result of device wear arising from cumulative reads, cumulative writes, cumulative erasure cycles, etc.

[0222] There are fundamental differences between the ONFI and Toggle protocols in terms of physical flash signaling layer. The present flash controller design allows abstraction of much of the low-level complexity away from upper-level software. Upper-level software could, for example, issue "flash read" or "flash write" commands which in turn are processed differently by the controller depending upon the type of flash to which the controller is communicating. The physical controller could decode the command and translate the decoded command to the correct protocol, depending upon the type of flash and corresponding channel configuration.

[0223] FIG. **6** depicts a flash controller **102** that is configurable to couple to flash memories **106**, **108** with differing flash memory device interfaces. In some embodiments, the flash controller **102** is implemented on a programmable logic device (PLD) or application-specific integrated circuit (ASIC), and includes a processor capable of multithreaded operation and various modules that can be implemented in circuitry, firmware, software executing on the processor, or various combinations thereof. Flash controller **102** corresponds to PLD **208** of FIG. **3** in some embodiments. Software program commands **110** are written into the flash controller **102**, for example from an external device that has a processor. Each physical interface **104**, or phy, is selectable as to protocol, operating frequency and signal timing, as appropriate to the specific NAND flash **106**, **108** and associated flash memory device interface coupled to that physical interface **104**. The physical interfaces **104** are independent of each other in the above and below-described selectability and tunability. In the example shown, one of the flash memory devices **106** is a Toshiba™ NAND flash, and another one of the flash memory devices **108** is a Micron™ NAND flash, but other flash memory devices from other manufacturers, or that have other flash memory interfaces and/or other protocols, could be used.

[0224] FIG. **7**A is a block diagram showing structural details of an embodiment of the flash controller **102** of FIG. **6**, including a multithreaded/virtualized microcode sequence engine **213** and multiple channels **215**, each with phy (physical) controls **217**, **219**, channel configuration registers **221** and a software calibrated I/O module **223**. An embodiment of the flash controller **102** is depicted with N channels **215**, which could be two channels, three channels, four channels, etc., i.e., for N greater than or equal to two up to however many channels can be physically produced on physical device(s) for the flash controller **102**. Each channel **215** is independent of each other channel **215**, as to flash memory protocol, operating frequency and/or signal rates of the flash memory device interface, and signal timing relative to the selected flash memory protocol. It should be appreciated that signal rate, as used herein, is inclusive of frequency and/or signal timing. In FIG. **7**A, the channel **215** labeled channel 1 is shown with Toshiba™ Toggle phy controls **217** (i.e., physical device controls for the Toggle protocol according to the Toshiba™ manufacturer flash devices), per the selected protocol for channel 1. Channel 1 is coupled to multiple NAND flash devices **106**, which, in this example, are Toshiba™ flash memories that use the Toggle protocol. Channel configuration registers **221** for channel 1 are loaded with the appropriate values to direct the software calibrated I/O module **223** for channel 1 to time sequences in a protocol (e.g., by timing state machine states, microcode sequences or events, etc.) or to produce timed signals at a particular operating frequency (or signal rate) for the flash devices **106**, in various embodiments. A process for how the channel configuration registers **221** are loaded, and a mechanism for how the software calibrated I/O module **223** generates timing for signal rates or generates signals in some embodiments.

[0225] Each channel **215** in the flash controller **102** has its own phy controls **217**, **219**, channel configuration registers **221** and software calibrated I/O module **223**, the combina-

tion of which are selectable and tunable on an individual, per channel basis, as to protocol, operating frequency, and signal timing. The channel **215** labeled channel N is depicted as having Micron™ ONFI (Open NAND Flash Interface) phy controls **219** (i.e., physical device controls for the ONFI protocol according to the Micron™ manufacturer flash devices), per the selected protocol for channel N. Channel N is coupled to multiple NAND flash devices **108**, which, in this example, are Micron™ flash memories that use the ONFI protocol. The flash controller **102** could be operated with flash devices **106** that are all the same (or flash devices **108** that are all the same, etc.), or mixes of flash devices **106**, **108** of the various protocols, flash memory device interfaces and manufacturers. Each channel **215** should have the same flash memory devices across that channel **215**, but which flash memory device and associated flash memory device interface that channel has is independent of each other channel.

[0226] Software program commands **110**, which are device independent (i.e., not dependent on a particular flash memory protocol or flash memory device interface) are written by an external device (i.e., a device external to the flash controller **102**), such as a processor, into the microcode command FIFO **207** of the flash controller **102**. Read/write data **203** is read from or written into the data FIFOs **209**. More specifically, write data intended for the flash memories is written into one or more write FIFOs, and read data from the flash memories is read from one or more read FIFOs, collectively illustrated as data FIFOs **209**. A memory mapped control/configuration interface **211** is used for control/configuration data, which could also be from an external device such as a processor. The microcode command FIFO **207**, the data FIFOs **209**, and the memory mapped control/configuration interface **211** are coupled to the multithreaded/virtualized microcode sequence engine **213**, which couples to the channels **215**, e.g., channels 1 through N. Each channel **215** has a dedicated one or more threads, in a multithreaded operation of the multithreaded/virtualized microcode sequence engine **213**. This multithreading virtualizes the microcode sequence engine **213**, as if each channel **215** had its own microcode sequence engine **213**. In further embodiments, there are multiple physical microcode sequence engines **213**, e.g., in a multiprocessing multithreaded operation. This would still be considered an embodiment of the multithreaded/virtualized microcode sequence engine **213**.

[0227] In some embodiments, state machines control the channels **215**. These may act as the above-described virtualized microcode sequence engines **213**. For example, in various embodiments, each channel has a state machine, or a state machine could control two channels, two state machines could control each channel, etc. These state machines could be implemented in hardware and fed by the multithreaded/virtualized microcode sequence engine **213**, or implemented in threads of the multithreaded/virtualized microcode sequence engine **213**, or combinations thereof. The functionality provided by the state machines is described in more detail below. In some embodiments, software injects commands into state machine queues, and state machines arbitrate for channels, then issue read or write commands to the channels, depending upon operations. In some embodiments, the state machines implement reads, writes and erases, with other commands such as reset, initialization sequences, feature settings, etc., communicated from an external processor along a bypass path which could be controlled by a register. Each state machine could have multiple states for a write, further states for a read, and still further states for erasure cycle(s), with timing and/or frequency (i.e., as affect signal rate) controlled by states, state transitions, and/or an embodiment of the software calibrated I/O module **223**.

[0228] The microcode command FIFO **207** allows upstream logic to present transactions to the flash controller **102**. The format of the command allows for the upstream logic to present entire transactions (with indicators for start of transaction, and end of transaction). The flash controller begins operating upon entire transactions on receipt of end of transaction markers, in some embodiments. In addition to the microcode command FIFO **207**, there are two data FIFOs **209**, and in some embodiments more than two, to handle data flowing in and out of flash. Also, there is a memory-mapped register interface **211** for the upstream logic to be able to program the different parameters used to set up the flash controller (e.g., calibration, flash mode, flash type, etc.) as described above.

[0229] FIG. **7**B is a block diagram showing further structural details of an embodiment of the flash controller **102** of FIG. **6**, including a scheduler **702** with a scoreboard **706**, an arbiter **714** with configurable policies **716**, and a sequencer **704** with multiple channels **215** each with a phy **718** and flash memory **106**, **108**. For optimal scheduling of flash operations, the scheduler **702** receives operation requests **710**, such as read, write or erasure requests, from various masters **708**. From each operation request **710**, the scheduler forms a sequence of one or more operation phases **712**. These masters **708** are, for example, storage nodes **150** or differing authorities **168** in various storage nodes **150** (including compute-only nodes) making I/O requests to a storage unit **152** that has the flash controller **102** in the storage unit **152**. Examples of splitting operation requests into operation phases are further discussed below with reference to FIG. **10**. The scheduler **702** includes the arbiter **714**, which arbitrates the operation requests **710** and the operation phases **712**, according to various configurable policies **716**. These operation phases **712** are flash memory device independent commands, in some embodiments.

[0230] Policies **716** could specify priorities for operation requests **710** and operation phases **712**, in various rules that depend on various factors such as arrival sequences, aging of requests or phases, origins of requests, memory conditions, etc., and could have optimization for latency, bandwidth or quality of service, etc. For example, one of the masters could have a faster path with higher priority in arbitration according to a policy, and another master could have a slower path with lower priority in arbitration according to another policy. Garbage collection would be an example of such a slower path with lower priority, and memory reads for a client expecting high quality of service would be an example of a faster path with higher priority. Another example is one master **708** performing read recovery with the scheduler **702** issuing operation phases **712** for varying a voltage threshold of a flash memory device through one of the channels **215**, while the scheduler **702** is concurrently receiving and arbitrating another operation request **710** from another master **708** and issuing operation phases **712** to another flash memory device through another channel **215**. The policies **716** are configurable in some

embodiments, so that priorities for the arbiter **714** can be set and changed for various system operations and clients.

[0231] In some embodiments, the sequencer **704** includes the multithreaded/virtualized sequence engine **213** of FIG. **7**A. In some embodiments, each phy **718** includes a single phy, or multiple phys, which could be differing phy controls **217**, **219** as shown in FIG. **7**A. The sequencer **704** receives the operation phases and communicates, through the phy **718**, to flash memory according to the operation phases. For example, the sequencer **704** could operate one or more phy controls **217** to communicate to one type of flash memory **106**, and operate one or more phy controls **219** to communicate to another type of flash memory **108**, through the channels **215**. These flash memory devices could have different types of flash memory device interfaces, as described above with reference to FIG. **7**A, or they could all have the same type of flash memory device interface. The sequencer **704** produces the differing signaling for the differing types of flash memory device interfaces. The sequencer **704** uses the scoreboard **706**, which can be implemented as a data structure in memory, to track status of each of the operation phases **712**, in some embodiments.

[0232] FIG. **8** is a high level block diagram of the flash controller in accordance with some embodiments. The host processor, which can correspond to CPU **156** of FIGS. **1** and **3** in some embodiments, issues commands to the CPU **801** that lives in the non-volatile solid state storage **152** of FIGS. **1** and **3** in some embodiments. Communication between the host processor and the non-volatile solid state storage is done via an Non Volatile Memory express (NVMe) like protocol that runs over a PCIe interface in some embodiments. CPU **801** is then responsible for sending the flash reads and writes to the appropriate flash channels. Data read out of flash is passed through an ECC decode engine **803**, and then emptied into a FIFO **805**. Once the data is available in this FIFO **805**, it is transferred to the host machine by a DMA engine **214** (configured by the CPU **801**). Writes from the host machine are of two types in some embodiments: 1) NVRAM writes (that live in a portion of non-volatile solid state storage DRAM inaccessible to the CPU **801**), and 2) flash writes. All write data is buffered in DRAM. Flash writes are then sent to the appropriate flash channels by the CPU **801**. This data passes through an ECC encode engine **807** before being transmitted into flash. At power-down time, the FPGA logic is responsible for transferring all the NVRAM contents from DRAM **216** to flash **106**, **108**.

[0233] FIG. **9** is a microarchitecture diagram of the flash controller in accordance with some embodiments. The main flows for the reads and writes can be summarized as follows:

Reads

[0234] From the flash devices to host DRAM (responses to flash read commands)

[0235] From the flash devices to SU DRAM (for partial-processing by the CPU **801**/NVRAM recovery)

[0236] Writes

[0237] From host DRAM to SU DRAM (for NVRAM updates/buffering flash pages before being persisted)

[0238] From SU DRAM to flash devices (persisting flash data/saving NVRAM during a power loss)

[0239] Read Flows through the microarchitecture diagram are outlined below with the number for each step corresponding to the like circled number in the action flow of FIG. **9**. Additional details for the Read Flows may be found with reference to FIGS. **11** and **13**A.

[0240] 1. The host uses a proprietary NVMe-like protocol to signal a new request to the storage unit. This request is written into the submission queue interface **902** exposed by the non-volatile solid state storage, also referred to as a storage unit.

[0241] 2. The command is written to non-volatile solid state storage DRAM **914**, and is then consumed by the CPU **801**.

[0242] 3. Software on the CPU **801** assigns a transaction ID, and builds a scoreboard that tracks the progress of the transaction. Relevant fields tracked through the scoreboard may include transaction ID, addresses, size, etc. This information is also written to the Read Destination Table (RDT) **904**, which tells the FPGA where to direct data.

[0243] 4. Software is responsible for injecting the command into the state machines queues. Every state machine is a slave that can be driven by two masters in some embodiments: the CPU **801**, and the power down/loss engine **906**. There is a logical mux **908** that selects between the two sources.

[0244] 5. The state machine, upon receipt of the command, arbitrates for the channel. The state machine is cognizant of plane/bus conflicts on the flash device. Each state machine controls two physical hardware channels, and has a logical view of 16 chip-enables in some embodiments.

[0245] 6. The state machine issues the read commands to the flash channel. The state machine is also responsible for polling the flash registers to check if data is ready (this mechanism is in lieu of using the ready/busy bit in some embodiments).

[0246] 7. When data is ready for transfer, the state machine attempts to win arbitration of the upstream bus. Upon winning control of the bus, it starts transferring data from the channel to the staging queues **910** that are located in front of the ECC decode engine. The queues **910** are banked such that eight channels write to one bank in some embodiments. In some embodiments, all transfers will be one page (e.g., 16K) long.

[0247] 8. When a codeword (e.g., 2K) is ready, the decode process is initiated. There are two ECC decode data paths in some embodiments. After decode, the data goes through cyclical redundancy check (CRC) calculations, and is inserted into another queue **912**.

[0248] 9. The front-end is alerted after the entire transaction is decoded. It then initiates a data transfer to the destination buffer (either non-volatile solid state storage DRAM **914**, or the host **916**, as programmed into the RDT **904**). The front-end alerts the back-end, upon completion of the transfer.

[0249] 10. The back-end then generates an interrupt to the CPU **801**. The CPU **801** reads the state machine registers to glean information about the operation (e.g. number of bits corrected by the decoder, errors, etc.).

[0250] 11. The CPU **801** cleans up its scoreboard, and signals a completion back to the host **916** via the completion queue **918**.

[0251] Write Flows through the microarchitecture diagram are outlined below with the number for each step corresponding to the like circled number in the action flow

of FIG. **9**. Additional details for the Write Flows may be found with reference to FIGS. **12** and **13**A.

[0252] 1. The host uses a proprietary NVMe-like protocol to signal a new request to the storage unit. This request is written into the submission queue interface **902** exposed by the non-volatile solid state storage, also referred to as a storage unit.

[0253] 2. The command is written to non-volatile solid state storage DRAM **914**, and is then consumed by the CPU **801**.

[0254] 3. Software on the CPU **801** assigns a transaction ID, and builds a scoreboard that tracks the progress of the transaction. Relevant fields tracked through the scoreboard may include transaction ID, addresses, size, etc. This information is also written to the Read Destination Table (RDT) **904**, which tells the FPGA where to direct data.

[0255] 4. Software is responsible for injecting the command into the state machines queues. Every state machine is a slave that can be driven by two masters in some embodiments: the CPU **801**, and the power down/loss engine **906**. There is a logical mux **908** that selects between the two sources.

[0256] 5. The state machine, upon receipt of the command, arbitrates for the channel. The state machine is cognizant of plane/bus conflicts on the flash device. Each state machine controls two physical hardware channels, and has a logical view of 16 chip-enables in some embodiments.

[0257] 6. The state machine issues the read commands to the flash channel. The state machine is also responsible for polling the flash registers to check if data is ready (this mechanism is in lieu of using the ready/busy bit in some embodiments).

[0258] 12. The state machine advertises availability to the downstream arbiter **920**, which picks from all pending writes.

[0259] 13. The downstream arbiter **920** then initiates a direct memory access (DMA) transfer from DRAM **918**, which goes through the CRC logic and is written into a staging queue **922** that is located in front of the encoder

[0260] 14. The ECC encoder **924** operates on the data, and writes it into a staging buffer **926**, which is banked (one set of buffers for every 4 channels in some embodiments). This banking sustains throughput to the write channels during the data transfer process at power loss.

[0261] 15. Data is then transferred to the state machine, where it gets written to the data-register in the flash.

[0262] 10. The back-end waits for the data to be persisted to flash, and then generates an interrupt to the CPU **801**. The CPU **801** reads the state machine registers to glean information about the operation (e.g. errors, etc.).

[0263] 11. The CPU **801** cleans up its scoreboard, and signals a completion back to the host **916** via the completion queue **918**.

[0264] Still referring to FIG. **9**, the host server communicates with the storage unit via a protocol overlaid on top of PCIe. The host machine contains a submission and completion queue in its memory. The non-volatile solid state storage maintains a corresponding submission queue **902** and completion queue **918** for commands. In addition, the FPGA has two DMA engines to move data between the host machine, and the non-volatile solid state storage. Software running on CPU **801** is responsible for coordinating all communication, including the DMA transfers. The non-volatile solid state storage supports two physical PCIe functions in some embodiments. The protocol runs over function-0, while function-1 is used as a "debug" interface to the non-volatile solid state storage. All registers resident in the FPGA can be read by the host over function-1. The main components of the "front-end" of the design are the protocol engine, the DMA engines, and the non volatile (NV) status engine.

[0265] The flash controller **102** is the system's gateway to the flash channels. The flash controller **102** maintains queues of commands, which are populated by the CPU **801**. The flash controller **102** is responsible for the following:

[0266] Sending commands to the flash

[0267] Monitoring the state of the flash (since some embodiments do not use the RnB pins)

[0268] Arbitration across multiple devices within a channel

[0269] Coordinating with upstream and downstream arbiters, and ensuring the transfer of data between the flash controller, and the FPGA logic

[0270] Flash controller **102** exposes a bank of 8 commands to the CPU **801** in some embodiments. These are registers that the CPU **801** can read and write via an AXI interface in some embodiments. Software running on the CPU **801** is responsible for managing these queues, and writing commands to them. The state machine maps these commands to "micro-instructions" (uops) that it then issues to a flash sequencer. In some embodiments, each flash state machine coordinates two physical flash channels.

[0271] In the case of reads, the state machine is responsible for securing the upstream bus via a req-gnt-ack handshake with the upstream arbiter. Once the bus is secured, it proceeds to read out data from the register in the flash device. One implementation of the state machine will always read out a whole page (16K) worth of data from the flash register in some embodiments. It should be appreciated that other implementations will have the ability to read selective codewords (e.g., 2K blocks) from the flash register.

[0272] In the case of writes, the state machine first initiates a DMA of data from DRAM, which in turn feeds an LDPC encode engine. Once data is available (post-encode), the downstream arbiter drives an ack back to the requestor channel. The channel then sends a write command to the flash, and establishes a conduit for data to flow from the staging FIFOs that maintain encoded data to the cache register in flash. Writes can complete once they are successfully persisted to flash. In one embodiment. the flash devices run at 100 MHz (DDR), giving us a peak bandwidth of 200 MB/s, per-flash channel.

[0273] In some embodiments, the flash controller state machine may not implement all the possible flash commands. For example, in one embodiment, commands that will be implemented are reads (hard and soft decision), writes, and erases. All other commands (such as reset, initialization sequences, setting features etc.) may be directly communicated to the sequencer by software running on the CPU **801**. This mode of operation may be referred to as the "bypass" path. Software activates this path by writing to a dedicated bypass register in one embodiment. The flash command and sequencer register address are encoded in the

command packet written to this register. The state machine then communicates the same command to the sequencer via an open core protocol in some embodiments. Data read out of the flash or the sequencer are stored in another dedicated bypass register that can be read by software running on the CPU **801**. When a command has successfully been sent to the sequencer, hardware sets a completion bit in this aforementioned register, and then allows for the next bypass command to be enqueued.

[0274] As described earlier, the state machine has a bank of 8 registers that maintain outstanding flash transactions to the channels. These entries are managed by software running on the ARM, and are mapped as registers visible on the CPUs AXI bus in some embodiments. Commands issued to the flash channels can complete out of order; it is software's responsibility to poll their completion status. Software running on the CPU **801** will maintain a scheduler that keeps track of the command-queue state in each of the flash channels. Upon receipt of a new command from the host processor, the CPU **801** will perform a conflict check to ensure that there are no pending requests to the flash controller **102** that use the same flash plane (note that there are 1024 blocks per plane, and 512 16K pages per block). This implies that software needs to do a compare on bits [32:14] of the incoming physical address against all other pending transactions. It is software's responsibility to ensure that there is a maximum of one request issued to a flash plane, at any given point of time in some embodiments.

[0275] FIG. **10** is a high level overview of the flash controller handling operation requests from multiple masters in accordance with some embodiments. The embodiments described herein provide for a scheduler state-machine that connects to a flash controller which in turn talks to multiple flash channels. Commands are programmed into this state-machine by upper-level stacks in some embodiments. The state machine can support multiple masters. As one example, three masters are provided. The three masters cover a slow maintenance path used to setup flash state, a fast hardware-accelerated path that performs latency-critical operations such as reads, writes, and erases, and a fast power-down flush path which is responsible for transferring contents of an NVRAM device to flash in the event of a power-loss. It should be appreciated that the masters may be embodied as software running on a CPU or hardware, and the term masters refers to who is driving the operations. The state-machine is responsible for arbitrating between transactions that appear on these paths and enforcing correct quality of service (QoS). Operations on the fast path are split into multiple phases in order to extract maximum performance and efficiency form the flash. Reads are split up into three phases in some embodiments. The three phases include a command phase, a poll phase that checks if the flash is ready, and finally a read out phase to transfer data. Writes are may be split into two phases in some embodiments. The two phases include a command phase, and a poll phase that checks for completion. The state-machine is responsible for splitting up these operation requests into phases, forming a sequence from the phases, keeping track of their progress across the different flash devices, and optimally scheduling the phases, sequence of phases, and/or operation requests without incurring conflicts. In addition, data transfer operations may need to arbitrate for other system resources (channels, buffers, DMA engines etc.), which is the responsibility of the state-machines. In some embodiments, there may be a need for operation of different flash chips in different modes (SLC/MLC/TLC, etc.). The scheduler can keep track of these modes, and enforce the correct signaling in accordance with the mode the flash is in. The different phases that the operation requests may be segmented/broken down/partitioned into to form a sequence is further illustrated with reference to FIGS. **11** and **12**.

[0276] Multi-master arbitration is also provided by the embodiments. One example of a situation that uses multi-master arbitration is the implementation of the read recovery logic. In the case of read failure, the system can sometimes try and recover data by changing the voltage thresholds of the flash devices. The state machine scheduler allows these maintenance operations to run concurrently with the regular fast path operations, by arbitrating between the different operations at transactional boundaries. As noted above, the master refers to the driver of the operation and may be embodied as hardware or software executing on a processor such as CPU**801**. A master may also be referred to as an agent in some embodiments.

[0277] Traditional drives typically use a combination of firmware running on a controller that outputs microcode in a format that can be consumed by the flash sequencer. All requests flow through the controller, and the controller is responsible for arbitrating between different requestors, and ensuring that commands are fully formed before being sent to the sequencer. The embodiments described herein push this boundary into the hardware in order to maximize performance, and minimize firmware's involvement. The embodiments allow for multiple masters or agents to directly access the flash sequencer via the scheduler (state machine). The scheduler takes care of buffering commands received from different masters, arbitrating between them, and pushing them to the sequencer when fully formed. This allows the mechanism to minimize the number of hops that need to be made through the controller in system configurations where there are multiple masters/agents (requestors) talking to the same flash sequencer. It should be appreciated that in addition to the "multiple master" portion, the embodiments also enable for the flash operations to be chunked into different states or phases and scheduled for maximal efficiency by the hardware (and not firmware).

[0278] FIG. **11** is an action diagram illustrating the read flow for the flash controller state machine in accordance with some embodiments. The state machine is responsible for sequencing commands out to the flash devices, and also for querying their state. When "status check" commands return from the flash devices, they raise an interrupt, and alert the state machine. The state machine is responsible for reading out the status of the interrupt register in the flash controller, and clearing it, as required. The state machine maintains a timer to track these "polling" intervals. The state machine may just maintain one global polling timer, or timers for every entry of the command queue in some embodiments. Before sending any commands to the flash controller, the state machine queries the status of the controller itself. If the controller is busy, the state machine will have to backoff, and reissue. The main steps in performing a read are as follows:

[0279] Issue the read command to the flash device

[0280] Issue a status check command to the flash device

[0281] Once the flash device is ready, arbitrate for control of the upstream FIFO write port

[0282] After winning arbitration, issue a read-from-cache-register command to the flash device

[0283] Once the transfer of data completes, relinquish control of the upstream FIFO write port

[0284] FIG. **12** is an action diagram illustrating the write flow for the flash controller state machine in accordance with some embodiments. The main steps in performing a write are as follows:

[0285] Arbitrate for control of the downstream FIFO read port

[0286] After winning arbitration, issue a write command to the flash device

[0287] Once the transfer of data completes, relinquish control of the downstream FIFO read port

[0288] Issue a status check command to the flash device

[0289] Once the flash device is done, mark the write as having been persisted

[0290] From the non-volatile solid state storage unit **152** point of view, reads complete when data gets decoded, and are sent back to the host processor. Writes and erases complete once data has been persisted/erased, respectively. Completions are indicated to software running on the CPU **801** by a bank of registers that live in the back end of the design. The back-end sets completions for writes and erases. The front-end DMA engine passes a transaction ID to the back-end to set the completion for read transactions. The back-end design maintains 4 64-bit registers, in one embodiment, that indicate the completion status for all issued transactions. When a read, write, or erase completes, the hardware will decode its transaction ID, and set a bit in the relevant position in one of the completion registers. The first register latches completions for transaction IDs 0-63, the second for IDs 64-127, and so on. When a completion register transitions from 0->1, hardware will raise an interrupt. This interrupt stays high as long as the completion register is non-zero. Software is responsible for reading out the contents of the completion register, and writing a mask value back to this register. Hardware will perform a bit-wise AND of this mask provided by software with the existing value in the register. This ensures that any completions registered in the time window between software reading and writing the register, do not get dropped.

[0291] FIG. **13**A illustrates a flash controller state machine transition diagram in accordance with some embodiments. The NVRAM region in DRAM is sized for a maximum of 2 GB in some embodiments. It should be appreciated that the exact size is configurable by software. The NVRAM flush process stripes this memory evenly across 32 channels, implying that every flash channel needs to write four to eight 8 MB blocks of data. Every state machine thus receives a maximum of 16 block-writes, which are then splits across the two flash channels it manages in some embodiments. Since each state machine has eight slots, the NVRAM dump will be performed on 128 flash blocks (16*8) at a time, with DRAM data being uniformly distributed. That is, the hardware will first dump 1 GB across 128 flash blocks, and then transition over to dumping the second GB across the next set of 128 flash blocks. If the NVRAM section is sized between 1 and 2 GB (e.g., 1.5 GB), then the second set of 128 flash blocks will be partially populated. Software has control over the number of DRAM pages in each set of flash blocks that get written to during the dump (refer to the NVRAM configuration register format).

[0292] There are n (e.g., 4) different NVRAM regions in flash that can be used to hold data on a power-loss event in some embodiments. At boot-up time, software will choose one of these regions, and configure the storage unit with the address map of the appropriate NVRAM region. The address map of each NVRAM region requires 256 (number of flash blocks)*5 Bytes (of address) worth of storage. During the initialization process, software is responsible for choosing an NVRAM region, and configuring the flush engines appropriately. The flush engines themselves are distributed, with one engine associated with each state machine. Each flush engine has a map that stores 16 destination (flash) block addresses, and 1 source (DRAM) address. The flush engine uses this programmed source DRAM base address, and keeps incrementing it by 32*16K (512K) to get subsequent DRAM source addresses that are bound to the same state machine. Each state machine only has 8 slots, and uses the first 8 destination flash block addresses to dump the first GB, and the second 8 destination addresses to dump the second GB in some embodiments. It should be appreciated that all the 8 flash blocks used in each stage of the NVRAM dump sequence use separate chip-enables in some embodiments. That is, blocks 0-7 are on 8 distinct chip-enables, as should blocks 8-15.

[0293] Referring back to FIG. **9**, there is a single DMA engine that feeds the encoder in some embodiments. The encoder output in turn gets split across eight FIFO's. Each FIFO talks to two state machines (and hence four channels). Thus, the maximum throughput to the flash channels in this embodiment is 200 MB/s*8=1.6 GB/s. In order to maximize this throughput, adjacent (16K) DRAM pages are first striped across the different FIFO's, then across the different flash channels serviced by a FIFO, and finally across different flash blocks within a flash channel. This allows the ability to get maximum row-buffer parallelism from the DRAM, and also get maximum write throughput across the flash channels. In this way, every 8th DRAM page goes to the same FIFO, every 16th page to the same flash state machine, every 32nd page goes to the same flash channel, and every 128th DRAM page gets written to the same flash block. The state machine may be programmed with the 8 flash block addresses, and the first DRAM source address accessed by that state machine.

[0294] As part of initialization, software may be responsible for "arming" the NVRAM dump sequence by first programming the source and destination addresses across all the flash state machines, and then finally writing to the NVRAM control configuration register. Both the FPGA logic and the HPS are notified during a power-loss event. If the NVRAM dump process has been armed, the backend logic will first wait for the front-end NV-status engine to push all its contents into DRAM. Additionally, the backend state machines will wait for all the state machines to quiesce before beginning the NVRAM flush process. This quiescence prevents multiple requests from simultaneously accessing the same flash planes. Once all aforementioned conditions are met, the flush engines start kicking out requests to the state machines.

[0295] Data being returned from the flash devices is passed through a LDPC decoder before it can be consumed. The FPGA has two decoders for bandwidth reasons in some embodiments. In order to fully saturate the decoders, the channels are statically partitioned to talk to one of the decoders (16 channels per decoder). Additionally, since the

LDPC decoder bandwidth exceeds channel bandwidth, each LDPC decoder will have the ability to read from two different queues. This implies that 8 channels need to arbitrate to win the write-port into their assigned FIFO. This arbitration is done by the upstream arbiter **930**. There are four arbiters in the design in some embodiments, one for every group of 8 channels. The state machines send their requests to the arbiter once data is ready in the cache register (note that there is no data buffering available in the state machines themselves). The arbiter performs an age-ordered pick, and sends a grant back to one of the requesting channels in some embodiments. Once granted, the state machine is allowed to send data out of flash into the FIFO, and cannot be interrupted until it is done with its entire data transfer (this behavior may change in other embodiments). Once the data transfer is complete, the state machine relinquishes its hold of the FIFO write port, allowing the arbiter to grant another outstanding requestor. The backend RTL is responsible for sending a packet header to the decoder (along with the start of packet (SOP)). The decoder bounces this packet header back with the data output, and adds two fields that indicate CRC pass/fail, and decoder pass/fail status.

[0296] As indicated in FIG. **9**, there are FIFO's that latch data that is output by the decoder. The DMA engine that resides in the front-end is responsible for reading out data from these FIFOs and transferring the data to the host server. In order to facilitate this, there is logic that keeps track of which code-word is being decoded (within a page), and generates a "completion packet" when the last code-word has been decoded. These completion packets are enqueued in a completion FIFO which is exposed to the front-end DMA controller.

[0297] The front-end DMA engine controller is alerted whenever there is a valid completion packet in the FIFO. The front-end DMA engine then performs the following actions:

[0298] 1. Read out the head of the completion queue **918** to get the completion packet. This completion packet will provide information about the transaction ID, and an identifier to the FIFO that contains the user data.

[0299] 2. When ready, send a read signal to the data FIFO to read out the user data. The front-end DMA engine will be responsible for maintaining a counter to make sure it dequeues the correct number of data bytes from the FIFO.

[0300] 3. Finally, the front-end is responsible for sending back the read's transaction ID to the back-end, after the DMA to the host completes. The back-end is responsible for registering this completion in the aforementioned completion vector, and raise an interrupt appropriately.

[0301] It should be appreciated that the read data from the completion FIFOs and the data FIFOs will be available one cycle after the read is asserted. Also, consecutive reads to the completion FIFO are at least four cycles apart (to allow for delays due to asynchronous crossings) in some embodiments. The front-end is expected to stop sending new read-transfer TID's (to signal completion for reads) when it sees the be_fe_rdXferFifoFull signal to be high.

[0302] Data being written to the flash devices is passed through a low density parity check (LDPC) encoder before it can be persisted in some embodiments. The FPGA has one "fixed-code" encoder which has a bandwidth of 2 GB/s in some embodiments. Since there is a significant bandwidth mismatch between the encoder and the flash channels, there will be 8 FIFOs (each 32 KB deep) that will buffer data that is to be written to flash. Each FIFO will service 4 channels (2 state machines). The encoder has one datapath, and thus requires only one DMA engine to transfer data from DRAM to the input FIFO of the encoder in this embodiment. The bandwidth of the DMA transfer is expected to be about 3 GB/s, in one embodiment, so one DMA engine is enough to fully saturate the encoder.

[0303] The arbitration of the downstream FIFO read paths, and initiation of the DMA transfers, is done by the downstream arbiter **920**. This block functions as a hierarchical arbiter—it contains—"channel arbiters" that arbitrate for the FIFO read access across four channels, and another "DMA arbiter" that arbitrates for the DMA engine access across the eight channel arbiters. The 16 state machines send their write requests to the downstream arbiter, which then farms them out to the assigned channel arbiters. The channel arbiter performs an age-ordered pick across two input state machine requests, and then assigns a winner, which in turn gets sent to the DMA arbiter in some embodiments. The DMA arbiter picks across the eight winners from the channel arbiters, performs a DMA access, and then sends an acknowledgment back to the channel arbiters. Once the LDPC encode completes, the encoder notifies the channel arbiter, which can then proceed to finally send an acknowledgment to the requesting state machine. After this, the state machine can pull data out of the FIFO, and transfer to the cache register. Once data is written to the cache register, the state machine relinquishes control of the FIFO-read port, allowing the channel arbiter to grant the next request. Refer to FIG. **8** for a graphical illustration of this path. The backend RTL is responsible for sending a packet header to the encoder (along with the SOP). The encoder bounces this packet header back with the data output, and adds a field that indicates CRC pass/fail status.

[0304] FIG. **13**B is a flow diagram of a method for scheduling in a memory controller. The method can be practiced by various embodiments of memory controllers described herein, such as a controller of a storage unit, or the flash controller. In an action **1302**, operation requests are received from masters. In an action **1304**, the operation requests are arbitrated, according to policies. These policies are configurable in some embodiments. In an action **1306**, operation phases are formed from the operation requests. The operation phases may be formed as described above with reference to FIGS. **9**-**13**A. In an action **1308**, the operation phases are arbitrated according to the policies. The operation phases may be arbitrated as described above with reference to FIGS. **9**-**13**A and utilize any know arbitration mechanism. The operation phases are communicated through channels to solid-state memory devices according to the arbitrating, in an action **1310**. The operation phases are tracked with a scoreboard, in an action **1312** in some embodiments.

[0305] It should be appreciated that the methods described herein may be performed with a digital processing system, such as a conventional, general-purpose computer system. Special purpose computers, which are designed or programmed to perform only one function may be used in the alternative. FIG. **14** is an illustration showing an exemplary computing device which may implement the embodiments

described herein. The computing device **1400** of FIG. **14** may be used to perform embodiments of the functionality for an external processor (i.e., external to the flash controller), CPU **801** (internal to the flash controller) or the multithreaded/virtualized microcode sequence engine (internal to the flash controller) in accordance with some embodiments. The computing device **1400** includes a central processing unit (CPU) **1401**, which is coupled through a bus **1405** to a memory **1403**, and mass storage device **1407**. Mass storage device **1407** represents a persistent data storage device such as a disc drive, which may be local or remote in some embodiments. The mass storage device **1407** could implement a backup storage, in some embodiments. Memory **1403** may include read only memory, random access memory, etc. Applications resident on the computing device may be stored on or accessed via a computer readable medium such as memory **1403** or mass storage device **1407** in some embodiments. Applications may also be in the form of modulated electronic signals modulated accessed via a network modem or other network interface of the computing device. It should be appreciated that CPU **1401** may be embodied in a general-purpose processor, a special purpose processor, or a specially programmed logic device in some embodiments.

[0306] Display **1411** is in communication with CPU **1401**, memory **1403**, and mass storage device **1407**, through bus **1405**. Display **1411** is configured to display any visualization tools or reports associated with the system described herein. Input/output device **1409** is coupled to bus **1405** in order to communicate information in command selections to CPU **1401**. It should be appreciated that data to and from external devices may be communicated through the input/output device **1409**. CPU **1401** can be defined to execute the functionality described herein to enable the functionality described with reference to FIGS. **1-13**B. The code embodying this functionality may be stored within memory **1403** or mass storage device **1407** for execution by a processor such as CPU **1401** in some embodiments. The operating system on the computing device may be MS-WINDOWS™, UNIX™, LINUX™, iOS™, CentOS™, Android™, Redhat Linux™, z/OS™, or other known operating systems. It should be appreciated that the embodiments described herein may also be integrated with a virtualized computing system implemented with physical computing resources.

[0307] FIG. **15** illustrates a memory controller **1502** with a scheduling system **1508** that prioritizes operations in accordance with present embodiments. Policies **1512** according to which the scheduling system **1508** operates can be set to solve system and component-specific problems and/or provide improvements in storage system technology, for example as discussed below.

[0308] In SSDs, the read and write latencies for QLC are relatively high as compared to SLC, MLC, and TLC. This difference can lead to die contention at the LUNs as other, faster operations (e.g., SLC reads/writes) have to wait until the QLC access operations have completed, causing large increases in latency. In a distributed storage system such as a system with reference to FIGS. **2**A-G this issue becomes even more of a problem. For example, in the distributed system of FIGS. **2**A-G there may be 128 authorities making their own scheduling decisions independent of each other. In addition, as flash chips continue to get larger, the number of LUNs or dies available per drive are shrinking, decreasing available entropy. While some vendors may provide "suspend-resume" functionality which can pause an existing operation, e.g., a write or erase, and allow a read to proceed. However, this functionality has higher overhead than the intelligent scheduling described herein. In addition, the "suspend-resume" functionality cannot allow an SLC write to pause an older QLC write and take precedence as enabled through the embodiments described herein.

[0309] Embodiments of storage systems, memory controllers and scheduling systems described with reference to FIGS. **15-18** involve a die operation queue being maintained by the host for each of the dies of a storage device. The die operation queue implements policies that limit the number of operations scheduled on each die and prioritize the different operations to reduce die contention, in various embodiments.

[0310] The operation queue limits the number of concurrent reads and writes sent to a die. For example, the policy would limit the maximum number of current reads for a die at **3** and the maximum number of writes per die at **1** in one exemplary embodiment.

[0311] The operation queue may also prioritize operations performed by the die as follows in the list of priorities/policies, which are one example and not meant to be limiting as numerous priorities/policies may be incorporated with the embodiments:

[0312] Differentiate frontend and backend read and write

[0313] Frontend is prioritized over backend

[0314] Frontend read is prioritized over frontend write

[0315] SLC writes (both frontend and backend) are prioritized over QLC writes

[0316] As an example with the above list in mind, if an operation queue for a die includes a frontend write and a frontend read is received, the frontend read would be inserted into the operation queue ahead of the frontend write so that the frontend read is performed first, i.e., has priority over the frontend write. In some embodiments, these examples can be broadened out to be based on latencies that are not limited solely to SLC and QLC flash memory, as further described below. It should be appreciated that a frontend operation may be any operation that is in the critical path of data I/O, i.e., an operation that should be given priority for immediate or rapid execution so as to contribute to data throughput and minimize data access latency. A backend operation may be any operation that is not in the critical path of data I/O, and can be executed at any time or scheduled for later execution in the background, such as garbage collection, without (or at most, minimally) interfering with data throughput and data access latency. Categorizing a specific operation as frontend or backend may be used in prioritizing that operation relative to other operations.

[0317] Continuing with reference to FIG. **15**, a processing device **1504**, which could be one or more processors, is available in the memory controller **1502** and/or in the scheduling system **1508** for execution of software or firmware of various modules and/or cooperation with modules that may be implemented in hardware, firmware, software or combinations thereof. The scheduling system **1508** includes a scheduler **1510** that receives operation request(s) and acts according to one or more policies **1512**. The scheduler **1510** performs a prioritizing action **1520**, prioritizing each operation request, and performs an insert action **1522**, inserting each operation request as an operation in an appropriate one of the operation queues **1514**. It should be appreciated that

the prioritizing (or prioritization) and insertion of each operation request into an operation queue **1514** is according to the policies **1512**. The operation queues **1514** are arranged one to one with the memory dies **1518**, i.e., each memory die **1518** has an operation queue **1514** associated with the corresponding memory die in a one to one relationship in some embodiments, and vice versa. Such implementation may involve, for example, allocating a number of operation queues **1514** equal to the number of memory dies **1518** in the portion of storage memory **1516** operated upon by the scheduler **1510** and scheduling system **1508**, or having spare operation queues **1514** that go unused, etc. It should also be appreciated that the memory controller **1502** and storage memory **1516** with memory dies **1518** may be a part of a larger, distributed storage system with multiple such memory controllers **1502** and associated portions of storage memory **1516**, as referenced above in FIGS. **1-5**.

[0318] FIG. **16** illustrates an operation queue **1514**, with operations **1602** that is suitable for embodiments of the scheduling system **1508** of FIG. **15**. In various embodiments, the operation queue **1514** is implemented as a hardware queue or a software queue with a specified number of queue locations in the queue, for operations **1602**. Various actions can be performed on or by the operation queue **1514**, for example using parallel input and sequential output, in some embodiments. One embodiment of the operation queue is content addressable, so that the scheduler **1510** can determine which operations **1602**, or characteristics of the operations **1602**, are in the operation queue **1514** for purposes of prioritizing operation requests **1506** and operations **1602**. An action **1604** to insert at tail inserts an operation **1602** at the tail of the operation queue **1514**. An action **1606** to insert at head inserts an operation **1602** at the head of the operation queue **1514**. The action **1608** of pop off operation at head of operation queue causes the operation queue **1514** to pop or select an operation **1602** off at the head of the operation queue **1514**, whereupon the operation queue **1514** points to the next operation **1602** (if present in the operation queue **1514** and therefore available) as being at the head of the operation queue **1514**. The action **1610** of execute operation causes the operation that has just been popped off from the head of the operation queue **1514** to be executed, for example an operation to write to, read from, or erase a portion of the memory die **1518** that is coupled to that operation queue **1514**. A further embodiment of the operation queue **1514** supports insertion of an operation **1602** at any location in the operation queue **1514**, not just the tail or the head.

[0319] FIG. **17** illustrates example policies **1512** that are suitable for the scheduling system of FIG. **15**. Fewer, further, and other policies suitable for various embodiments of storage systems are readily devised in keeping with the teachings herein as the example provided by FIG. **17** is not meant to be limiting. Policies may trade off latency or throughput for one type of data, operation or storage memory in comparison to another in some embodiments. For example, because user data I/O throughput and latency are important, operations relating to user data I/O may be made frontend operations and prioritized over garbage collection operations, which may be made backend operations in these embodiments. To minimize read latency and take advantage of low latency for write caching, read operations may be made frontend operations and prioritized over write operations in other embodiments.

[0320] One policy **1702** is to limit the number of concurrent reads or writes for memory die. For example, the number of concurrent reads could be limited to two reads, or three reads, and the number of concurrent writes could be limited to one in one embodiment but this is not meant to be limiting as numerous combinations/permutations of concurrent reads and concurrent writes are possible for integration with the embodiments contained herein.

[0321] Policy **1704** prioritizes frontend operations over backend operations, while policy **1706** prioritizes a frontend read over a backend read. Another possible policy **1708** is that garbage collection (GC) operations are designated as backend operations. Other operations could be declared backend operations, and various operations (e.g., relating to user data I/O) could be declared frontend operations, in various policies.

[0322] Policy **1710** prioritizes frontend read over frontend write, while policy **1712** prioritizes a frontend write over a garbage collection read and/or a garbage collection write. Policy **1714** holds off or delays queuing a second write if a first write or one or more read is in the queue. As mentioned previously further policies prioritizing the operations so as to manage die contention may be incorporated into the embodiments as the list is not exhaustive.

[0323] FIG. **18** is a flow diagram of a method that is practiced on and by embodiments of the scheduling system and memory controller of FIG. **15**. The method may be practiced by a processing device, e.g., one or more processors, or distributed processing, in a storage system, memory controller or scheduling system.

[0324] In an action **1802**, a scheduling system receives one or more operation requests. Each operation request is for an operation involving storage memory. Various operations include reading and writing operations as well as garbage collection operations.

[0325] In an action **1804**, the scheduling system prioritizes each operation request according to one or more policies. In some embodiments, the prioritization may be achieved through placement into a queue according to a set of policies.

[0326] In an action **1806**, the scheduling system inserts the operation request, as an operation, in an operation queue. The operation queue is coupled to a memory die, on a one-to-one basis, i.e., one operation queue to one memory die. As discussed with reference to FIG. **16** the operation queue is content addressable to assist in determining which operations are to be prioritized.

[0327] Operations involving a memory die are popped off the head of the operation queue and executed, as prioritized according to the policy or policies. Multiple operation queues, each coupled to an associated memory die, are operating in parallel in the storage system.

[0328] Detailed illustrative embodiments are disclosed herein. However, specific functional details disclosed herein are merely representative for purposes of describing embodiments. Embodiments may, however, be embodied in many alternate forms and should not be construed as limited to only the embodiments set forth herein.

[0329] It should be understood that although the terms first, second, etc. may be used herein to describe various steps or calculations, these steps or calculations should not be limited by these terms. These terms are only used to distinguish one step or calculation from another. For example, a first calculation could be termed a second cal-

culation, and, similarly, a second step could be termed a first step, without departing from the scope of this disclosure. As used herein, the term "and/or" and the "/" symbol includes any and all combinations of one or more of the associated listed items.

[0330] As used herein, the singular forms "a", "an" and "the" are intended to include the plural forms as well, unless the context clearly indicates otherwise. It will be further understood that the terms "comprises", "comprising", "includes", and/or "including", when used herein, specify the presence of stated features, integers, steps, operations, elements, and/or components, but do not preclude the presence or addition of one or more other features, integers, steps, operations, elements, components, and/or groups thereof. Therefore, the terminology used herein is for the purpose of describing particular embodiments only and is not intended to be limiting.

[0331] It should also be noted that in some alternative implementations, the functions/acts noted may occur out of the order noted in the figures. For example, two figures shown in succession may in fact be executed substantially concurrently or may sometimes be executed in the reverse order, depending upon the functionality/acts involved.

[0332] With the above embodiments in mind, it should be understood that the embodiments might employ various computer-implemented operations involving data stored in computer systems. These operations are those requiring physical manipulation of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. Further, the manipulations performed are often referred to in terms, such as producing, identifying, determining, or comparing. Any of the operations described herein that form part of the embodiments are useful machine operations. The embodiments also relate to a device or an apparatus for performing these operations. The apparatus can be specially constructed for the required purpose, or the apparatus can be a general-purpose computer selectively activated or configured by a computer program stored in the computer. In particular, various general-purpose machines can be used with computer programs written in accordance with the teachings herein, or it may be more convenient to construct a more specialized apparatus to perform the required operations.

[0333] A module, an application, a layer, an agent or other method-operable entity could be implemented as hardware, firmware, or a processor executing software, or combinations thereof. It should be appreciated that, where a software-based embodiment is disclosed herein, the software can be embodied in a physical machine such as a controller. For example, a controller could include a first module and a second module. A controller could be configured to perform various actions, e.g., of a method, an application, a layer or an agent.

[0334] The embodiments can also be embodied as computer readable code on a tangible non-transitory computer readable medium. The computer readable medium is any data storage device that can store data, which can be thereafter read by a computer system. Examples of the computer readable medium include hard drives, network attached storage (NAS), read-only memory, random-access memory, CD-ROMs, CD-Rs, CD-RWs, magnetic tapes, and other optical and non-optical data storage devices. The computer readable medium can also be distributed over a network coupled computer system so that the computer readable code is stored and executed in a distributed fashion. Embodiments described herein may be practiced with various computer system configurations including hand-held devices, tablets, microprocessor systems, microprocessor-based or programmable consumer electronics, minicomputers, mainframe computers and the like. The embodiments can also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a wire-based or wireless network.

[0335] Although the method operations were described in a specific order, it should be understood that other operations may be performed in between described operations, described operations may be adjusted so that they occur at slightly different times or the described operations may be distributed in a system which allows the occurrence of the processing operations at various intervals associated with the processing.

[0336] In various embodiments, one or more portions of the methods and mechanisms described herein may form part of a cloud-computing environment. In such embodiments, resources may be provided over the Internet as services according to one or more various models. Such models may include Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). In IaaS, computer infrastructure is delivered as a service. In such a case, the computing equipment is generally owned and operated by the service provider. In the PaaS model, software tools and underlying equipment used by developers to develop software solutions may be provided as a service and hosted by the service provider. SaaS typically includes a service provider licensing software as a service on demand. The service provider may host the software, or may deploy the software to a customer for a given period of time. Numerous combinations of the above models are possible and are contemplated.

[0337] Various units, circuits, or other components may be described or claimed as "configured to" perform a task or tasks. In such contexts, the phrase "configured to" is used to connote structure by indicating that the units/circuits/components include structure (e.g., circuitry) that performs the task or tasks during operation. As such, the unit/circuit/component can be said to be configured to perform the task even when the specified unit/circuit/component is not currently operational (e.g., is not on). The units/circuits/components used with the "configured to" language include hardware—for example, circuits, memory storing program instructions executable to implement the operation, etc. Reciting that a unit/circuit/component is "configured to" perform one or more tasks is expressly intended not to invoke 35 U.S.C. 112, sixth paragraph, for that unit/circuit/component. Additionally, "configured to" can include generic structure (e.g., generic circuitry) that is manipulated by software and/or firmware (e.g., an FPGA or a general-purpose processor executing software) to operate in manner that is capable of performing the task(s) at issue. "Configured to" may also include adapting a manufacturing process (e.g., a semiconductor fabrication facility) to fabricate devices (e.g., integrated circuits) that are adapted to implement or perform one or more tasks.

[0338] The foregoing description, for the purpose of explanation, has been described with reference to specific embodiments. However, the illustrative discussions above

are not intended to be exhaustive or to limit the invention to the precise forms disclosed. Many modifications and variations are possible in view of the above teachings. The embodiments were chosen and described in order to best explain the principles of the embodiments and its practical applications, to thereby enable others skilled in the art to best utilize the embodiments and various modifications as may be suited to the particular use contemplated. Accordingly, the present embodiments are to be considered as illustrative and not restrictive, and the invention is not to be limited to the details given herein, but may be modified within the scope and equivalents of the appended claims.

What is claimed is:

1. A scheduling system, comprising:

a plurality of operation queues arranged to couple to a plurality of memory dies of storage memory wherein a relationship of operation queue to memory die is a one to one relationship; and

a scheduler arranged to:

receive a plurality of requests for the storage memory;

prioritize each request according to one or more policies that limit maximum number of concurrent writes and concurrent reads for a memory die and prioritize operations to be performed relating to the memory die; and

insert each request as an operation into one of the plurality of operation queues, with priority relative to one or more other operation in the one of the plurality of operation queues according to the one or more policies.

2. The scheduling system of claim **1**, wherein the scheduler is configured to differentiate frontend reads and writes and backend reads and writes.

3. The scheduling system of claim **1**, wherein the scheduler is configured to prioritize frontend operations over backend operations.

4. The scheduling system of claim **1**, wherein the scheduler is configured to prioritize a frontend read over a frontend write.

5. The scheduling system of claim **1**, wherein the scheduler is configured to prioritize a frontend write over a garbage collection read and a garbage collection write.

6. The scheduling system of claim **1**, wherein to prioritize a frontend read over a frontend write in an operation queue, the scheduler is to insert the frontend read ahead of the frontend write in the operation queue.

7. The scheduling system of claim **1**, wherein the scheduler is configured to limit the maximum number of concurrent reads to three for the memory die and the maximum number of concurrent writes to one for the memory die.

8. The scheduling system of claim **1**, wherein the scheduler is configured to delay queuing a second write in an operation queue when there is one of a first write and more than one read, for the memory die, in the one of the plurality of operation queues.

9. The scheduling system of claim **1**, wherein the scheduler is configured to insert a frontend write ahead of a garbage collection write in an operation queue, responsive to receiving a request for the frontend write when there is the garbage collection write in the operation queue.

10. A method, comprising:

receiving a plurality of requests for a memory die of a storage memory;

prioritizing each request according to one or more policies that limit maximum number of concurrent writes and concurrent reads for the memory die and prioritize operations to be performed relating to the memory die; and

inserting each request as an operation into an operation queue, with priority relative to one or more other operations in the operation queue according to the one or more policies.

11. The method of claim **10**, wherein the one or more policies prioritize frontend operations over backend operations.

12. The method of claim **10**, wherein:

the one or more policies prioritize a frontend read over a frontend write; and

the one or more policies prioritize a frontend write over a garbage collection read and a garbage collection write.

13. The method of claim **10**, wherein the one or more policies limit the maximum number of concurrent reads to three for the memory die and the maximum number of concurrent writes to one for the memory die.

14. The method of claim **10**, further comprising delaying queuing a second write in the operation queue responsive to having one of a first write and more than one read, for the memory die, in the operation queue.

15. The method of claim **10**, further comprising inserting a frontend write ahead of a garbage collection write in the operation queue, responsive to receiving a request for the frontend write and having the garbage collection write in the operation queue.

16. A tangible, non-transitory, computer-readable media having instructions thereupon which, when executed by a processor, cause the processor to perform a method comprising:

receiving a plurality of requests for one or more memory dies of a storage memory;

prioritizing each request according to one or more policies that limit maximum number of concurrent writes and concurrent reads for a memory die and prioritize operations to be performed relating to the memory die; and

inserting each request as an operation into one of a plurality of operation queues, with priority relative to one or more other operation in the operation queue according to the one or more policies.

17. The computer-readable media of claim **16**, wherein prioritizing comprises:

prioritizing a frontend write over a garbage collection read and a garbage collection write; and

prioritizing a frontend read over the frontend write, the garbage collection read and the garbage collection write.

18. The computer-readable media of claim **16**, wherein the inserting comprises inserting a frontend read ahead of a frontend write in the operation queue to prioritize the frontend read over the frontend write.

19. The computer-readable media of claim **16**, further comprising delaying queuing a second write in the operation queue when there is one of a first write and more than one read, for the memory die, in the operation queue.

20. The computer-readable media of claim **16**, wherein the inserting comprises inserting a frontend write ahead of a

garbage collection write in the operation queue, responsive to having the garbage collection write in the operation queue.

* * * * *