



US012393485B2

(12) **United States Patent**
Fay et al.(10) **Patent No.:** US 12,393,485 B2
(45) **Date of Patent:** Aug. 19, 2025(54) **RECOVER CORRUPTED DATA THROUGH SPECULATIVE BITFLIP AND CROSS-VALIDATION**(71) Applicant: **PURE STORAGE, INC.**, Mountain View, CA (US)(72) Inventors: **Matthew Fay**, Mountain View, CA (US); **Runmin Wang**, Santa Clara, CA (US); **Randy Zhao**, Fremont, CA (US); **Andrew Bernat**, Mountain View, CA (US)(73) Assignee: **PURE STORAGE, INC.**, Santa Clara, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 71 days.

(21) Appl. No.: **18/054,246**(22) Filed: **Nov. 10, 2022**(65) **Prior Publication Data**

US 2023/0244569 A1 Aug. 3, 2023

Related U.S. Application Data

(63) Continuation-in-part of application No. 17/587,514, filed on Jan. 28, 2022.

(51) **Int. Cl.**
G06F 11/08 (2006.01)
G06F 11/10 (2006.01)(52) **U.S. Cl.**
CPC **G06F 11/1096** (2013.01); **G06F 11/1088** (2013.01); **G06F 11/108** (2013.01)(58) **Field of Classification Search**
CPC G06F 11/1096; G06F 11/108; G06F 11/1088; G06F 2201/84; G06F 3/064;
(Continued)(56) **References Cited**

U.S. PATENT DOCUMENTS

5,572,661 A 11/1996 Jacobson
5,706,210 A 1/1998 Kumano et al.
(Continued)

FOREIGN PATENT DOCUMENTS

EP 0725324 A2 8/1996
EP 3657828 A1 5/2020
(Continued)

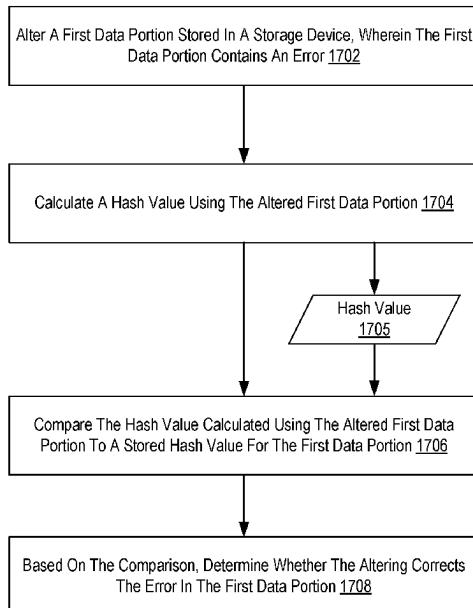
OTHER PUBLICATIONS

Bellamy-McIntyre et al., "OpenID and the Enterprise: A Model-based Analysis of Single Sign-On Authentication", 15th IEEE International Enterprise Distributed Object Computing Conference (EDOC), Aug. 29, 2011, pp. 129-138, IEEE Computer Society, USA, DOI: 10.1109/EDOC.2011.26, ISBN: 978-1-4577-0362-1.
(Continued)

Primary Examiner — Tyler J Torgrimson

(57) **ABSTRACT**

Recovering corrupted data through speculative bitflip and cross-validation, including altering a first data portion stored in a storage device, wherein the first data portion contains an error; calculating a hash value using the altered first data portion; comparing the hash value calculated using the altered first data portion to a stored hash value for the first data portion; and based on the comparing, determining whether the altering corrects the error in the first data portion.

20 Claims, 35 Drawing Sheets

(58) **Field of Classification Search**

CPC G06F 3/065; G06F 3/067; G06F 3/0689;
 G06F 11/1435; G06F 11/2094; G06F
 11/2097; G06F 3/0619

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,799,200 A	8/1998	Brant et al.	8,812,860 B1	8/2014	Bray
5,826,001 A	10/1998	Lubbers et al.	8,822,155 B2	9/2014	Sukumar et al.
5,913,927 A	6/1999	Nagaraj et al.	8,850,546 B1	9/2014	Field et al.
5,933,598 A	8/1999	Scales et al.	8,898,346 B1	11/2014	Simmons
6,012,032 A	1/2000	Donovan et al.	8,909,854 B2	12/2014	Yamagishi et al.
6,085,333 A	7/2000	Dekoning et al.	8,931,041 B1	1/2015	Banerjee
6,161,192 A	12/2000	Lubbers et al.	8,949,863 B1	2/2015	Coatney et al.
RE37,364 E	9/2001	Cohn et al.	8,954,649 B2	2/2015	Biswas et al.
6,295,541 B1	9/2001	Bodnar et al.	8,954,670 B1	2/2015	Subramanian et al.
RE37,601 E	3/2002	Eastridge et al.	8,959,280 B2	2/2015	Yu et al.
6,643,641 B1	11/2003	Snyder	8,984,602 B1	3/2015	Bailey et al.
6,647,514 B1	11/2003	Umberger et al.	8,990,905 B1	3/2015	Bailey et al.
6,789,162 B1	9/2004	Talagala et al.	9,081,713 B1	7/2015	Bennett
6,944,791 B2	9/2005	Humlcek et al.	9,124,569 B2	9/2015	Hussain et al.
6,959,413 B2	10/2005	Humlcek et al.	9,134,922 B2	9/2015	Rajagopal et al.
7,007,193 B1	2/2006	Fuld	9,135,300 B1	9/2015	Yang et al.
7,089,272 B1	8/2006	Garthwaite et al.	9,189,334 B2	11/2015	Bennett
7,107,389 B2	9/2006	Inagaki et al.	9,209,973 B2	12/2015	Aikas et al.
7,146,521 B1	12/2006	Nguyen	9,250,823 B1	2/2016	Kamat et al.
7,334,124 B2	2/2008	Pham et al.	9,280,678 B2	3/2016	Redberg
7,437,530 B1	10/2008	Rajan	9,300,660 B1	3/2016	Borowiec et al.
7,464,289 B2	12/2008	Hung	9,311,182 B2	4/2016	Bennett
7,493,424 B1	2/2009	Bali et al.	9,323,775 B2	4/2016	Srivastav et al.
7,523,257 B2	4/2009	Horn et al.	9,378,210 B2	6/2016	Raanan et al.
7,577,783 B2	8/2009	Hsu et al.	9,395,922 B2	7/2016	Nishikido et al.
7,669,029 B1	2/2010	Mishra et al.	9,442,814 B2	9/2016	Chatterjee et al.
7,680,983 B2	3/2010	Sutoh et al.	9,444,822 B1	9/2016	Borowiec et al.
7,689,609 B2	3/2010	Lango et al.	9,449,040 B2	9/2016	Gupta
7,743,191 B1	6/2010	Liao	9,501,356 B2	11/2016	Palmer et al.
7,783,604 B1	8/2010	Yueh	9,507,532 B1	11/2016	Colgrove et al.
7,865,475 B1	1/2011	Yadav et al.	9,547,589 B2	1/2017	Yu et al.
7,890,538 B2	2/2011	Roden	9,569,460 B1	2/2017	Yadav et al.
7,899,796 B2	2/2011	Pawar et al.	9,626,367 B1	4/2017	Kokotovich et al.
7,899,780 B1	3/2011	Shmulyovich et al.	9,632,715 B2	4/2017	Palmer et al.
7,975,115 B2	7/2011	Wayda et al.	9,632,870 B2	4/2017	Bennett
7,979,742 B2	7/2011	Uysal et al.	9,646,024 B2	5/2017	Srivastav et al.
7,991,942 B2	8/2011	Biswas et al.	9,659,026 B2	5/2017	Leverett et al.
8,042,163 B1	10/2011	Karr et al.	9,720,620 B1	8/2017	Wei et al.
8,086,583 B1	12/2011	Brashers et al.	9,807,164 B2	10/2017	Wareing
8,099,571 B1	1/2012	Driscoll et al.	9,870,165 B2	1/2018	Palmer et al.
8,156,392 B2	4/2012	Flynn et al.	9,875,160 B2	1/2018	Bezarbarua et al.
8,200,887 B2	6/2012	Bennett	9,916,203 B1	3/2018	Pogde et al.
8,239,713 B2	8/2012	Borchers et al.	9,928,248 B2	3/2018	Akirav et al.
8,239,714 B2	8/2012	Flynn et al.	10,061,834 B1	8/2018	Kulesza et al.
8,271,700 B1	9/2012	Annen et al.	10,067,032 B2	9/2018	Palmer et al.
8,280,858 B2	10/2012	Ahrens et al.	10,114,878 B2	10/2018	Bhide et al.
8,301,942 B2	10/2012	Galbraith et al.	10,176,217 B1	1/2019	Dang et al.
8,321,380 B1	11/2012	Leverett	10,235,065 B1	3/2019	Miller et al.
8,352,422 B2	1/2013	Prahlad et al.	10,275,176 B1	4/2019	Gold et al.
8,386,834 B1	2/2013	Goel et al.	10,324,639 B2	6/2019	Seo
8,387,136 B2	2/2013	Lee et al.	10,340,025 B2	7/2019	Liu et al.
8,417,987 B1	4/2013	Goel et al.	10,360,214 B2	7/2019	Gold et al.
8,437,189 B1	5/2013	Montierth et al.	10,365,982 B1	7/2019	Brooks et al.
8,465,332 B2	6/2013	Hogan et al.	10,452,444 B1	10/2019	Jibaja et al.
8,495,469 B2	7/2013	Bakke et al.	10,466,913 B2	11/2019	Bonwick
8,503,750 B2	8/2013	Benson et al.	10,467,107 B1	11/2019	Abrol et al.
8,504,797 B2	8/2013	Mimatsu	10,503,427 B2	12/2019	Botes et al.
8,527,544 B1	9/2013	Colgrove et al.	10,509,581 B1	12/2019	Abrol et al.
8,566,546 B1	10/2013	Marshak et al.	10,545,987 B2	1/2020	Borowiec et al.
8,578,442 B1	11/2013	Banerjee	10,567,406 B2	2/2020	Astigarraga et al.
8,613,066 B1	12/2013	Brezinski et al.	10,585,851 B2	3/2020	Zhong et al.
8,620,970 B2	12/2013	English et al.	10,621,050 B2	4/2020	Horowitz et al.
8,677,486 B2	3/2014	Olney et al.	10,621,102 B2	4/2020	Cherubini et al.
8,732,519 B2	5/2014	Sun et al.	10,664,316 B2	5/2020	Murugan et al.
8,751,463 B1	6/2014	Chamness	10,671,434 B1	6/2020	Gold et al.
8,762,642 B2	6/2014	Bates et al.	10,671,435 B1	6/2020	Gold et al.
8,769,622 B2	7/2014	Chang et al.	10,671,494 B1	6/2020	Abrol et al.
8,800,009 B1	8/2014	Beda et al.	10,672,497 B2	6/2020	Cai et al.
			10,685,081 B2	6/2020	Chattopadhyay et al.
			10,817,392 B1	10/2020	McAuliffe et al.
			10,833,848 B1	11/2020	Zhuo
			10,846,137 B2	11/2020	Vallala et al.
			10,853,194 B1	12/2020	Certain et al.
			10,877,683 B2	12/2020	Wu et al.
			10,884,993 B1	1/2021	Grunwald et al.
			10,901,845 B2	1/2021	Nightingale
			10,983,705 B2	4/2021	Chawla et al.
			10,990,490 B1	4/2021	Brooks et al.
			10,992,751 B1	4/2021	Cudak et al.

US 12,393,485 B2

Page 3

(56)

References Cited

U.S. PATENT DOCUMENTS

10,996,898 B2	5/2021	Meiri et al.	2010/0262868 A1	10/2010	Galbraith et al.
11,010,089 B2	5/2021	Seguy et al.	2010/0274763 A1	10/2010	Ingen et al.
11,025,410 B2	6/2021	Zhuo	2010/0306500 A1	12/2010	Mimatsu
11,036,677 B1	6/2021	Grunwald et al.	2010/0325345 A1	12/2010	Ohno et al.
11,049,585 B1	6/2021	Hung et al.	2010/0332446 A1	12/2010	Ahrens et al.
11,061,786 B1	7/2021	Miller et al.	2010/0332754 A1	12/2010	Lai et al.
11,075,745 B1	7/2021	Zhuo	2011/0035540 A1	2/2011	Fitzgerald et al.
11,076,509 B2	7/2021	Alissa et al.	2011/0072290 A1	3/2011	Davis et al.
11,086,555 B1	8/2021	Botes et al.	2011/0119429 A1	5/2011	Tu et al.
11,089,105 B1	8/2021	Karumbunathan et al.	2011/0125955 A1	5/2011	Chen
11,093,519 B2	8/2021	Venkatesan et al.	2011/0131231 A1	6/2011	Haas et al.
11,106,810 B2	8/2021	Natanzon et al.	2011/0153944 A1	6/2011	Kursawe
11,194,707 B2	12/2021	Stalzer	2011/0167221 A1	7/2011	Pangal et al.
2002/0013802 A1	1/2002	Mori et al.	2011/0246430 A1	10/2011	Prahlad et al.
2003/0028716 A1	2/2003	Sved	2011/0258702 A1	10/2011	Olney et al.
2003/0145172 A1	7/2003	Galbraith et al.	2011/0289266 A1	11/2011	Biswas et al.
2003/0191783 A1	10/2003	Wolczko et al.	2011/0302446 A1*	12/2011	Becker-Szencyd
2003/0208511 A1	11/2003	Earl et al.			G06F 11/1076
2003/0225961 A1	12/2003	Chow et al.			714/6.1
2003/0233611 A1	12/2003	Humlcek et al.	2011/0313973 A1	12/2011	Srivast et al.
2004/0015657 A1	1/2004	Humlcek et al.	2012/0023144 A1	1/2012	Rub
2004/00080985 A1	4/2004	Chang et al.	2012/0054264 A1	3/2012	Haugh et al.
2004/0111573 A1	6/2004	Garthwaite	2012/0060054 A1	3/2012	Sun et al.
2004/0128582 A1	7/2004	Chou	2012/0079318 A1	3/2012	Colgrove et al.
2004/0153844 A1	8/2004	Ghose et al.	2012/0084611 A1	4/2012	Flynn et al.
2004/0193814 A1	9/2004	Erickson et al.	2012/0101991 A1	4/2012	Srivast et al.
2004/0260967 A1	12/2004	Guha et al.	2012/0131253 A1	5/2012	McKnight et al.
2005/0050383 A1*	3/2005	Horn	2012/0284587 A1	11/2012	Yu et al.
		G06F 11/2087	2012/0297258 A1	11/2012	Flynn et al.
		714/6.22	2012/0303919 A1	11/2012	Hu et al.
			2012/0311000 A1	12/2012	Post et al.
			2013/0007845 A1	1/2013	Chang et al.
			2013/0019049 A1	1/2013	Yeh
2005/0160416 A1	7/2005	Jamison et al.	2013/0031414 A1	1/2013	Dhuse et al.
2005/0188246 A1	8/2005	Emberty et al.	2013/0036272 A1	2/2013	Nelson
2005/0216790 A1	9/2005	Ohran	2013/0071087 A1	3/2013	Motiwala et al.
2005/0216800 A1	9/2005	Bicknell et al.	2013/0145085 A1	6/2013	Yu et al.
2006/0015771 A1	1/2006	Van Gundy et al.	2013/0145447 A1	6/2013	Maron
2006/0064444 A1	3/2006	Ingen et al.	2013/0166991 A1	6/2013	Shimada
2006/0129817 A1	6/2006	Borneman et al.	2013/0191555 A1	7/2013	Liu
2006/0149999 A1	7/2006	Hung	2013/0198459 A1	8/2013	Joshi et al.
2006/0161726 A1	7/2006	Lasser	2013/0205173 A1	8/2013	Yoneda
2006/0230245 A1	10/2006	Gounares et al.	2013/0219164 A1	8/2013	Hamid
2006/0239075 A1	10/2006	Williams et al.	2013/0227201 A1	8/2013	Talagala et al.
2006/0250720 A1	11/2006	Hsu et al.	2013/0262803 A1	10/2013	Satoyama et al.
2006/0259812 A1	11/2006	Chen	2013/0290607 A1	10/2013	Chang et al.
2007/0022227 A1	1/2007	Miki	2013/0311434 A1	11/2013	Jones
2007/0028068 A1	2/2007	Golding et al.	2013/0318297 A1	11/2013	Jibbe et al.
2007/00338682 A1	2/2007	Kulesza et al.	2013/0332614 A1	12/2013	Brunk et al.
2007/0055702 A1	3/2007	Fridella et al.	2014/0006462 A1	1/2014	Raanan et al.
2007/0081401 A1	4/2007	Chen	2014/0020083 A1	1/2014	Fetik
2007/0109856 A1	5/2007	Pellicone et al.	2014/0074850 A1	3/2014	Noel et al.
2007/0150689 A1	6/2007	Pandit et al.	2014/0082715 A1	3/2014	Grajek et al.
2007/0168321 A1	7/2007	Saito et al.	2014/0086146 A1	3/2014	Kim et al.
2007/0220227 A1	9/2007	Long	2014/0090009 A1	3/2014	Li et al.
2007/0294563 A1	12/2007	Bose	2014/0096220 A1	4/2014	Pinto et al.
2007/0294564 A1	12/2007	Reddin et al.	2014/0101434 A1	4/2014	Senthurpandi et al.
2008/0005587 A1	1/2008	Ahlquist	2014/0149357 A1	5/2014	Gupta
2008/0028144 A1	1/2008	Sutoh et al.	2014/0164774 A1	6/2014	Nord et al.
2008/0071980 A1	3/2008	Elliott et al.	2014/0164866 A1*	6/2014	Bolotov
2008/0077825 A1	3/2008	Bello et al.			H03M 13/1111
2008/0155316 A1	6/2008	Pawar et al.			714/785
2008/0162674 A1	7/2008	Dahiya	2014/0173232 A1	6/2014	Reohr et al.
2008/0195833 A1	8/2008	Park	2014/0181575 A1	6/2014	Kalach et al.
2008/0256141 A1	10/2008	Wayda et al.	2014/0195636 A1	7/2014	Karve et al.
2008/0270678 A1	10/2008	Cornwell et al.	2014/0201512 A1	7/2014	Seethaler et al.
2008/0282045 A1	11/2008	Biswas et al.	2014/0201541 A1	7/2014	Paul et al.
2009/0024813 A1	1/2009	Uysal et al.	2014/0208155 A1	7/2014	Pan
2009/0077340 A1	3/2009	Johnson et al.	2014/0215590 A1	7/2014	Brand
2009/0100115 A1	4/2009	Park et al.	2014/0220561 A1	8/2014	Sukumar et al.
2009/0198889 A1	8/2009	Ito et al.	2014/0229654 A1	8/2014	Goss et al.
2009/0282073 A1	11/2009	Roden	2014/0230017 A1	8/2014	Saib
2009/0282301 A1	11/2009	Flynn et al.	2014/0258526 A1	9/2014	Le Sant et al.
2010/0050015 A1	2/2010	Corbett et al.	2014/0282983 A1	9/2014	Ju et al.
2010/0052625 A1	3/2010	Cagno et al.	2014/0285917 A1	9/2014	Cudak et al.
2010/0211723 A1	8/2010	Mukaida	2014/0325262 A1	10/2014	Cooper et al.
2010/0246266 A1	9/2010	Park et al.	2014/0351627 A1	11/2014	Best et al.
2010/0257142 A1	10/2010	Murphy et al.	2014/0373104 A1	12/2014	Gaddam et al.
2010/0262764 A1	10/2010	Liu et al.	2014/0373126 A1	12/2014	Hussain et al.

(56)	References Cited							
U.S. PATENT DOCUMENTS								
2014/0379671 A1*	12/2014	Barrus	G06F 16/2365 707/692	2021/0056099 A1	2/2021	Goodsitt et al.		
2015/0026387 A1	1/2015	Sheredy et al.		2021/0075592 A1	3/2021	Zhuo		
2015/0052393 A1	2/2015	Palmer et al.		2021/0089210 A1	3/2021	Seguy et al.		
2015/0074463 A1	3/2015	Jacoby et al.		2021/0158102 A1	5/2021	Lohia et al.		
2015/0089569 A1	3/2015	Sondhi et al.		2021/0200715 A1	7/2021	Aleksandrovich et al.		
2015/0095515 A1	4/2015	Krithivas et al.		2021/0200747 A1	7/2021	Aleksandrovich et al.		
2015/0106556 A1	4/2015	Yu et al.		2021/0248252 A1	8/2021	Darj		
2015/0106557 A1	4/2015	Yu et al.		2021/0286898 A1	9/2021	Gkoulalas-Divanis et al.		
2015/0113203 A1	4/2015	Dancho et al.		2021/0360833 A1	11/2021	Alissa et al.		
2015/0121137 A1	4/2015	McKnight et al.		FOREIGN PATENT DOCUMENTS				
2015/0134920 A1	5/2015	Anderson et al.		WO	02059749 A1	8/2002		
2015/0149822 A1	5/2015	Coronado et al.		WO	2012087648 A2	6/2012		
2015/0154090 A1	6/2015	Chatterjee et al.		WO	2013071087 A1	5/2013		
2015/0154418 A1	6/2015	Redberg		WO	2014082043 A2	5/2014		
2015/0169712 A1	6/2015	Bhide et al.		WO	2014110137 A1	7/2014		
2015/0193169 A1	7/2015	Sundaram et al.		WO	2016015008 A1	1/2016		
2015/0227543 A1	8/2015	Venkatesh et al.		WO	2016190938 A1	12/2016		
2015/0378888 A1	12/2015	Zhang et al.		WO	2016195759 A1	12/2016		
2016/0026397 A1	1/2016	Nishikido et al.		WO	2016195958 A1	12/2016		
2016/0028806 A1	1/2016	Wareing		WO	2016195961 A1	12/2016		
2016/0078653 A1	3/2016	Intoppa		WO	2018231806 A1	12/2018		
2016/0098323 A1	4/2016	Mutha et al.		WO	2019005766 A1	1/2019		
2016/0147830 A1	5/2016	Zhong et al.		WO	2023147469 A1	8/2023		
2016/0182542 A1	6/2016	Staniford		OTHER PUBLICATIONS				
2016/0239514 A1	8/2016	Srivast et al.		ETSI, "Network Function Virtualisation (NFV); Resiliency Requirements", ETSI GS NFCV-REL 001, V1.1.1, Jan. 2015, 82 pages, etsi.org (online), URL: www.etsi.org/deliver/etsi_gs/NFV-REL/001_099/001/01.01.60/gs_NFV-REL001v010101p.pdf.				
2016/0248631 A1	8/2016	Duchesneau		Faith, "dictzip file format", GitHub.com (online), accessed Jul. 28, 2015, 1 page, URL: github.com/fidlej/idzip.				
2016/0267122 A1	9/2016	Kennedy, Jr.		Google Search of "storage array define" performed by the Examiner on Nov. 4, 2015 for U.S. Appl. No. 14/725,278, Results limited to entries dated before 2012, 1 page.				
2016/0350009 A1	12/2016	Cerretti et al.		Hota et al., "Capability-based Cryptographic Data Access Control in Cloud Computing", International Journal of Advanced Networking and Applications, col. 1, Issue 1, Aug. 2011, 10 pages, Eswar Publications, India.				
2016/0352720 A1	12/2016	Hu et al.		Hu et al., "Container Marking: Combining Data Placement, Garbage Collection and Wear Levelling for Flash", 19th Annual IEEE International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunications Systems, Jul. 25-27, 2011, 11 pages, ISBN: 978-0-7695-4430-4, DOI: 10.1109/MASCOTS.2011.50.				
2016/0352830 A1	12/2016	Borowiec et al.		Hwang et al., "RAID-x: A New Distributed Disk Array for I/O-Centric Cluster Computing", Proceedings of The Ninth International Symposium on High-performance Distributed Computing, Aug. 2000, pp. 279-286, The Ninth International Symposium on High-Performance Distributed Computing, IEEE Computer Society, Los Alamitos, CA.				
2016/0352834 A1	12/2016	Borowiec et al.		International Search Report and Written Opinion, PCT/US2016/015006, Apr. 29, 2016, 12 pages.				
2017/0046081 A1	2/2017	Palmer et al.		International Search Report and Written Opinion, PCT/US2016/015008, May 4, 2016, 12 pages.				
2017/0109369 A1	4/2017	Feng et al.		International Search Report and Written Opinion, PCT/US2016/016333, Jun. 8, 2016, 12 pages.				
2017/0147441 A1	5/2017	Binford et al.		International Search Report and Written Opinion, PCT/US2016/020410, Jul. 8, 2016, 12 pages.				
2017/0220276 A1	8/2017	Palmer et al.		International Search Report and Written Opinion, PCT/US2016/032052, Aug. 30, 2016, 17 pages.				
2017/0262202 A1	9/2017	Seo		International Search Report and Written Opinion, PCT/US2016/032084, Jul. 18, 2016, 12 pages.				
2017/0371750 A1	12/2017	Horowitz et al.		International Search Report and Written Opinion, PCT/US2016/035492, Aug. 17, 2016, 10 pages.				
2018/0054454 A1	2/2018	Astigarraga et al.		International Search Report and Written Opinion, PCT/US2016/036693, Aug. 29, 2016, 10 pages.				
2018/0067018 A1	3/2018	Palmer et al.		International Search Report and Written Opinion, PCT/US2016/038758, Oct. 7, 2016, 10 pages.				
2018/0081562 A1	3/2018	Vasudevan		International Search Report and Written Opinion, PCT/US2016/040393, Sep. 22, 2016, 10 pages.				
2018/0089037 A1	3/2018	Liu et al.						
2018/0217883 A1	8/2018	Murugan et al.						
2018/0254092 A1	9/2018	Liu et al.						
2018/0260125 A1	9/2018	Botes et al.						
2018/0276134 A1	9/2018	Cherubini et al.						
2018/0357431 A1	12/2018	Ellingson et al.						
2018/0365193 A1	12/2018	Chattopadhyay et al.						
2019/0042606 A1	2/2019	Griffith et al.						
2019/0073152 A1	3/2019	Nagle et al.						
2019/0114289 A1	4/2019	Dang et al.						
2019/0121566 A1	4/2019	Gold et al.						
2019/0121673 A1	4/2019	Gold et al.						
2019/0121889 A1	4/2019	Gold et al.						
2019/0171777 A1	6/2019	Deraz et al.						
2019/0220315 A1	7/2019	Vallala et al.						
2019/0251279 A1	8/2019	Emberson et al.						
2019/0304562 A1	10/2019	Cai et al.						
2019/0317860 A1	10/2019	Nightingale						
2019/0354628 A1	11/2019	Grunwald et al.						
2019/0354708 A1	11/2019	Fisher et al.						
2019/0369875 A1	12/2019	Meiri et al.						
2019/0370354 A1	12/2019	Kucherov et al.						
2020/0012933 A1	1/2020	Truong et al.						
2020/0026459 A1*	1/2020	Harduf	G06F 3/0656					
2020/0034560 A1	1/2020	Natanzon et al.						
2020/0310691 A1	10/2020	Byun						
2020/0326871 A1	10/2020	Wu et al.						
2020/0336297 A1	10/2020	Zhuo						
2020/0341656 A1	10/2020	Chawla et al.						
2020/0341867 A1	10/2020	Horowitz et al.						
2020/0342260 A1	10/2020	Farchi et al.						
2020/0349169 A1	11/2020	Venkatesan et al.						
2020/0387518 A1	12/2020	Kesarwani et al.						
2021/0019237 A1	1/2021	Karr et al.						

(56)

References Cited**OTHER PUBLICATIONS**

- International Search Report and Written Opinion, PCT/US2016/044020, Sep. 30, 2016, 11 pages.
- International Search Report and Written Opinion, PCT/US2016/044874, Oct. 7, 2016, 11 pages.
- International Search Report and Written Opinion, PCT/US2016/044875, Oct. 5, 2016, 13 pages.
- International Search Report and Written Opinion, PCT/US2016/044876, Oct. 21, 2016, 12 pages.
- International Search Report and Written Opinion, PCT/US2016/044877, Sep. 29, 2016, 13 pages.
- Kong, "Using PCI Express as The Primary System Interconnect in Multiroot Compute, Storage, Communications and Embedded Systems", White Paper, IDT.com (online), Aug. 28, 2008, 12 pages, URL: www.idt.com/document/whp/idt-pcie-multi-root-white-paper.
- Li et al., "Access Control for the Services Oriented Architecture", Proceedings of the 2007 ACM Workshop on Secure Web Services (SWS '07), Nov. 2007, pp. 9-17, ACM New York, NY.
- Microsoft, "Hybrid for SharePoint Server 2013—Security Reference Architecture", Microsoft (online), Oct. 2014, 53 pages, URL: hybrid.office.com/img/Security_Reference_Architecture.pdf.
- Microsoft, "Hybrid Identity Management", Microsoft (online), Apr. 2014, 2 pages, URL: download.microsoft.com/download/E/A/E/EAE57CD1-A80B-423C-96BB-142FAAC630B9/Hybrid_Identity_Datasheet.pdf.
- Microsoft, "Hybrid Identity", Microsoft (online), Apr. 2014, 36 pages, URL: www.aka.ms/HybridIdentityWp.
- PCMag, "Storage Array Definition", Published May 10, 2013, URL: <http://web.archive.org/web/20130510121646/http://www.pcmag.com/encyclopedia/term/52091/storage-array>, 2 pages.
- Stalzer, "FlashBlades: System Architecture and Applications", Proceedings of the 2nd Workshop on Architectures and Systems for Big Data, Jun. 2012, pp. 10-14, Association for Computing Machinery, New York, NY.
- Storer et al., "Pergamum: Replacing Tape with Energy Efficient, Reliable, Disk-Based Archival Storage", FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies, Article No. 1, Feb. 2008, pp. 1-16, USENIX Association, Berkeley, CA.
- Storer et al., "Secure Data Deduplication", Proceedings of the 4th ACM International Workshop on Storage Security and Survivability (StorageSS'08), Oct. 2008, 10 pages, ACM New York, NY, USA, DOI: 10.1145/1456469.1456471.
- Sweere, "Creating Storage Class Persistent Memory with NVDIMM", Published in Aug. 2013, Flash Memory Summit 2013, URL: http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2013/20130814_T2_Sweere.pdf, 22 pages.
- Techopedia, "What is a disk array", techopedia.com (online), Jan. 13, 2012, 1 page, URL: web.archive.org/web/20120113053358/http://www.techopedia.com/definition/1009/disk-array.
- Webopedia, "What is a disk array", webopedia.com (online), May 26, 2011, 2 pages, URL: web.archive.org/web/20110526081214/http://www.webopedia.com/TERM/D/disk_array.html.
- Wikipedia, "Convergent Encryption", Wikipedia.org (online), accessed Sep. 8, 2015, 2 pages, URL: en.wikipedia.org/wiki/Convergent_encryption.
- International Search Report and Written Opinion, PCT/US2023/061446, May 4, 2023, 21 pages.

* cited by examiner

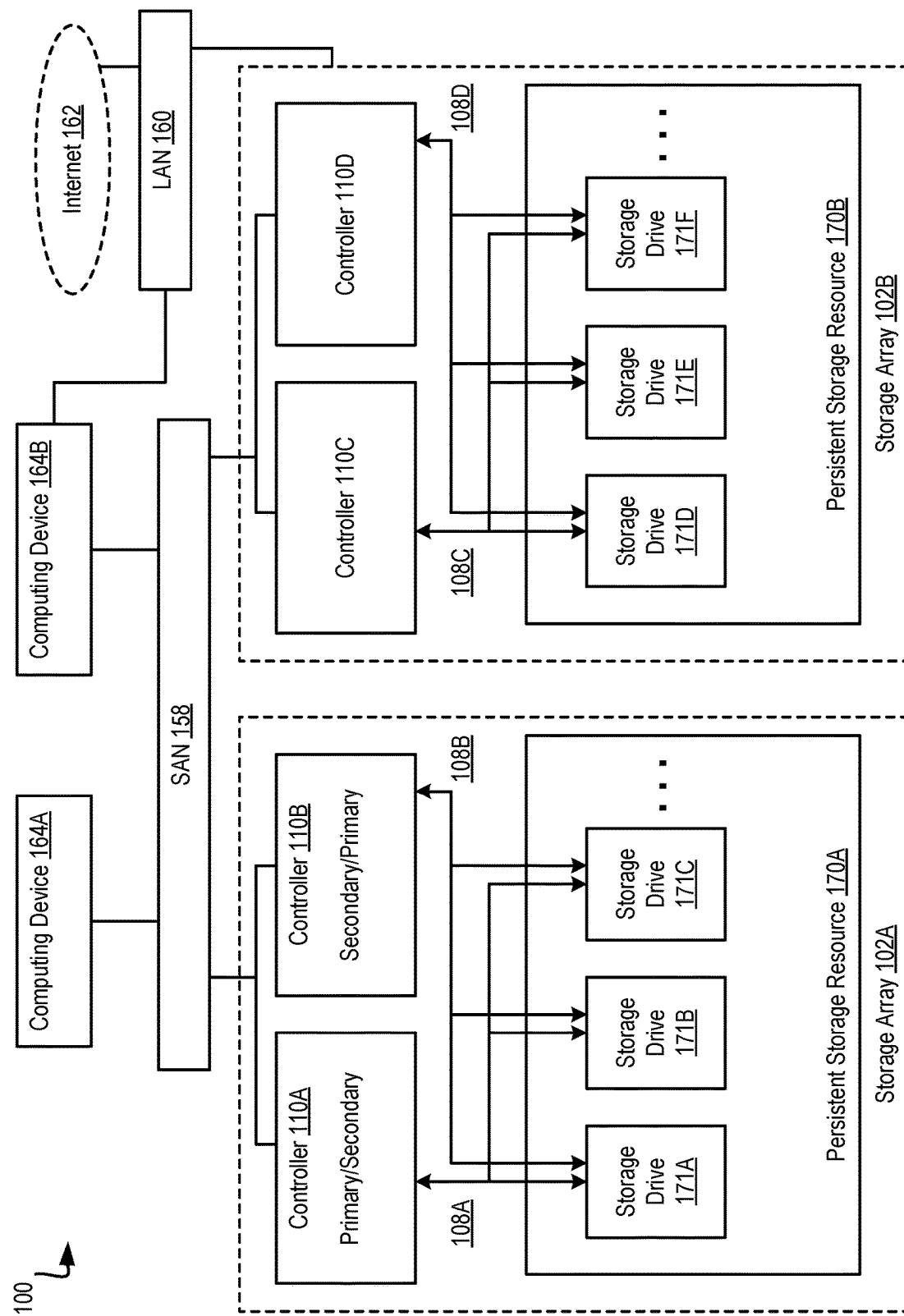


FIG. 1A

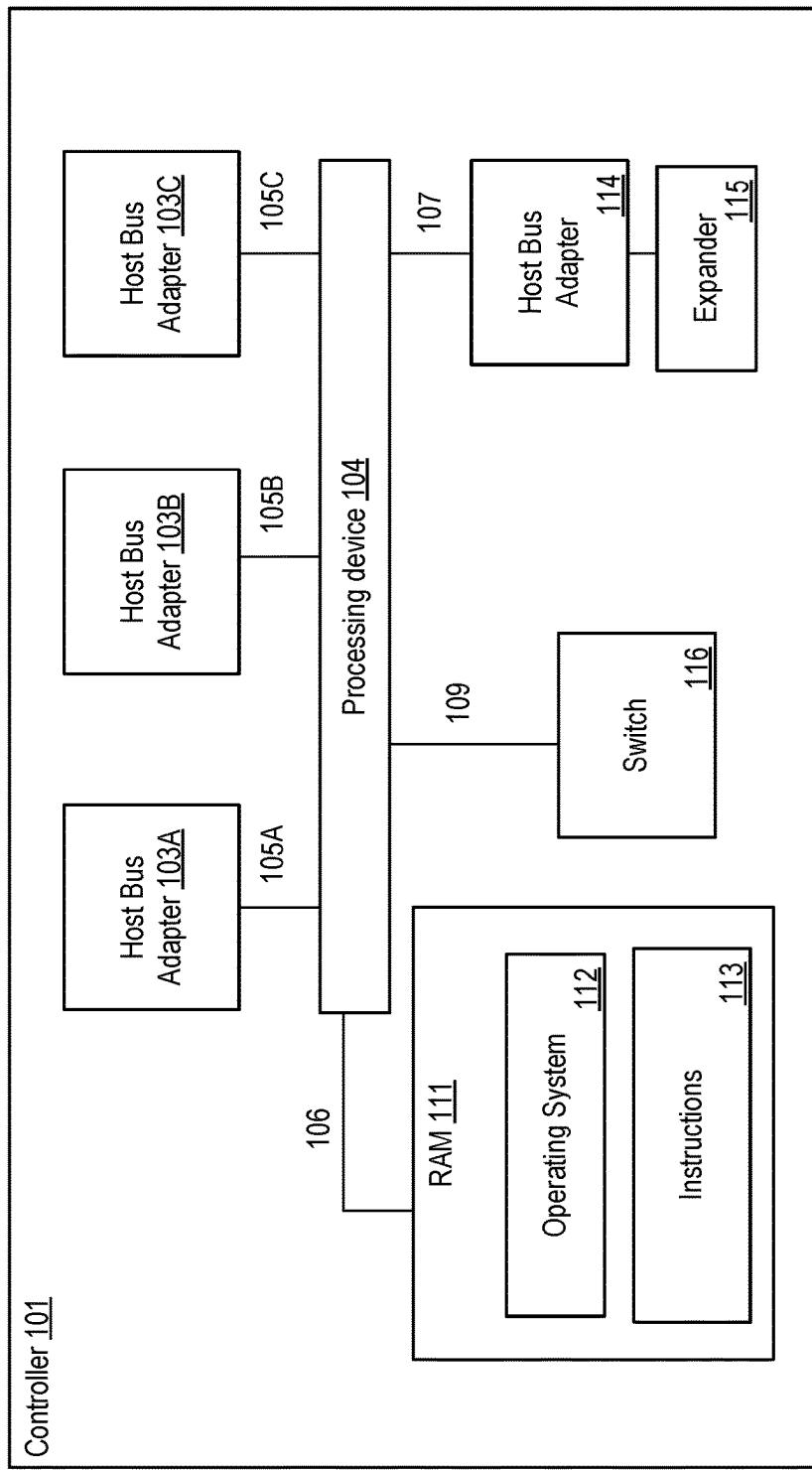


FIG. 1B

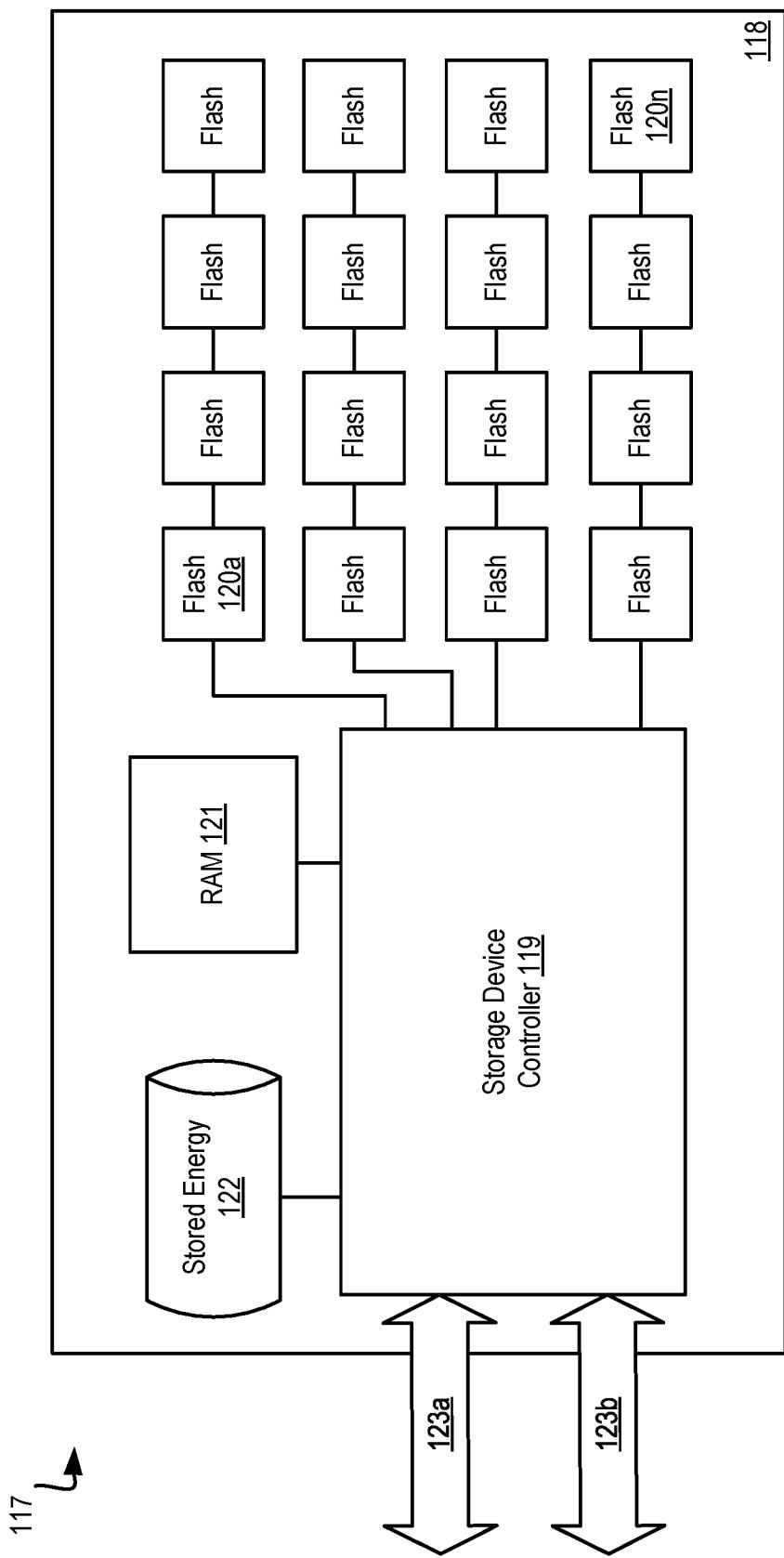


FIG. 1C

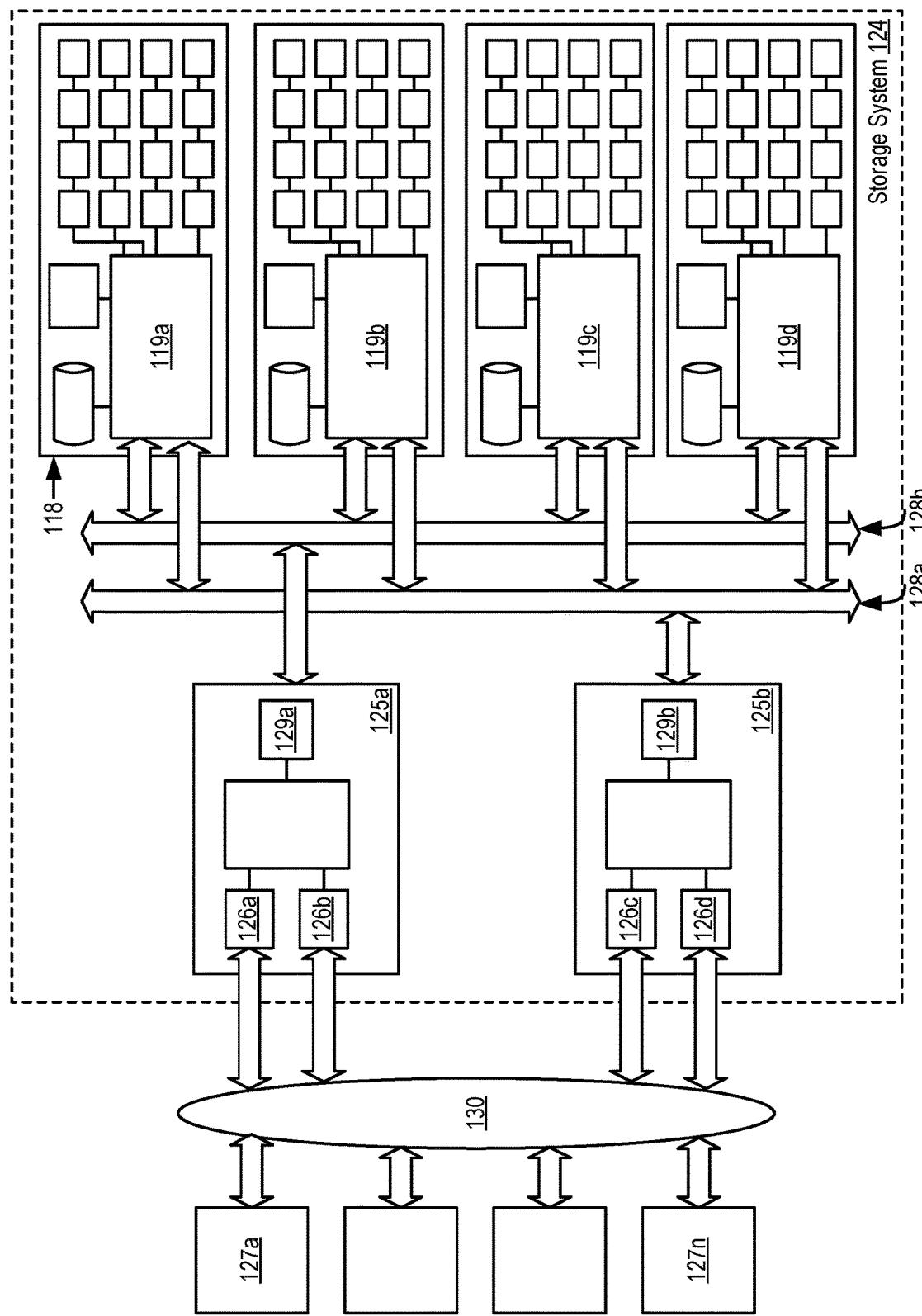


FIG. 1D

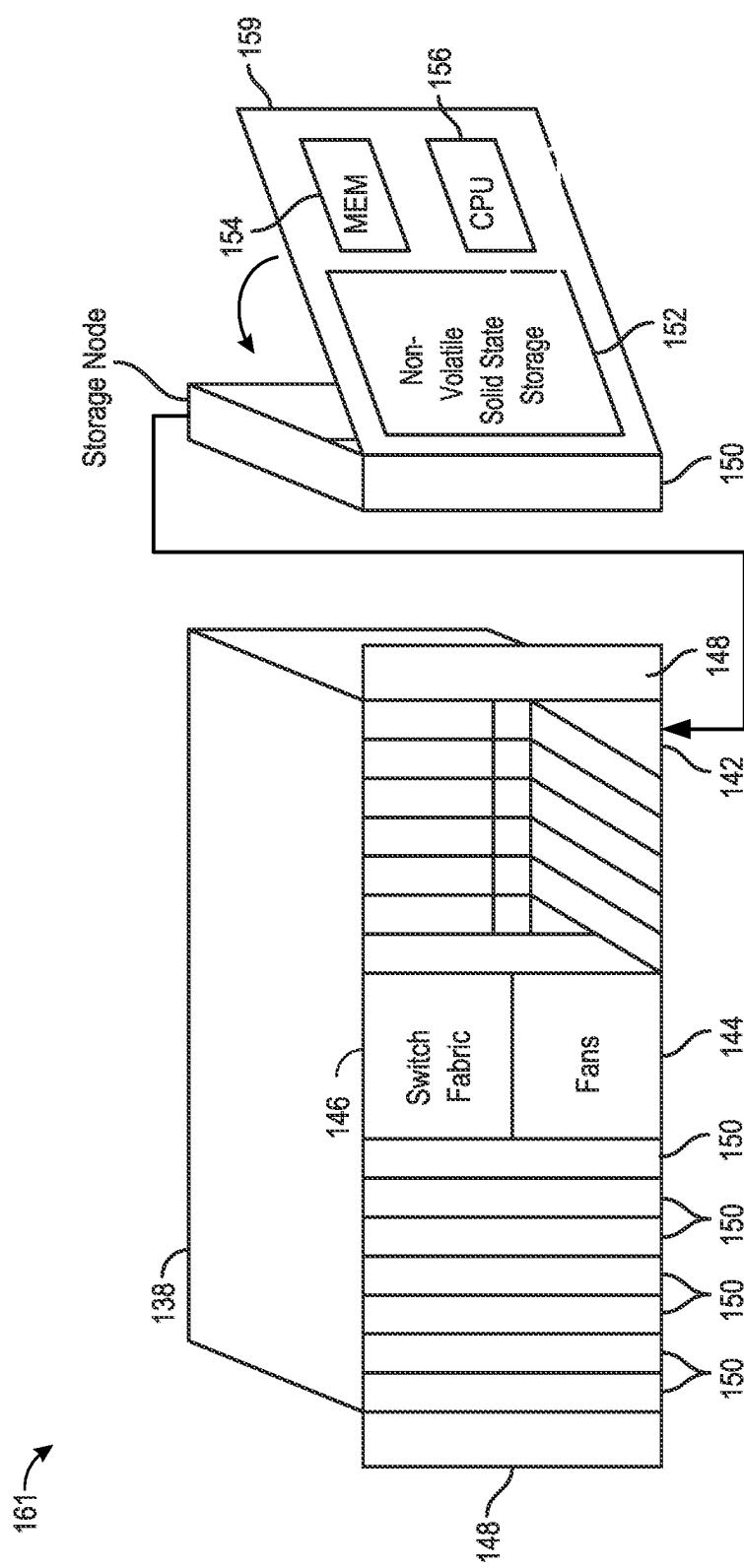


FIG. 2A

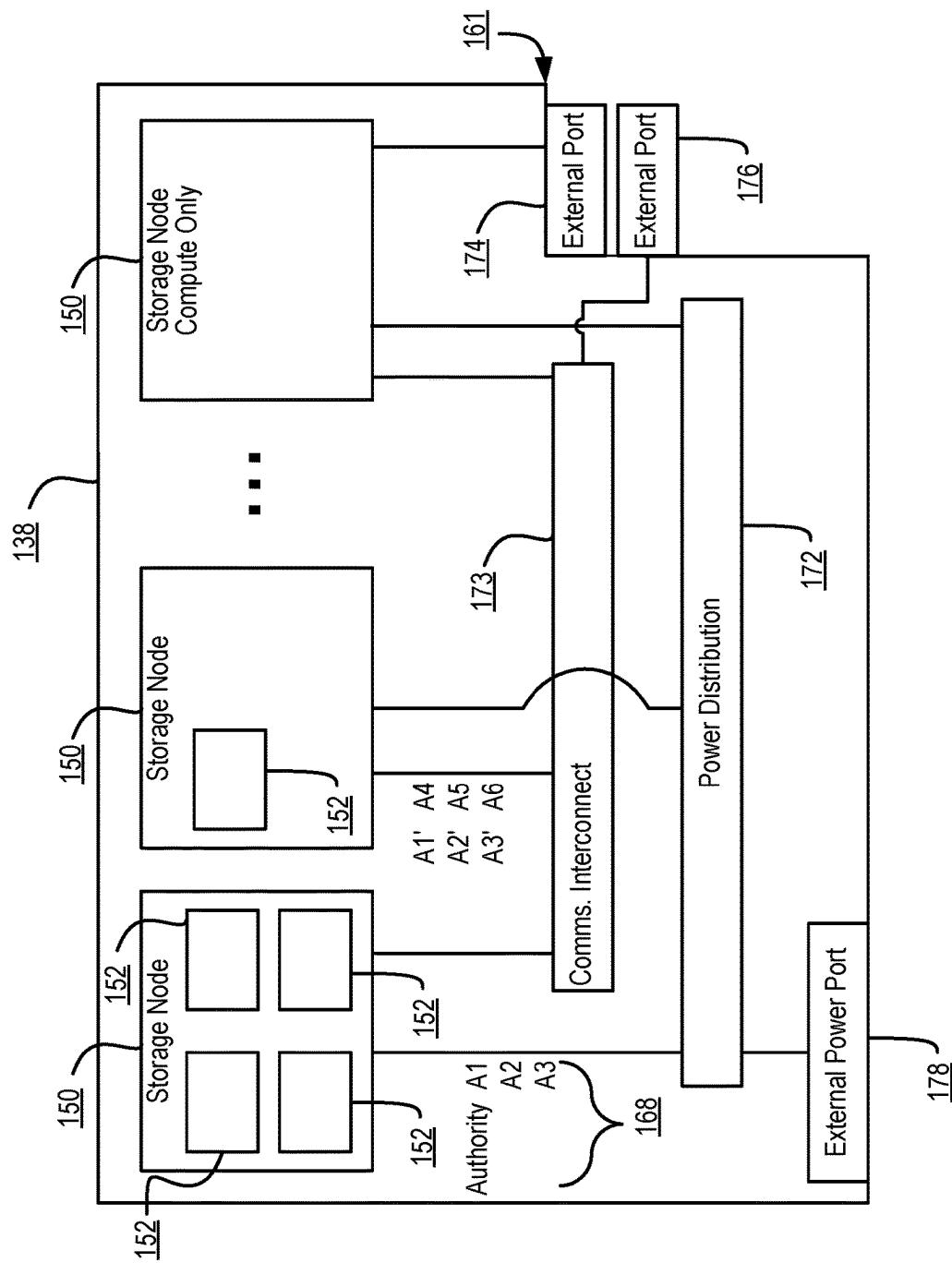


FIG. 2B

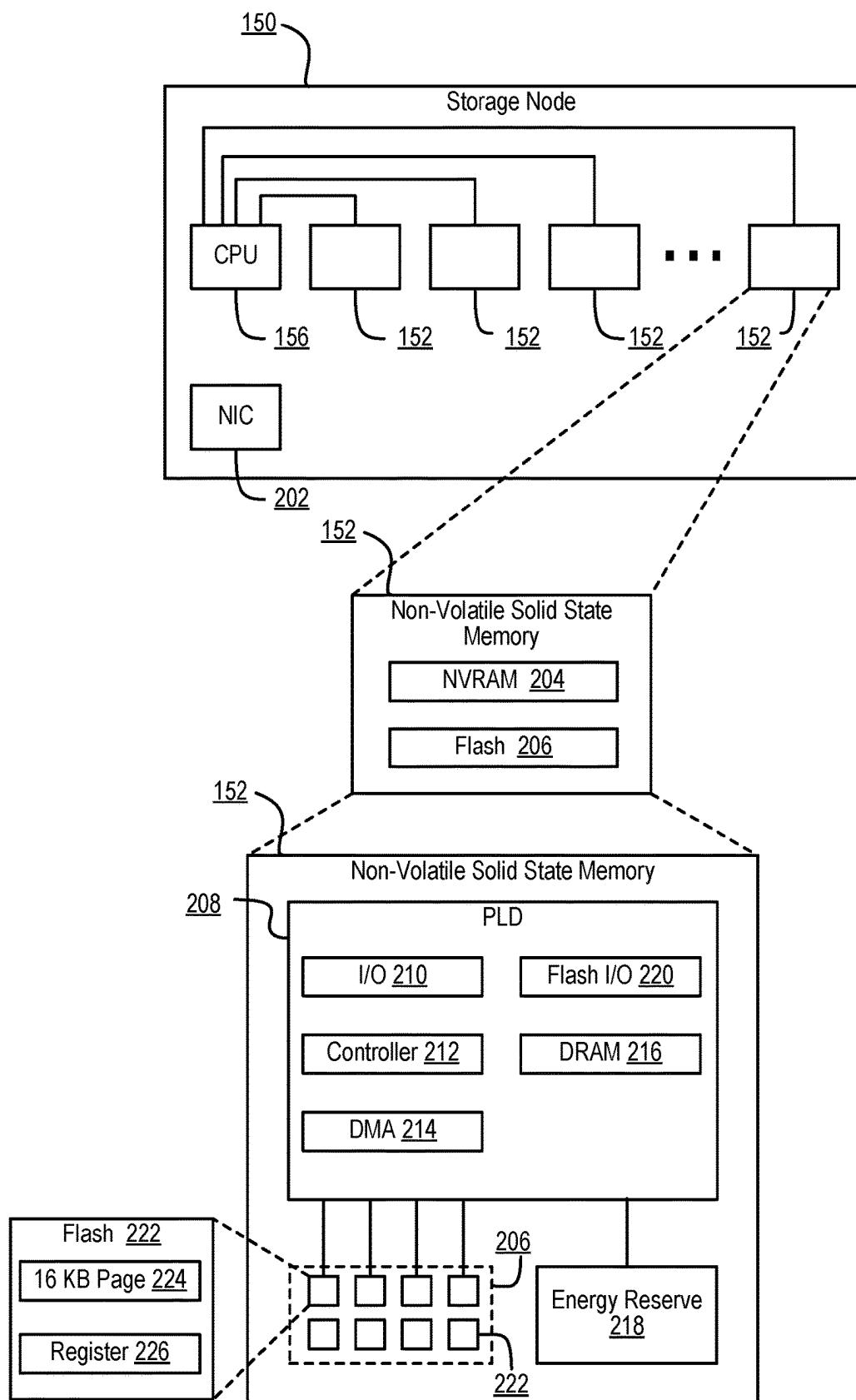


FIG. 2C

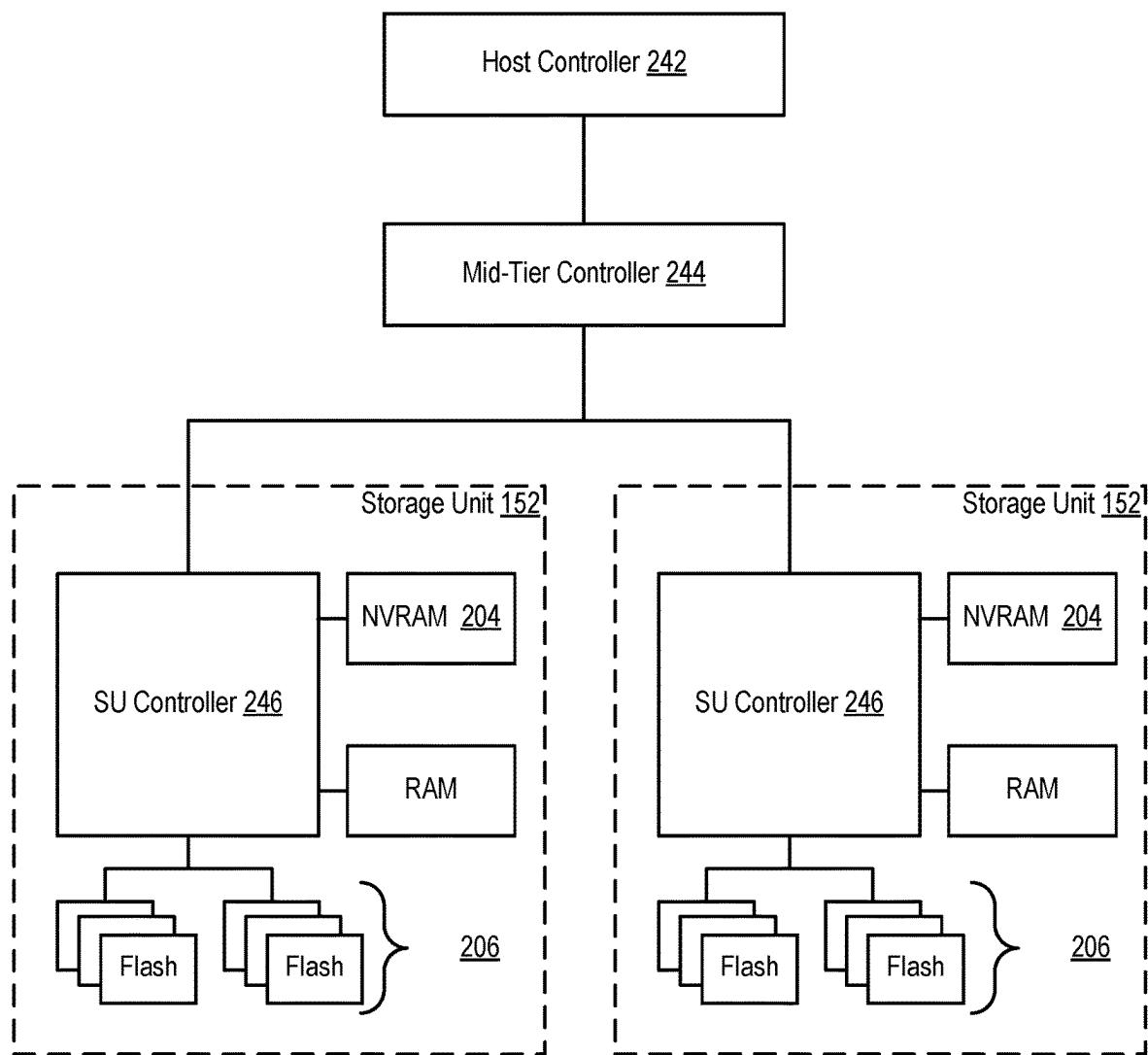


FIG. 2D

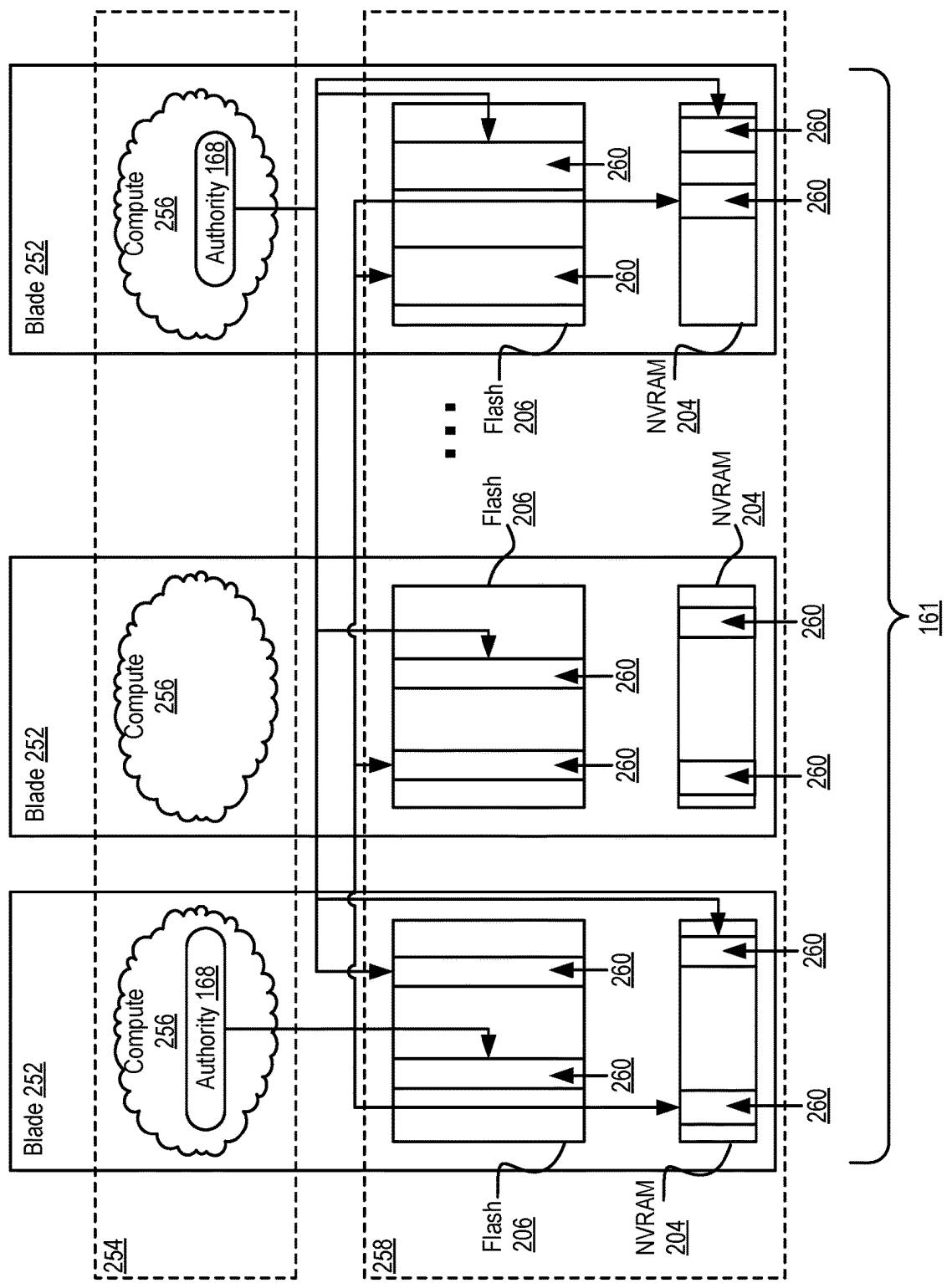


FIG. 2E

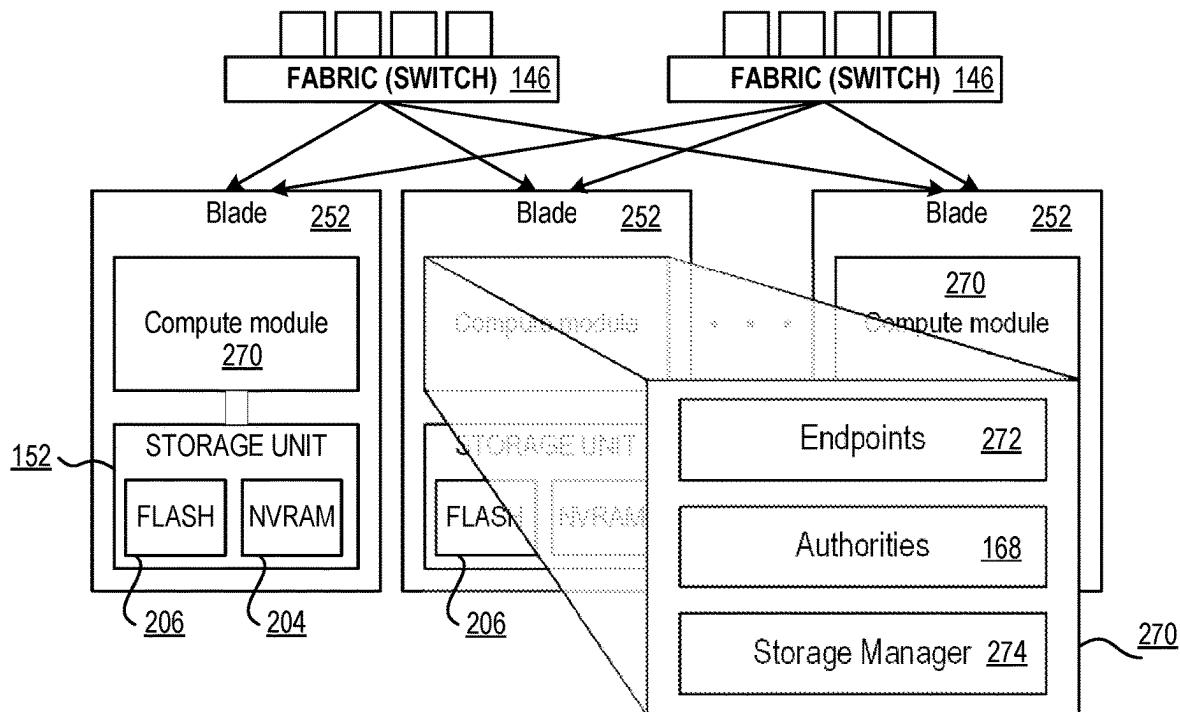


FIG. 2F

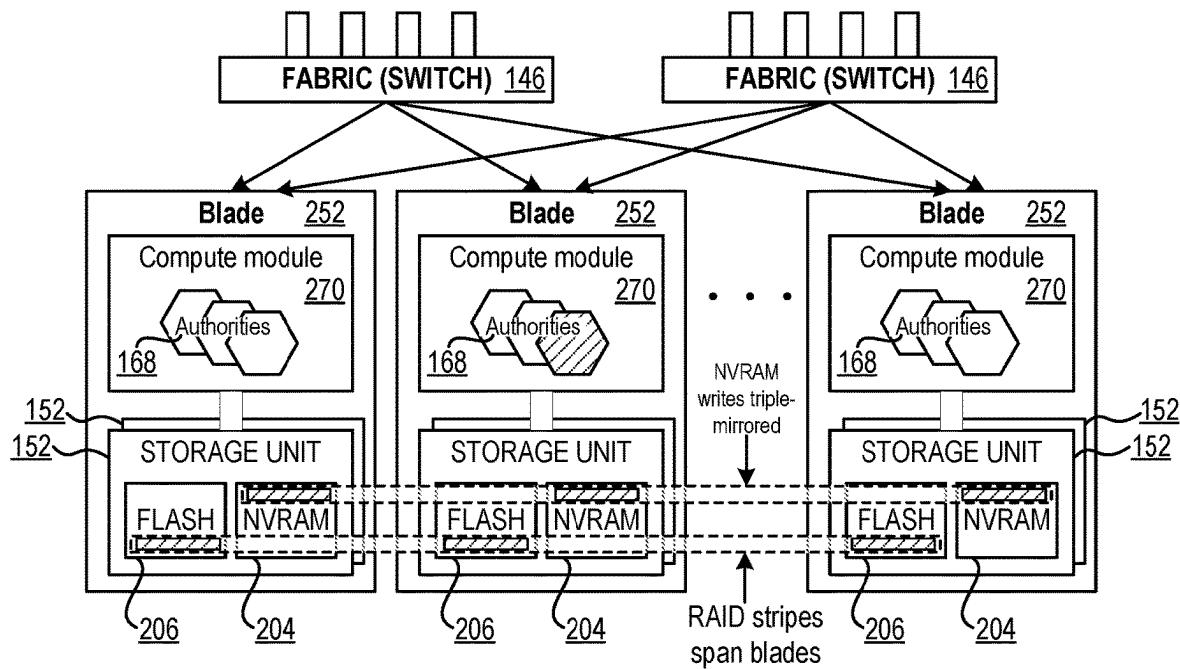


FIG. 2G

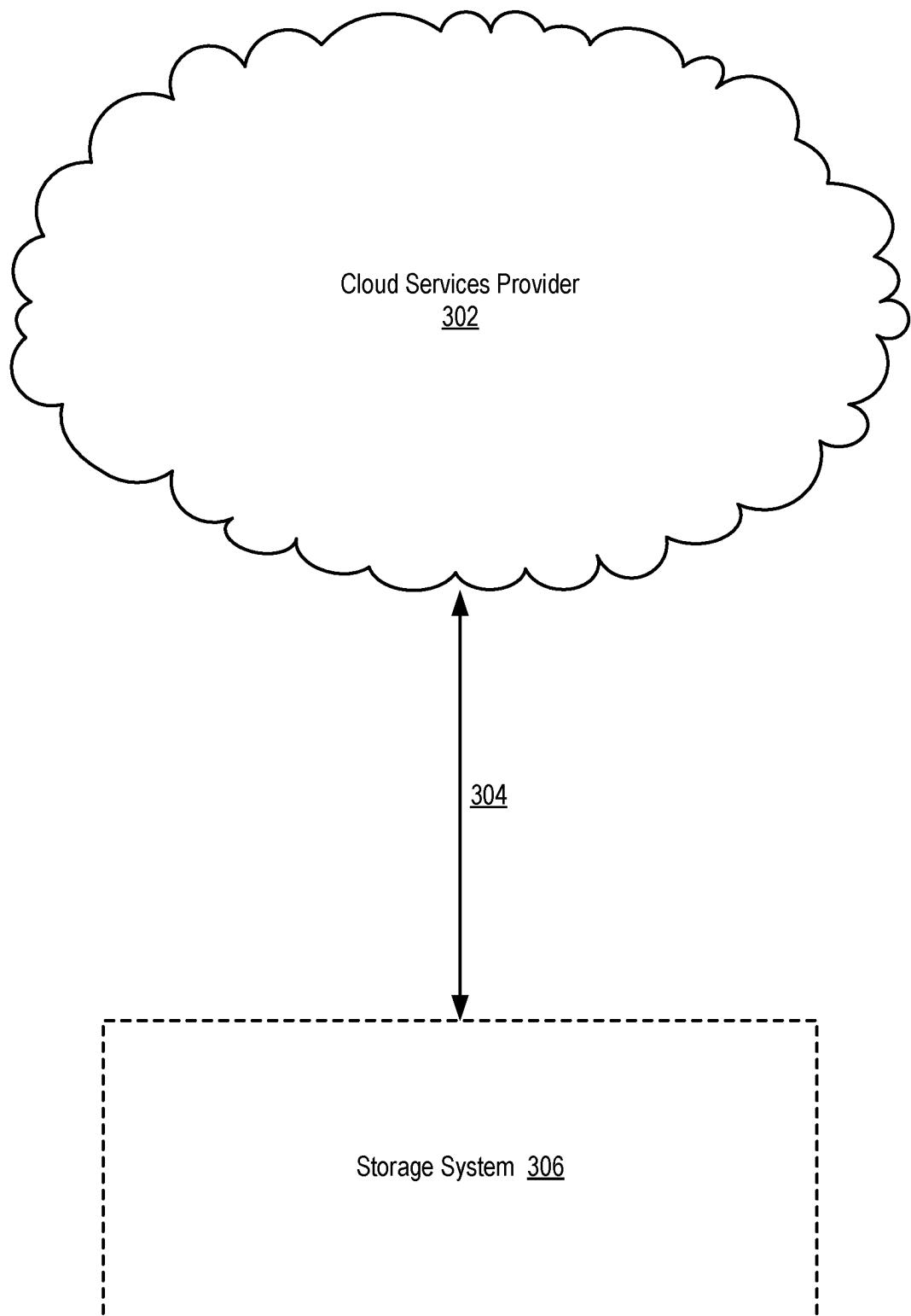


FIG. 3A

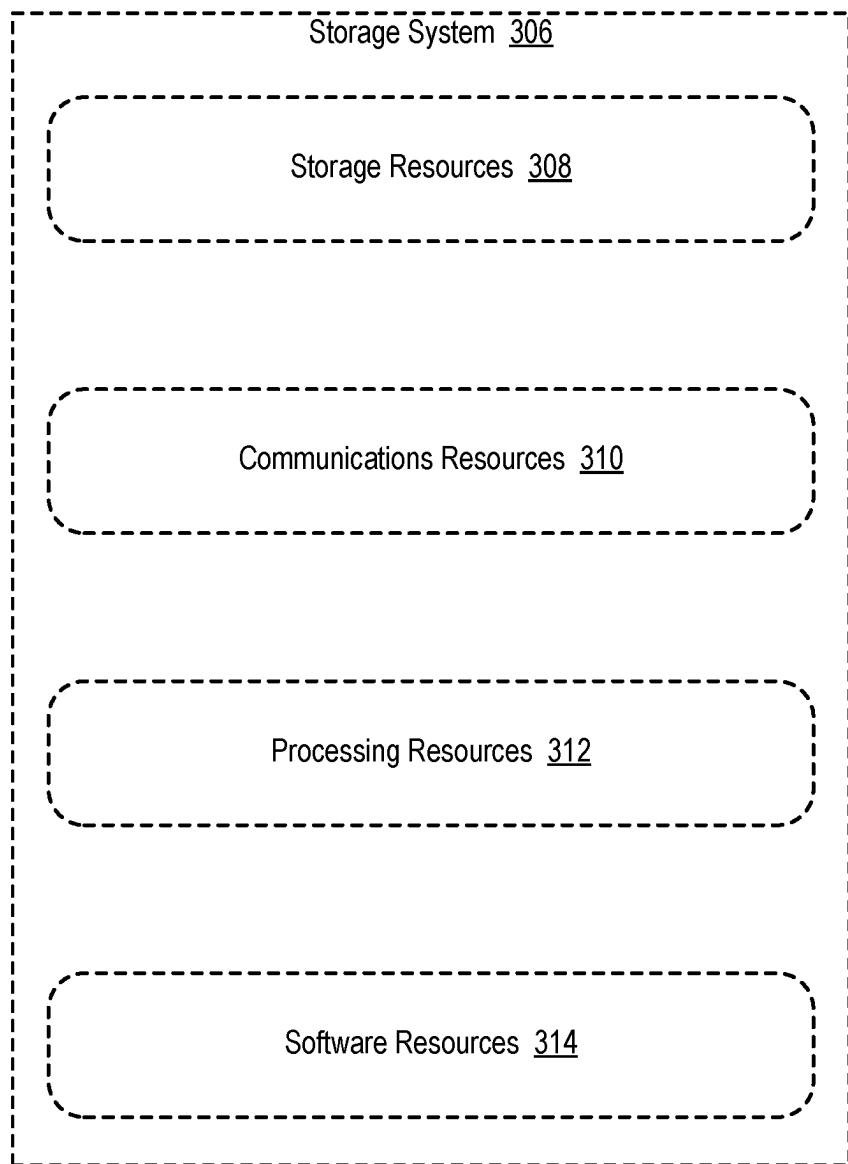


FIG. 3B

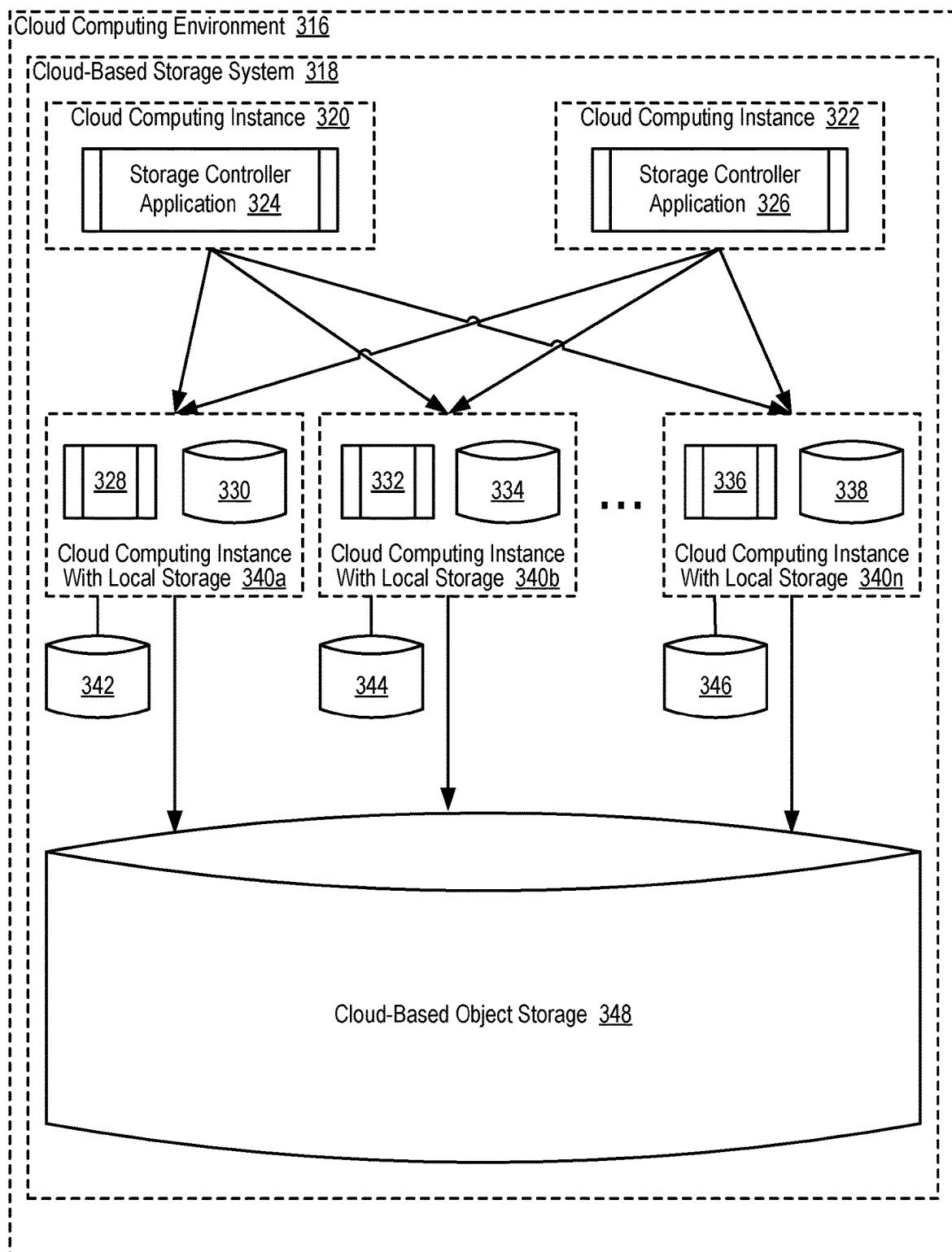


FIG. 3C

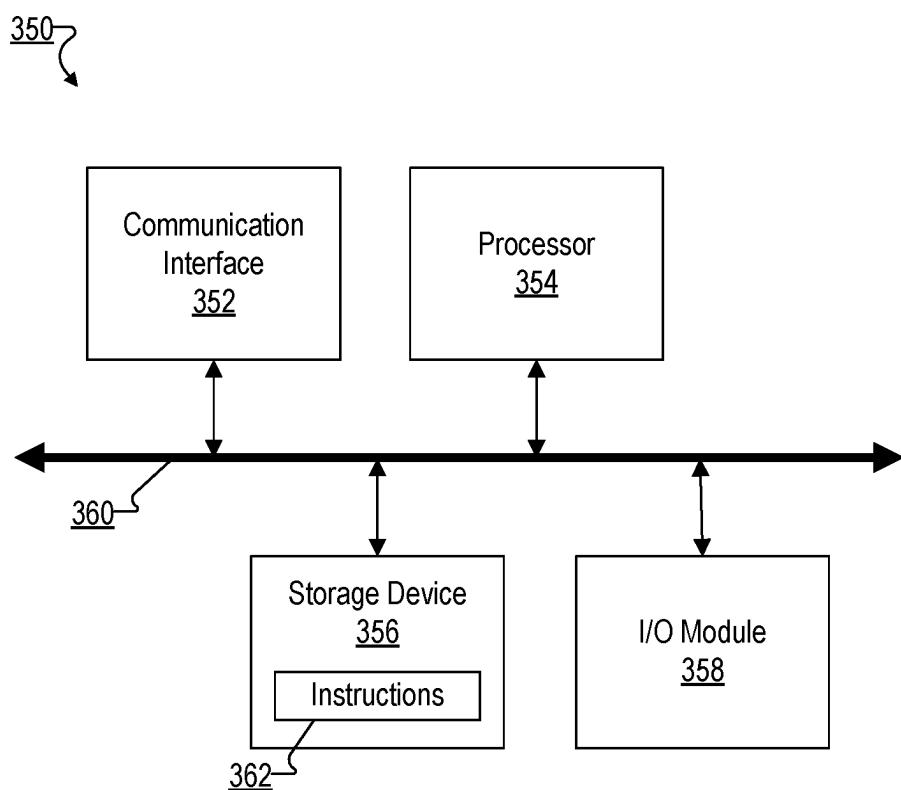


FIG. 3D

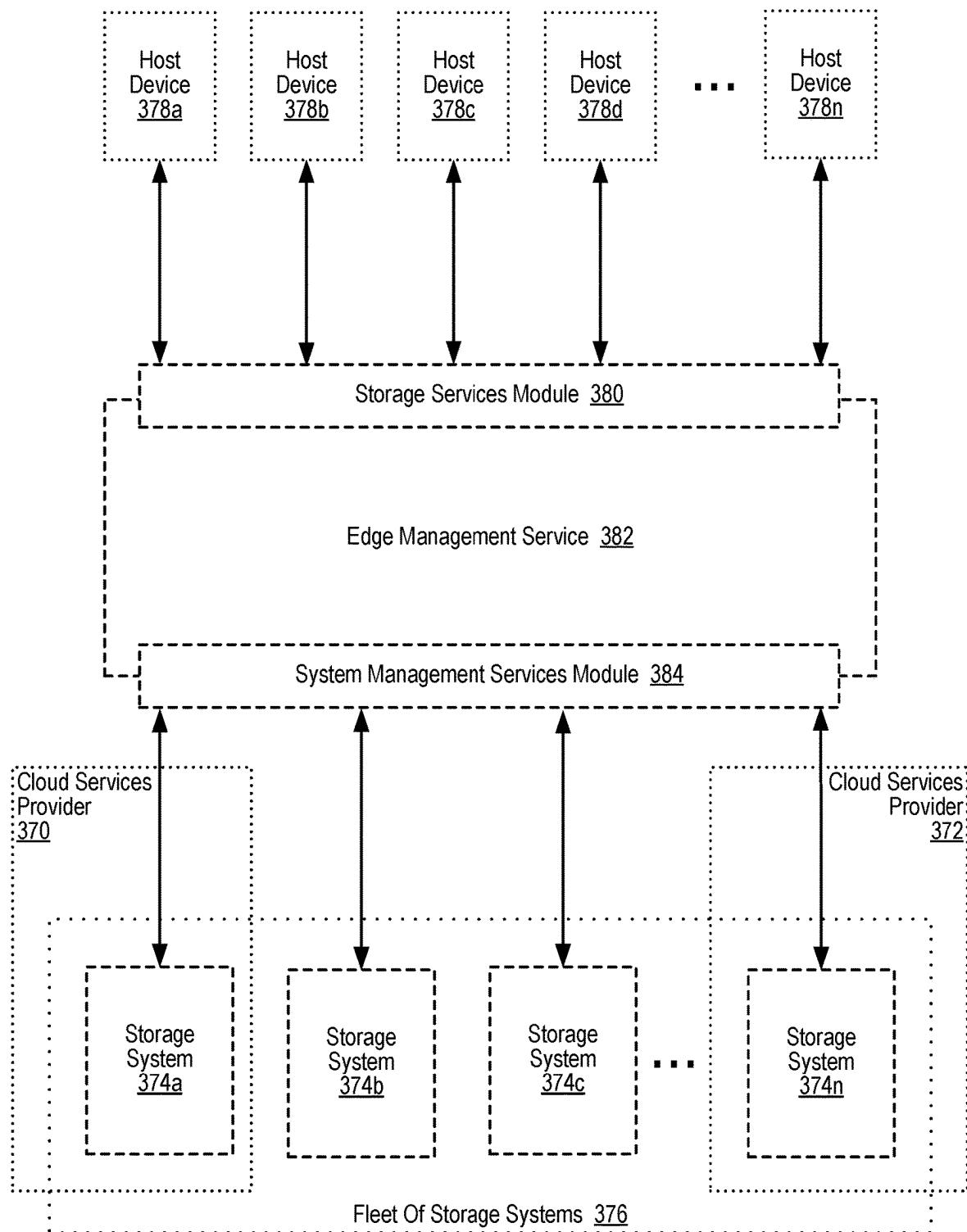


FIG. 3E

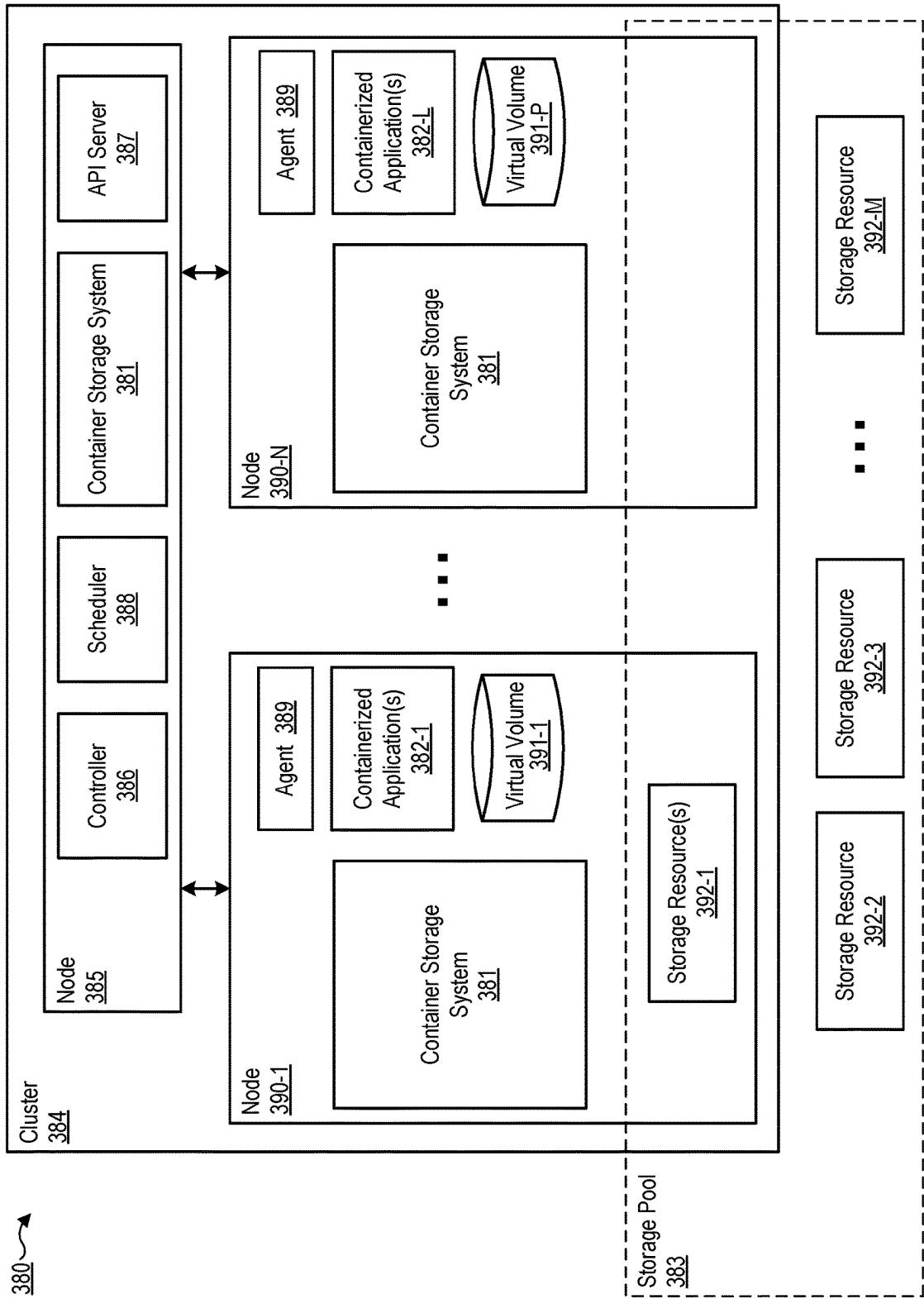


FIG. 3F

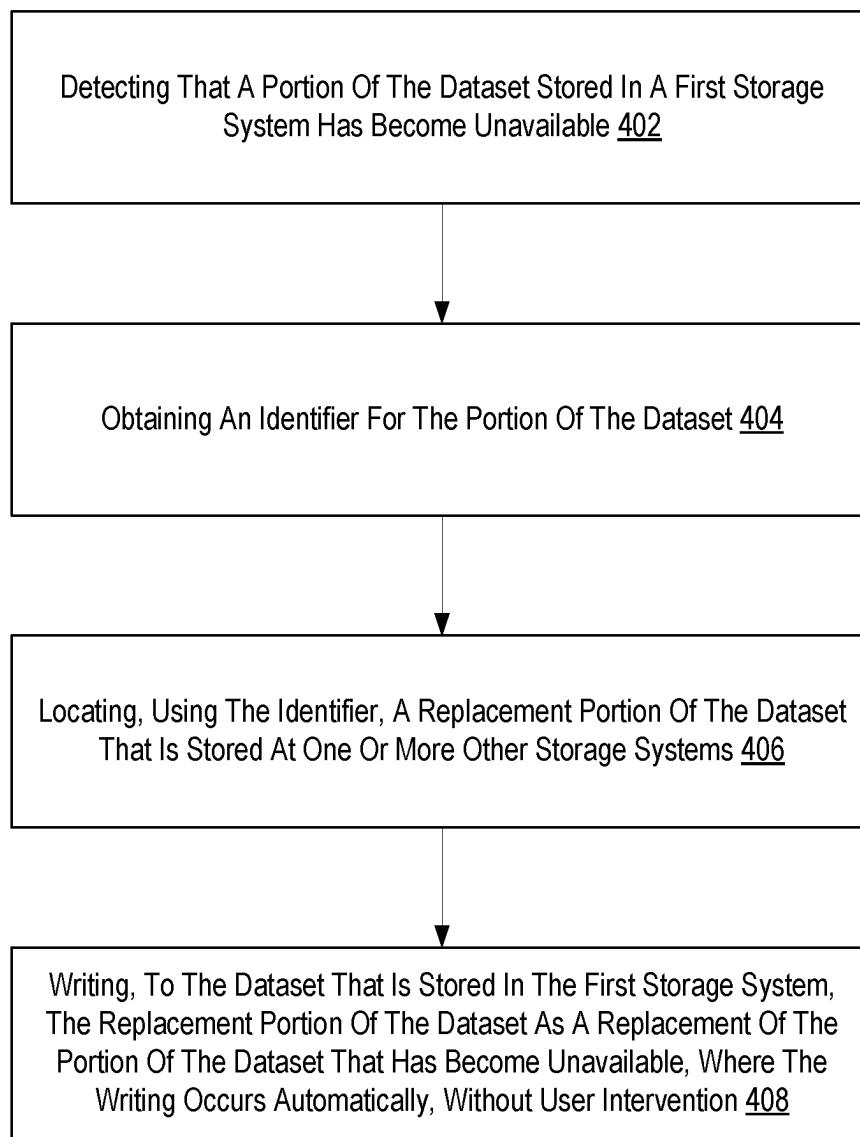


FIG. 4

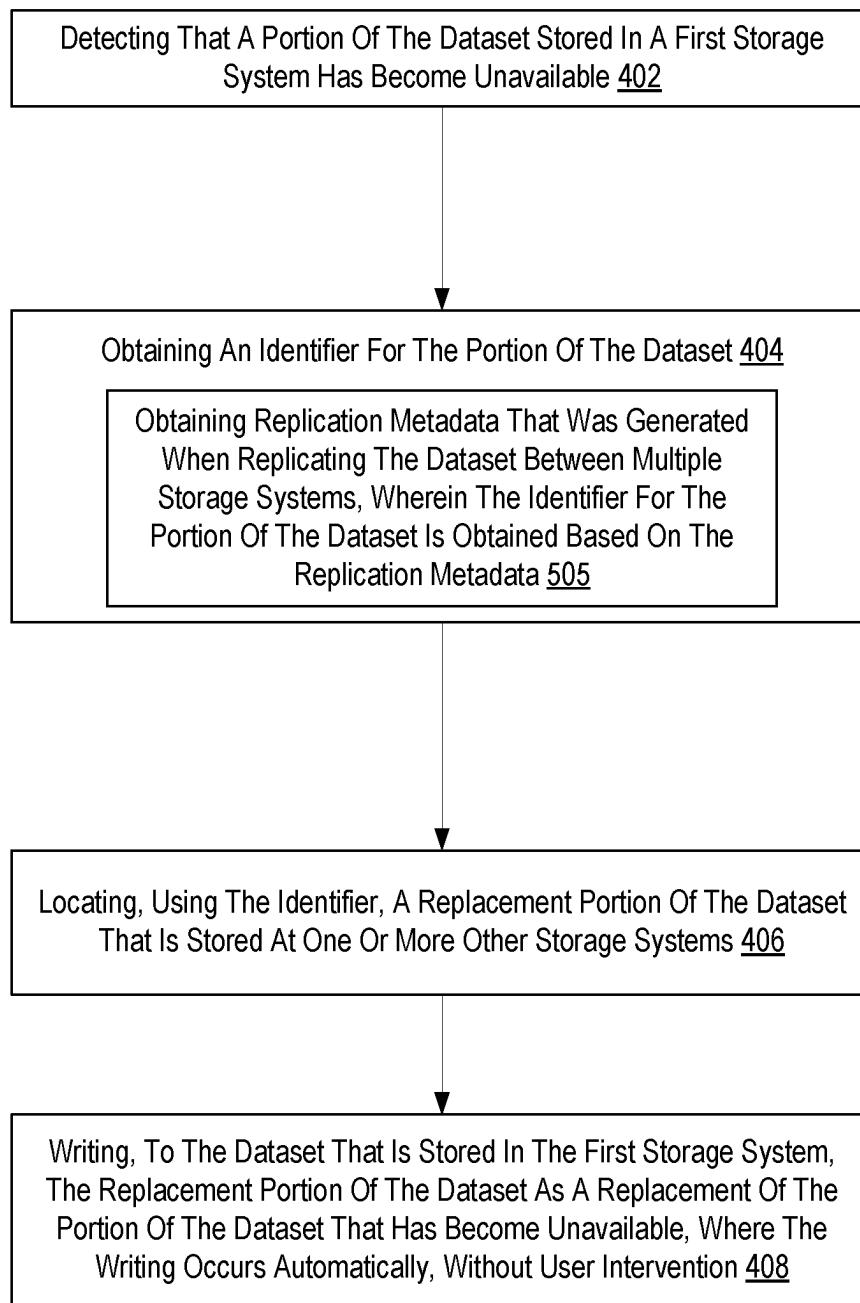


FIG. 5

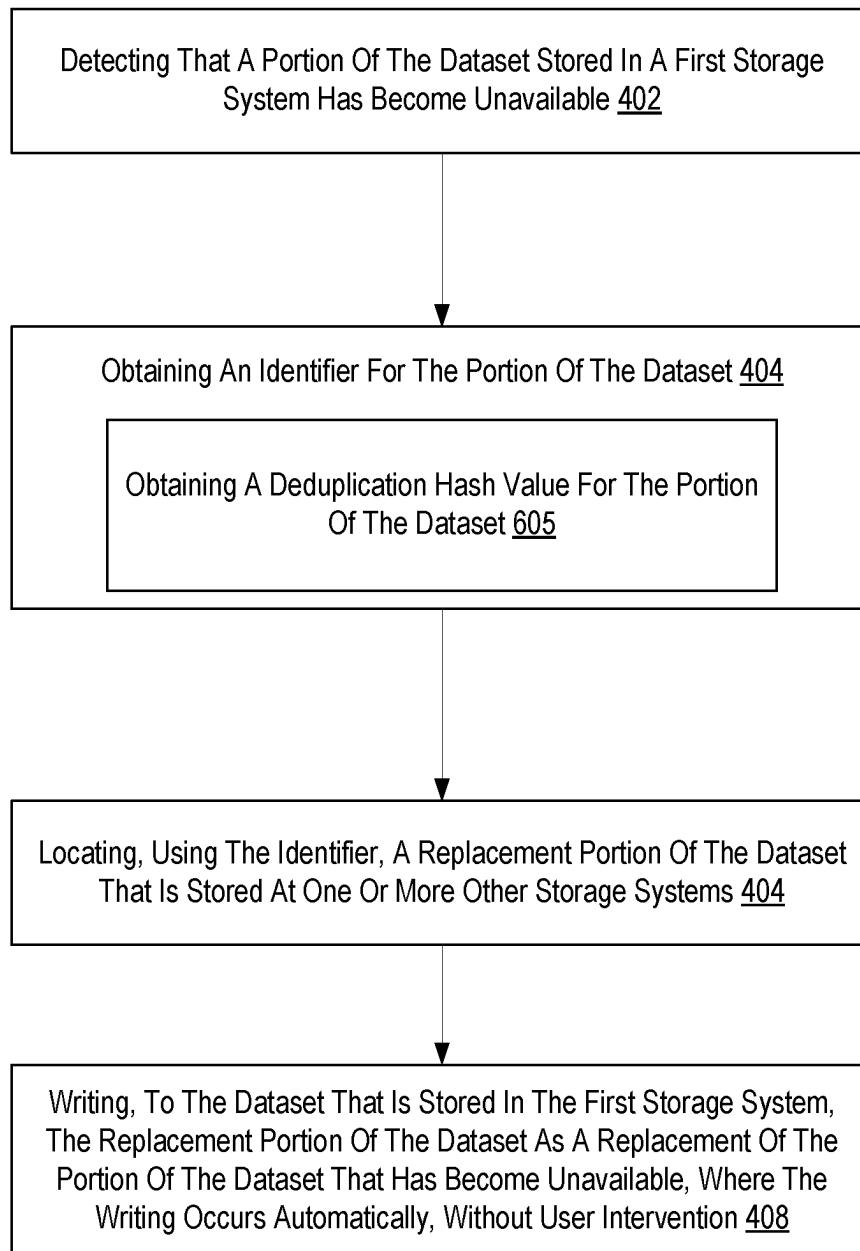


FIG. 6

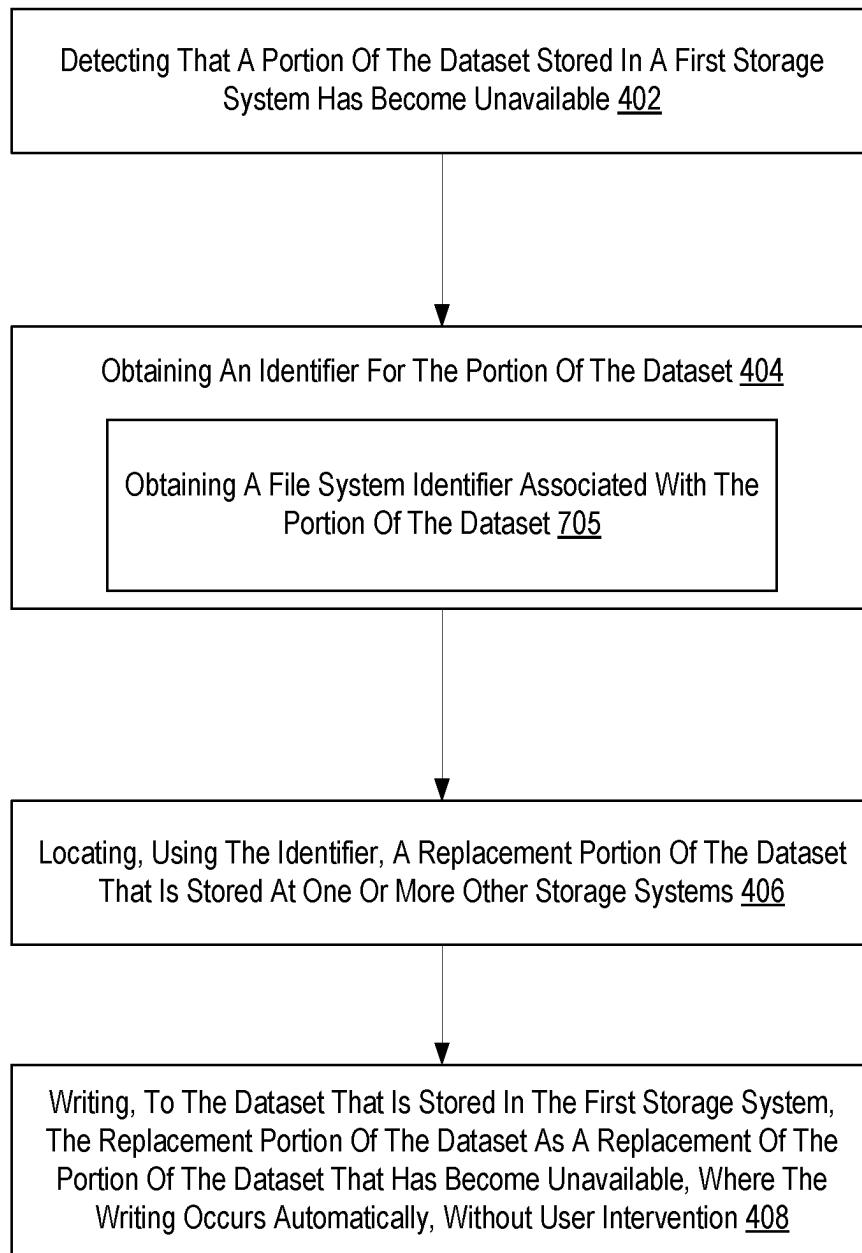


FIG. 7

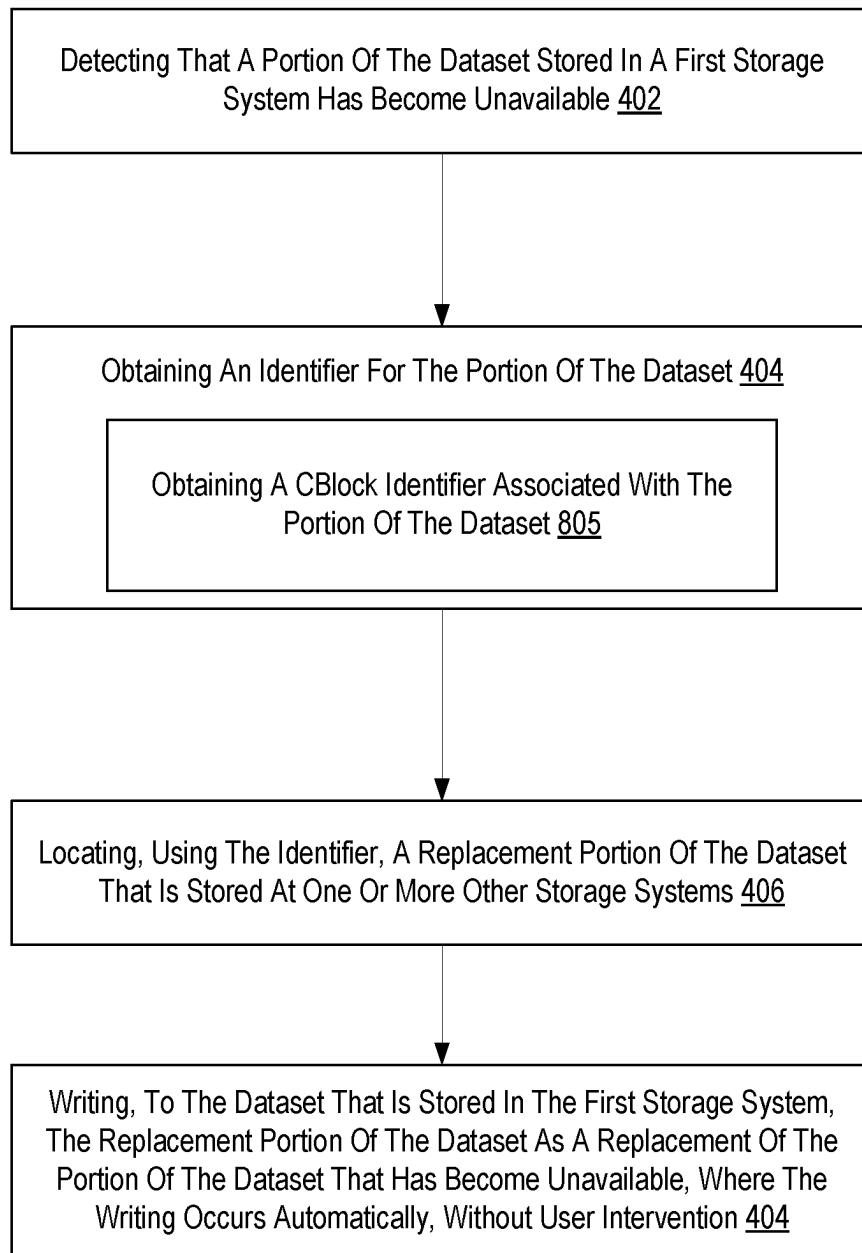


FIG. 8

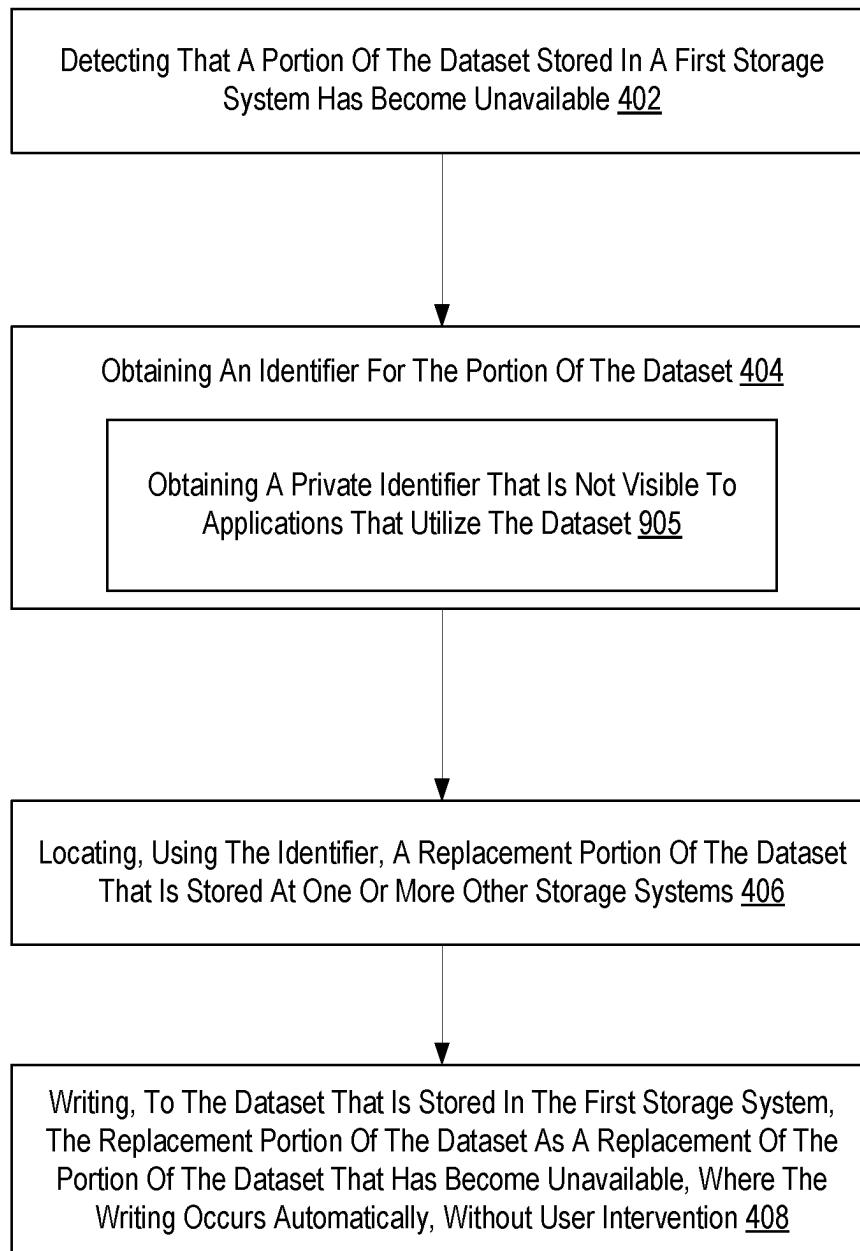


FIG. 9

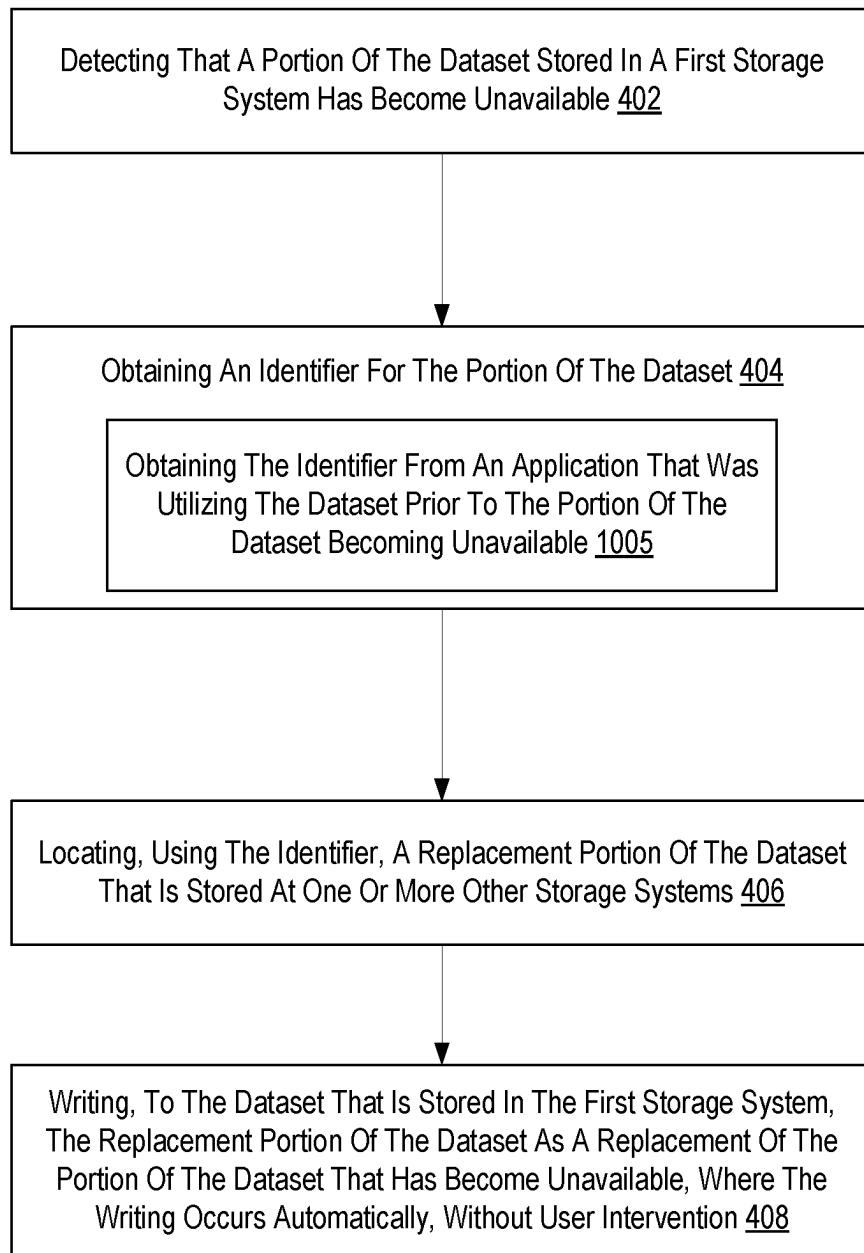


FIG. 10

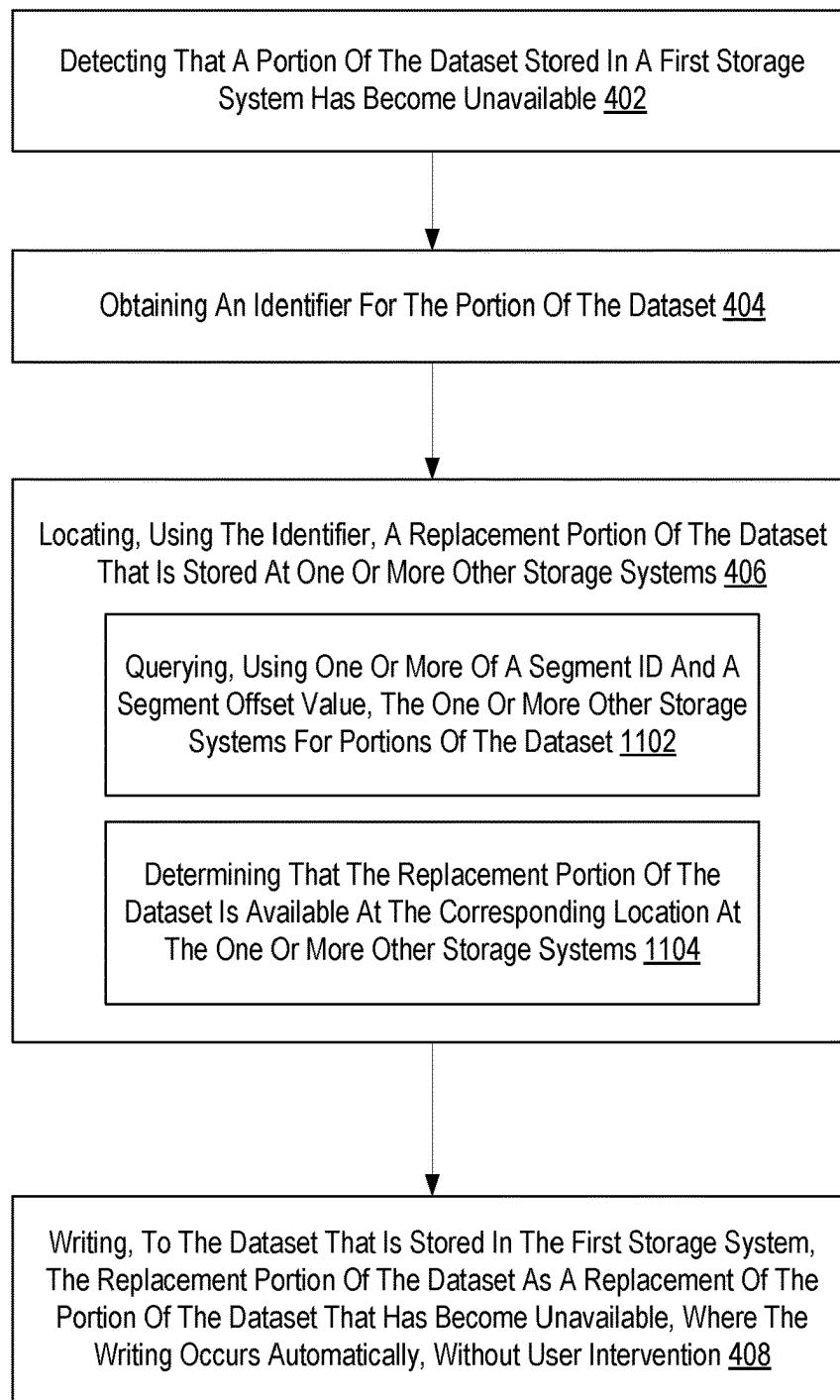


FIG. 11

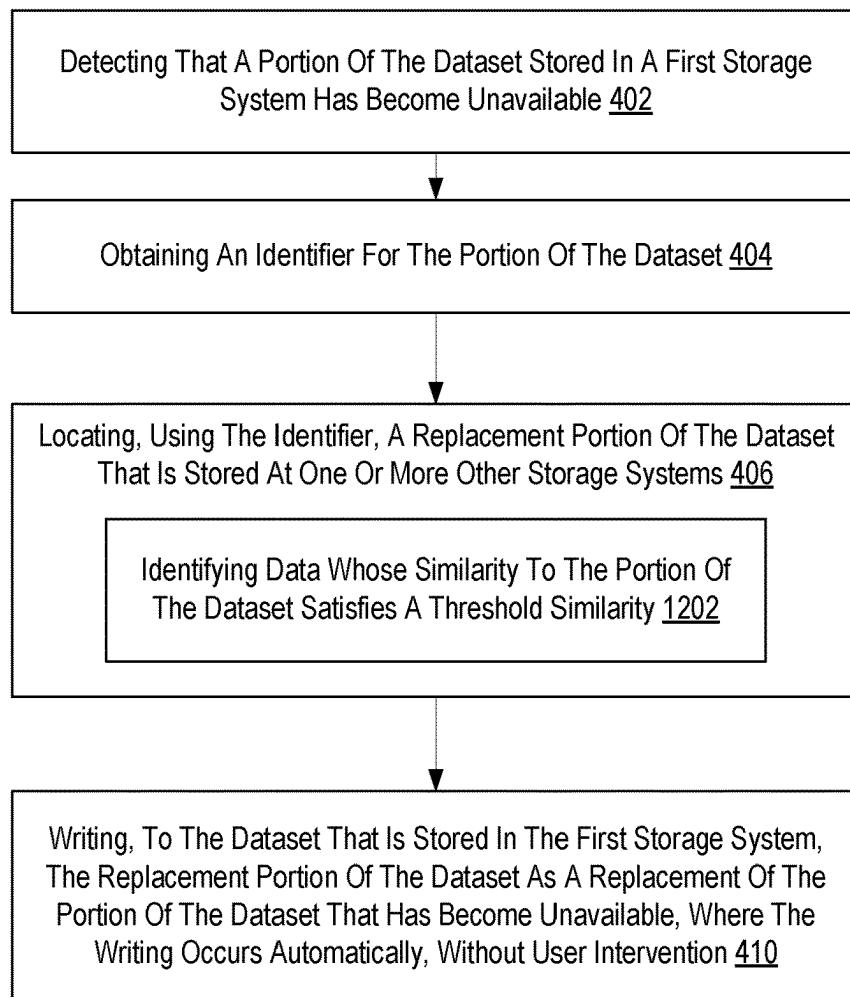


FIG. 12

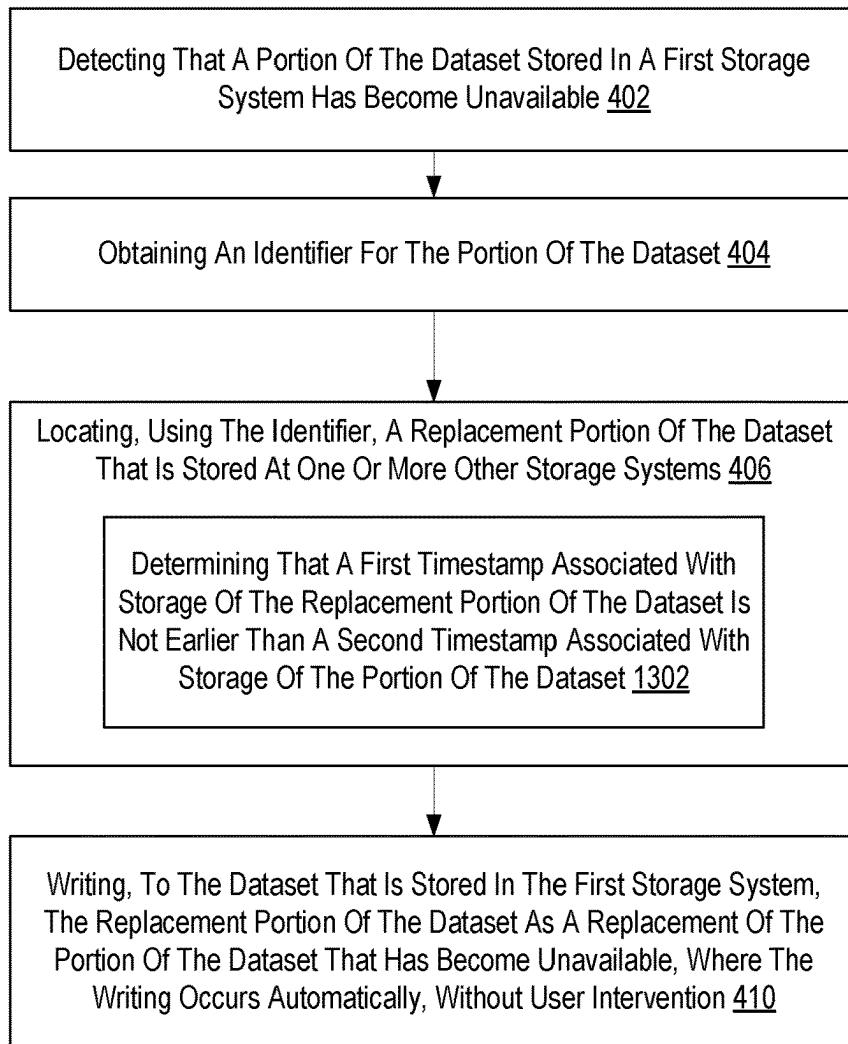


FIG. 13

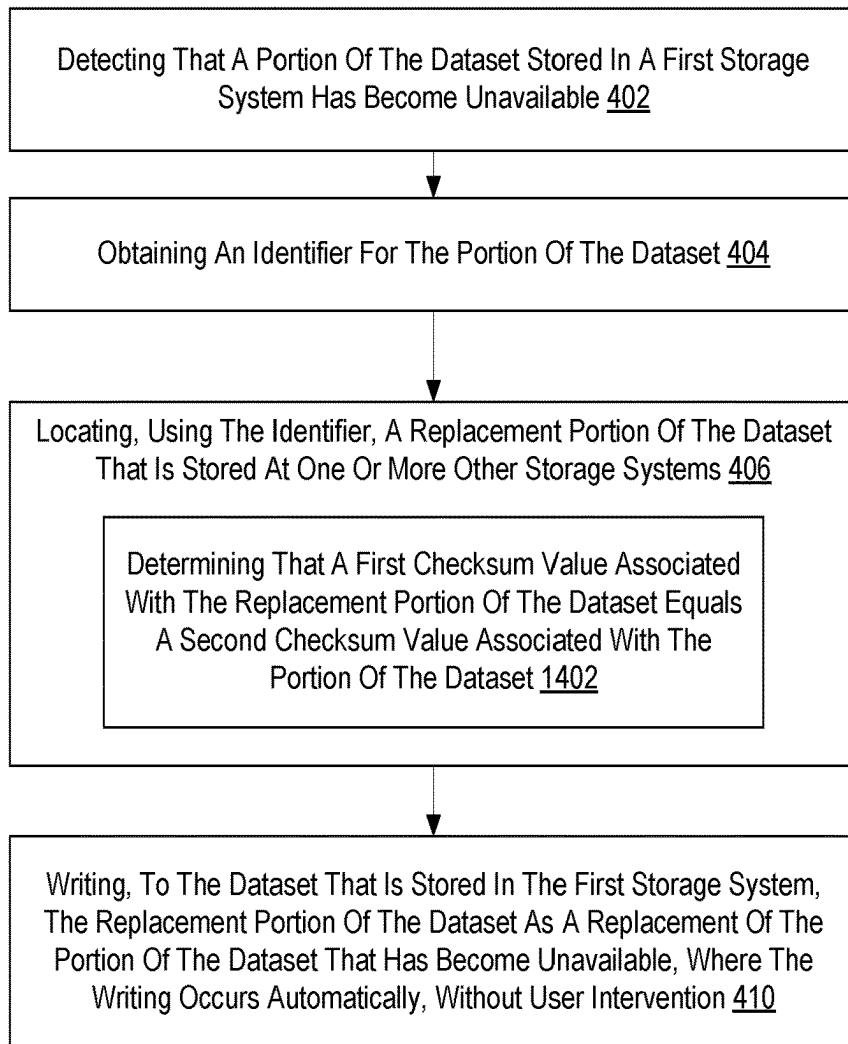


FIG. 14

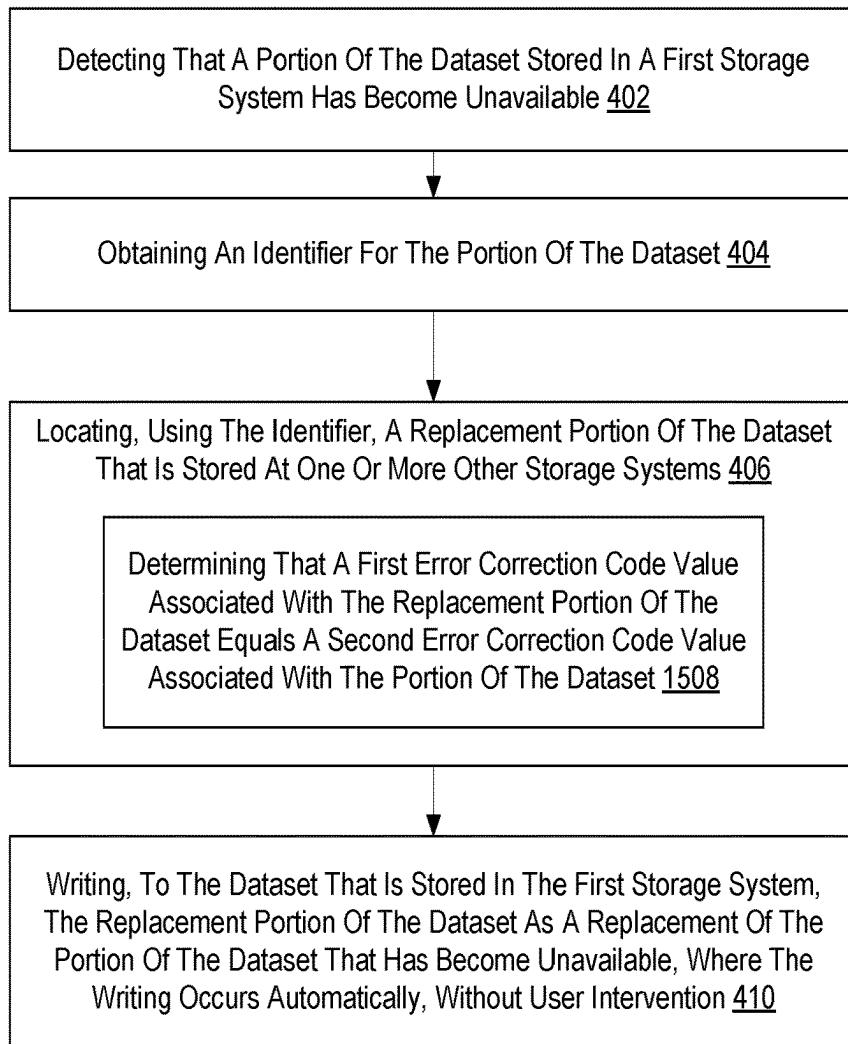


FIG. 15

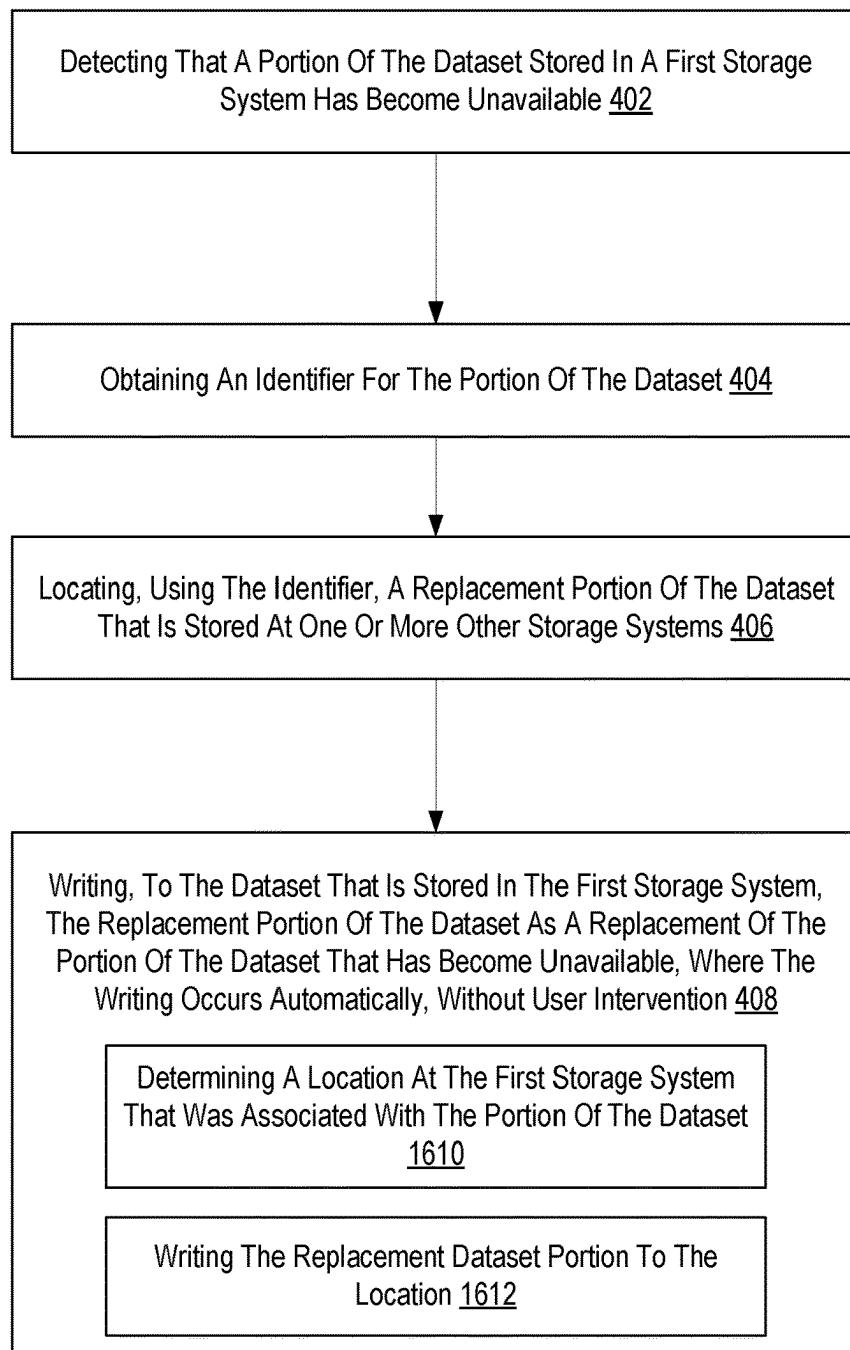


FIG. 16

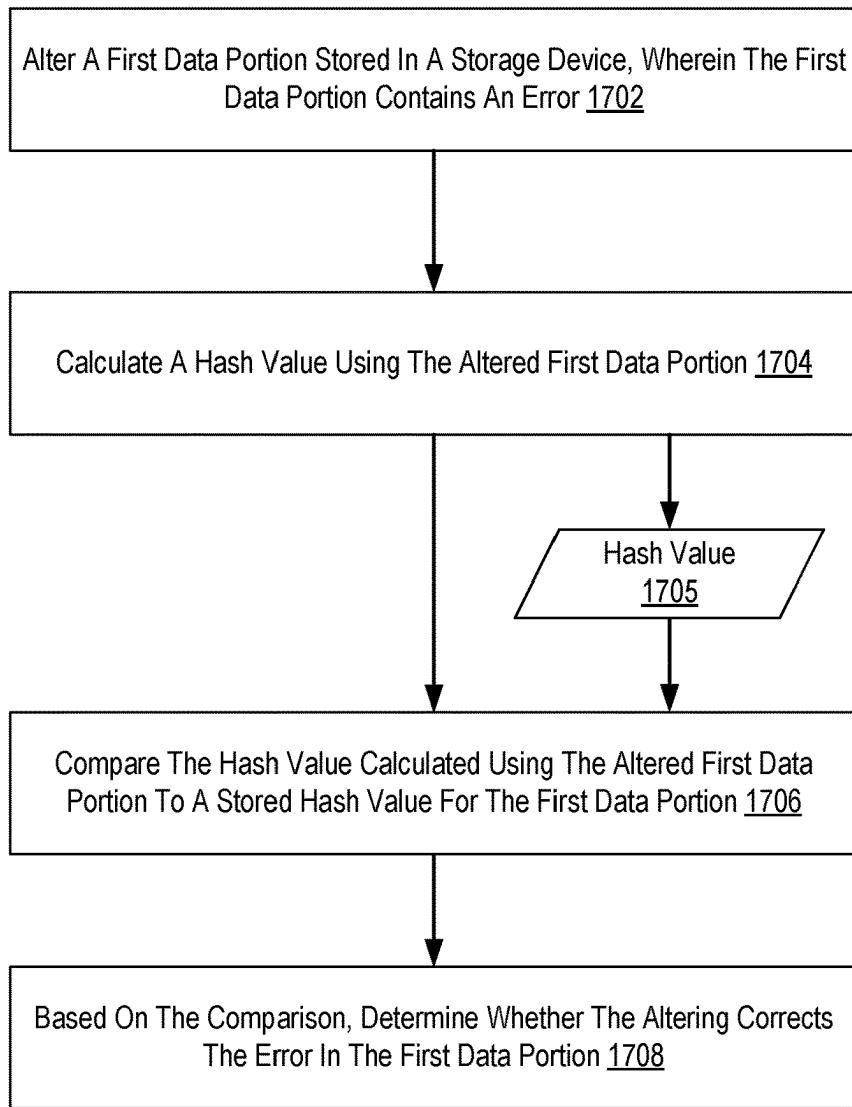


FIG. 17

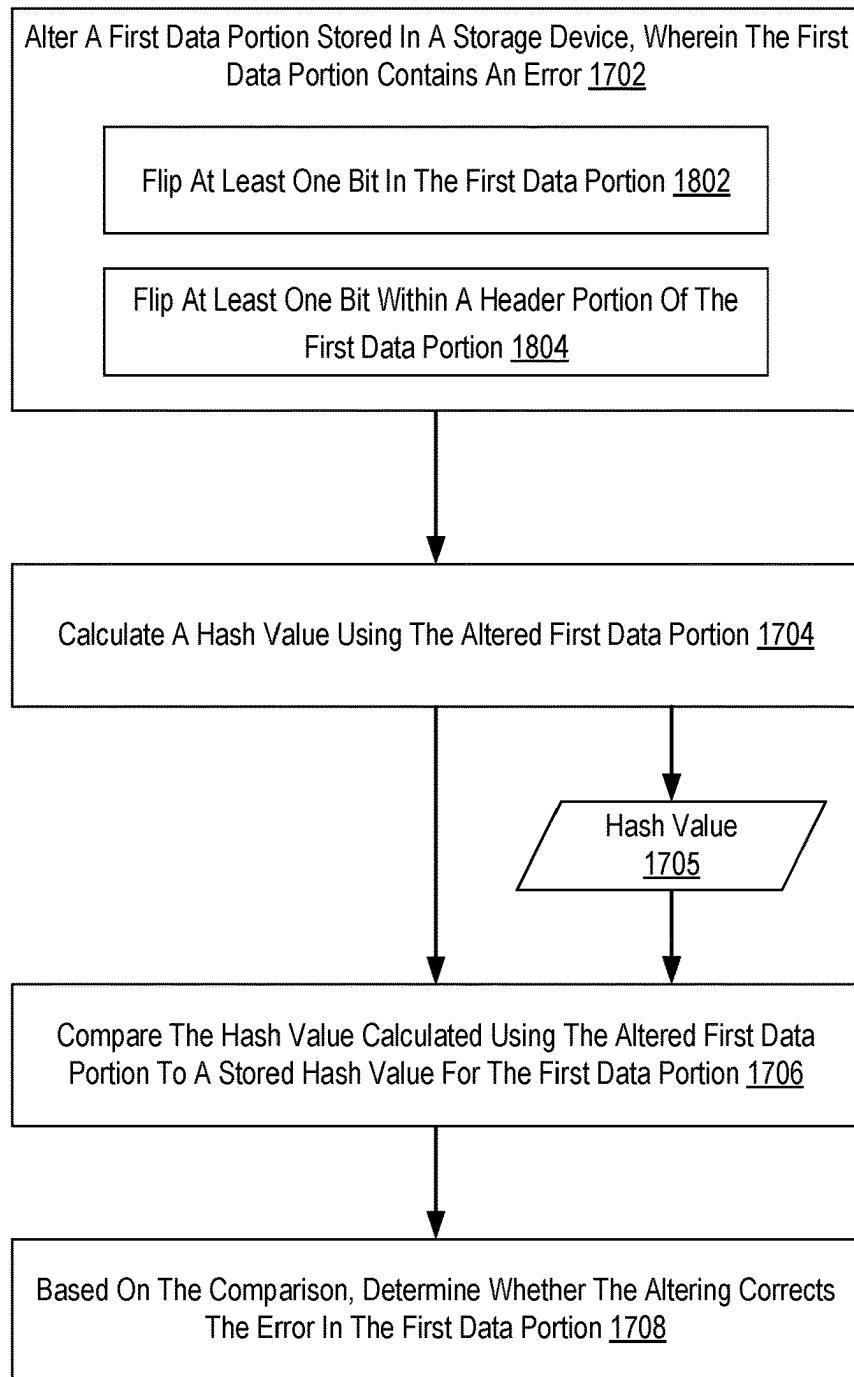


FIG. 18

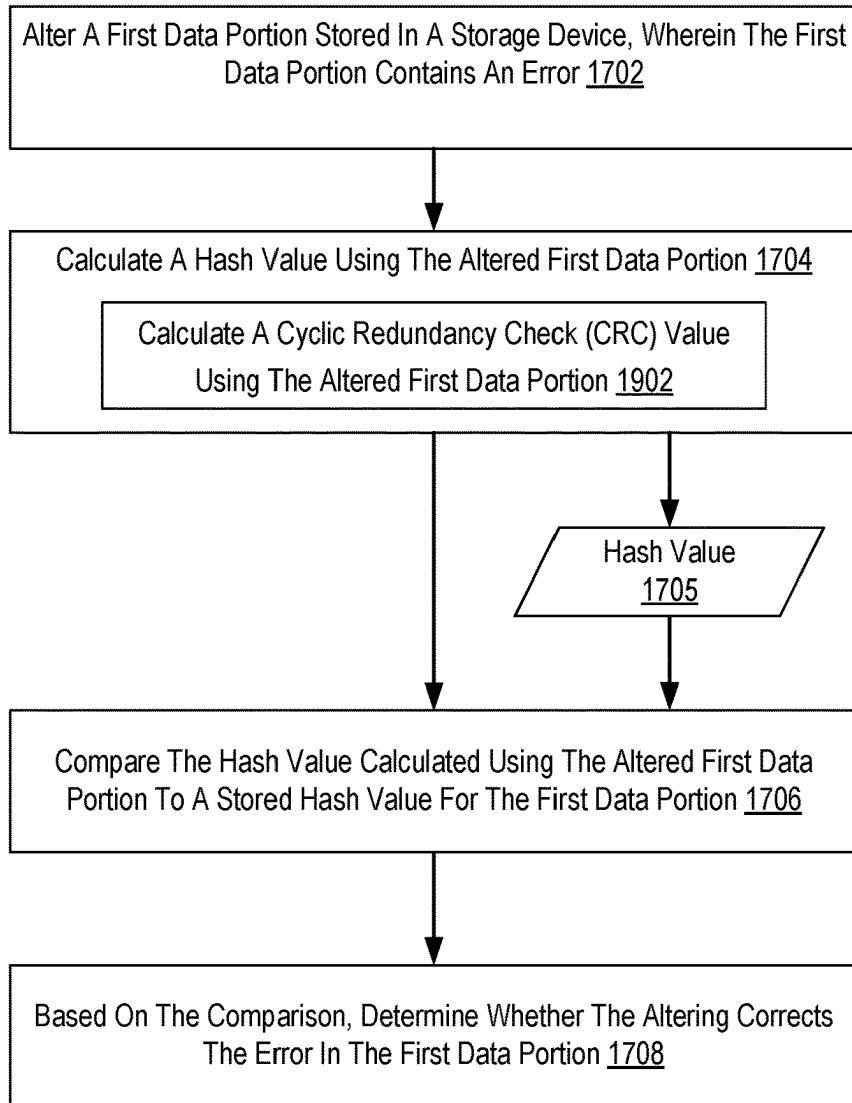


FIG. 19

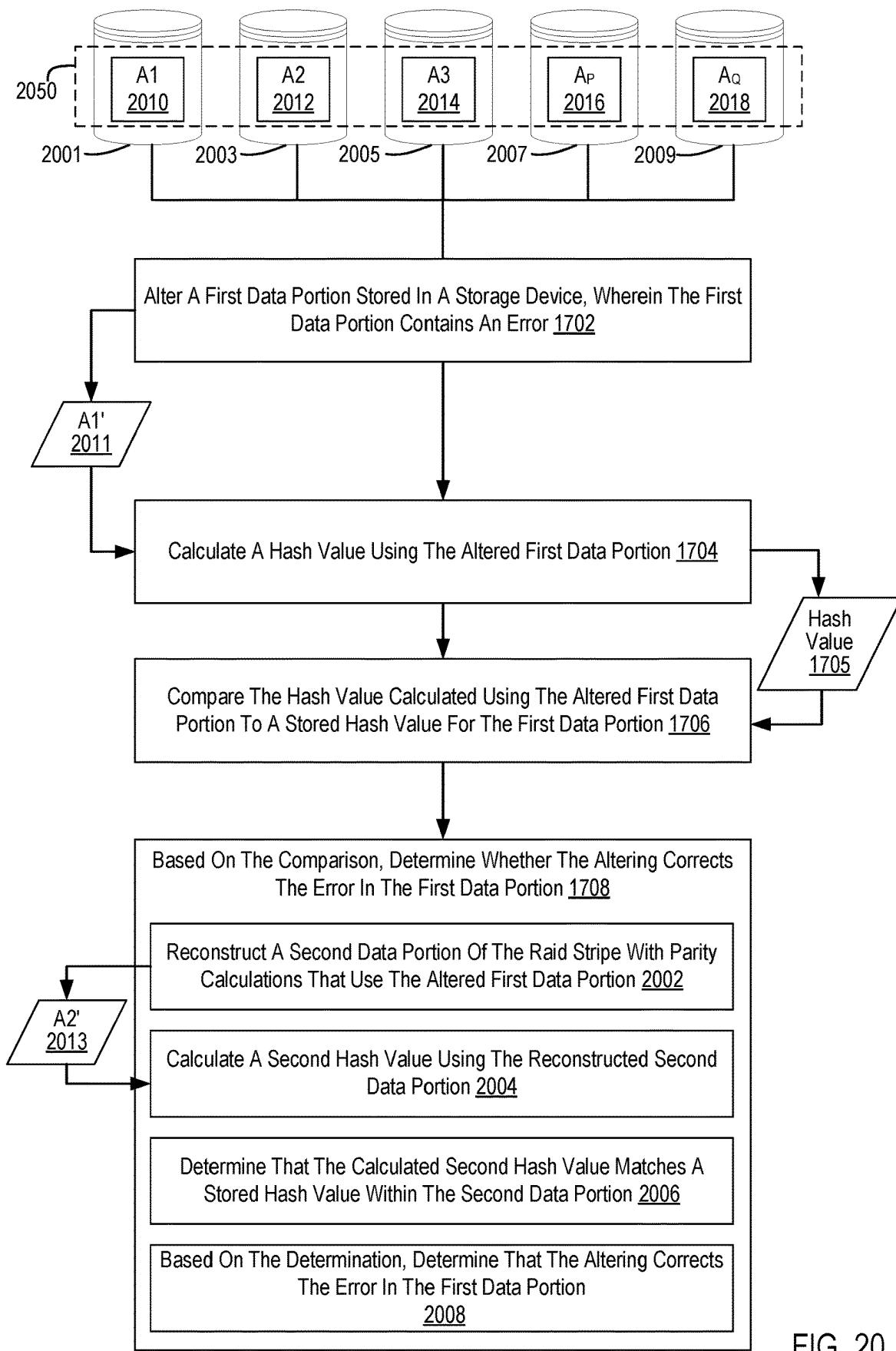


FIG. 20

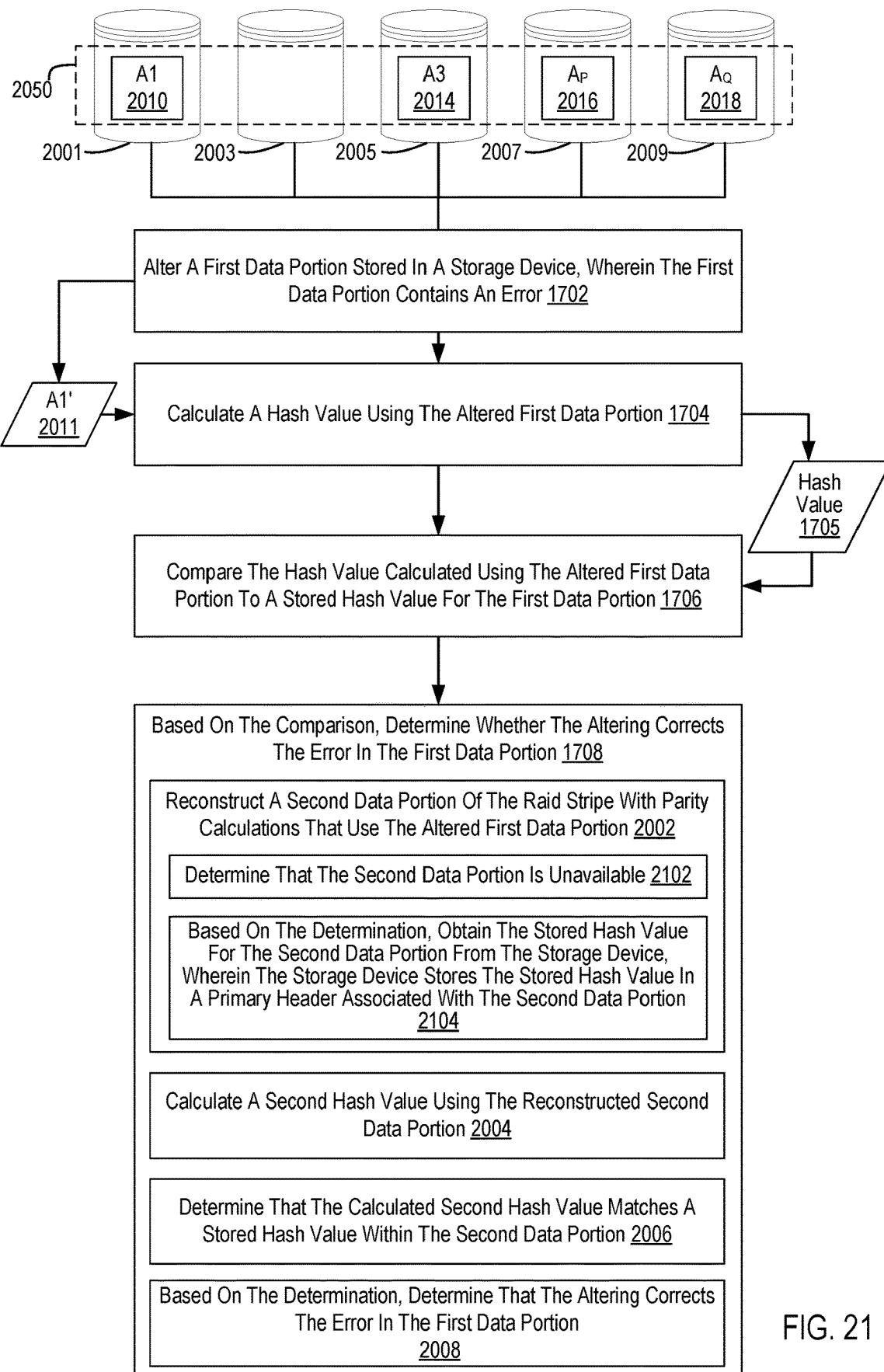


FIG. 21

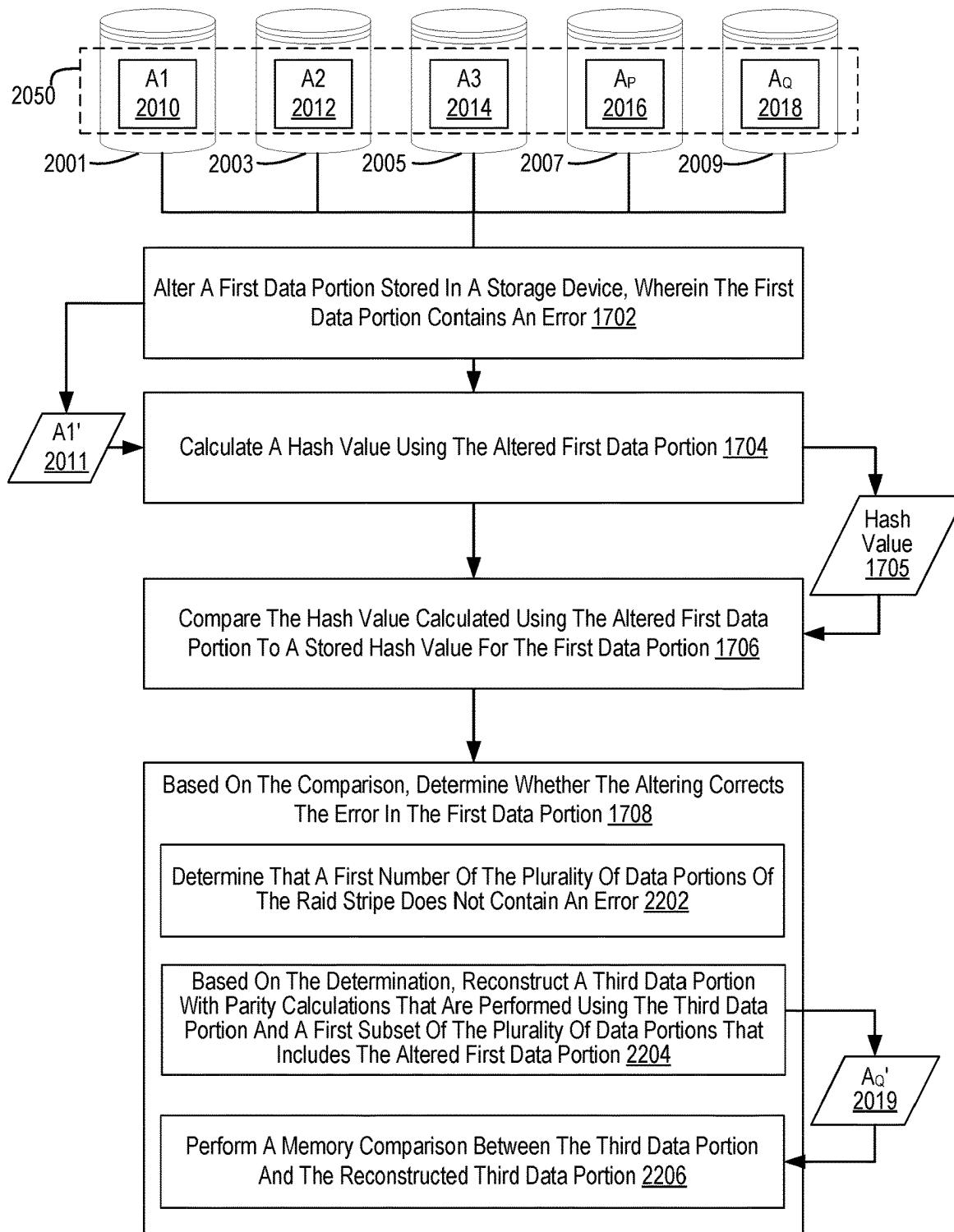


FIG. 22

1
**RECOVER CORRUPTED DATA THROUGH
SPECULATIVE BITFLIP AND
CROSS-VALIDATION**
**CROSS REFERENCE TO RELATED
APPLICATION**

This is a continuation in-part application for patent entitled to a filing date and claiming the benefit of earlier-filed U.S. patent application Ser. No. 17/587,514, filed Jan. 28, 2022, herein incorporated by reference in its entirety.

BRIEF DESCRIPTION OF DRAWINGS

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

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

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

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

FIG. 2A 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.

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

FIG. 2C 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.

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

FIG. 2E 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.

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

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

FIG. 3A 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.

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

FIG. 3C sets forth an example of a cloud-based storage system in accordance with some embodiments of the present disclosure.

FIG. 3D illustrates an exemplary computing device that may be specifically configured to perform one or more of the processes described herein.

FIG. 3E illustrates an example of a fleet of storage systems for providing storage services (also referred to herein as ‘data services’) in accordance with some embodiments.

FIG. 3F illustrates an example container system.

FIG. 4 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure.

2

FIG. 5 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure.

FIG. 6 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure.

FIG. 7 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure.

FIG. 8 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure.

FIG. 9 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure.

FIG. 10 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure.

FIG. 11 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure.

FIG. 12 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure.

FIG. 13 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure.

FIG. 14 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure.

FIG. 15 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure.

FIG. 16 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure.

FIG. 17 sets forth a flow chart illustrating an example method of recovering corrupted data through speculative bitflip and cross-validation in accordance with some embodiments of the present disclosure.

FIG. 18 sets forth a flow chart illustrating an example method of recovering corrupted data through speculative bitflip and cross-validation in accordance with some embodiments of the present disclosure.

FIG. 19 sets forth a flow chart illustrating an example method of recovering corrupted data through speculative bitflip and cross-validation in accordance with some embodiments of the present disclosure.

FIG. 20 sets forth a flow chart illustrating an example method of recovering corrupted data through speculative bitflip and cross-validation in accordance with some embodiments of the present disclosure.

FIG. 21 sets forth a flow chart illustrating an example method of recovering corrupted data through speculative bitflip and cross-validation in accordance with some embodiments of the present disclosure.

FIG. 22 sets forth a flow chart illustrating an example method of recovering corrupted data through speculative bitflip and cross-validation in accordance with some embodiments of the present disclosure.

DESCRIPTION OF EMBODIMENTS

Example methods, apparatus, and products for restoring lost data in accordance with embodiments of the present disclosure are described with reference to the accompanying drawings, beginning with FIG. 1A. FIG. 1A 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.

System 100 includes a number of computing devices 164A-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 164A-B may be coupled for data communications to one or more storage arrays 102A-B through a storage area network (“SAN”) 158 or a local area network (“LAN”) 160.

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 164A-B and storage arrays 102A-B.

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.

Storage arrays 102A-B may provide persistent data storage for the computing devices 164A-B. Storage array 102A may be contained in a chassis (not shown), and storage array 102B may be contained in another chassis (not shown), in some implementations. Storage array 102A and 102B may include one or more storage array controllers 110A-D (also referred to as “controller” herein). A storage array controller 110A-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 110A-D may be configured to carry out various storage tasks. Storage tasks may include writing data received from the computing devices 164A-B to storage array 102A-B, erasing data from storage array 102A-B, retrieving data from storage array 102A-B and providing data to computing devices 164A-B, monitoring and reporting of storage device 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.

Storage array controller 110A-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 110A-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 110A-D may be independently coupled to the LAN 160. In some implementations, storage array controller 110A-D may include an I/O controller or the like that couples the storage array controller 110A-D for data communications, through a midplane (not shown), to a persistent storage resource 170A-B (also referred to as a “storage resource” herein). The persistent storage resource 170A-B may include any number of storage drives 171A-F (also referred to as “storage devices” herein) and any number of non-volatile Random Access Memory (“NVRAM”) devices (not shown).

In some implementations, the NVRAM devices of a persistent storage resource 170A-B may be configured to receive, from the storage array controller 110A-D, data to be stored in the storage drives 171A-F. In some examples, the data may originate from computing devices 164A-B. In some examples, writing data to the NVRAM device may be carried out more quickly than directly writing data to the storage drive 171A-F. In some implementations, the storage array controller 110A-D may be configured to utilize the NVRAM devices as a quickly accessible buffer for data destined to be written to the storage drives 171A-F. Latency for write requests using NVRAM devices as a buffer may be improved relative to a system in which a storage array controller 110A-D writes data directly to the storage drives 171A-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 171A-F.

In some implementations, storage drive 171A-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 171A-F may correspond to non-disk storage media. For example, the storage drive 171A-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 171A-F may include mechanical or spinning hard disk, such as hard-disk drives (“HDD”).

In some implementations, the storage array controllers 110A-D may be configured for offloading device management responsibilities from storage drive 171A-F in storage array 102A-B. For example, storage array controllers 110A-D may manage control information that may describe the state of one or more memory blocks in the storage drives 171A-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 110A-D, the number of program-erase (“PIE”) 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 implemen-

tations, the control information for the storage drives 171A-F may be stored in one or more particular memory blocks of the storage drives 171A-F that are selected by the storage array controller 110A-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 110A-D in conjunction with storage drives 171A-F to quickly identify the memory blocks that contain control information. For example, the storage controllers 110A-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 drives 171A-F.

In some implementations, storage array controllers 110A-D may offload device management responsibilities from storage drives 171A-F of storage array 102A-B by retrieving, from the storage drives 171A-F, control information describing the state of one or more memory blocks in the storage drives 171A-F. Retrieving the control information from the storage drives 171A-F may be carried out, for example, by the storage array controller 110A-D querying the storage drives 171A-F for the location of control information for a particular storage drive 171A-F. The storage drives 171A-F may be configured to execute instructions that enable the storage drives 171A-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 171A-F and may cause the storage drive 171A-F to scan a portion of each memory block to identify the memory blocks that store control information for the storage drives 171A-F. The storage drives 171A-F may respond by sending a response message to the storage array controller 110A-D that includes the location of control information for the storage drive 171A-F. Responsive to receiving the response message, storage array controllers 110A-D may issue a request to read data stored at the address associated with the location of control information for the storage drives 171A-F.

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

In some implementations, storage array 102A-B may implement two or more storage array controllers 110A-D. For example, storage array 102A may include storage array controllers 110A and storage array controllers 110B. At a given instant, a single storage array controller 110A-D (e.g., storage array controller 110A) of a storage system 100 may be designated with primary status (also referred to as "primary controller" herein), and other storage array controllers 110A-D (e.g., storage array controller 110A) 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 170A-B (e.g., writing data to persistent storage resource 170A-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 170A-B when the primary controller has the right. The status of storage array controllers 110A-D may change. For example, storage array controller 110A may be designated with secondary status, and storage array controller 110B may be designated with primary status.

In some implementations, a primary controller, such as storage array controller 110A, may serve as the primary controller for one or more storage arrays 102A-B, and a second controller, such as storage array controller 110B, may serve as the secondary controller for the one or more storage arrays 102A-B. For example, storage array controller 110A may be the primary controller for storage array 102A and storage array 102B, and storage array controller 110B may be the secondary controller for storage array 102A and 102B. In some implementations, storage array controllers 110C and 110D (also referred to as "storage processing modules") may neither have primary or secondary status. Storage array controllers 110C and 110D, implemented as storage processing modules, may act as a communication interface between the primary and secondary controllers (e.g., storage array controllers 110A and 110B, respectively) and storage array 102B. For example, storage array controller 110A of storage array 102A may send a write request, via SAN 158, to storage array 102B. The write request may be received by both storage array controllers 110C and 110D of storage array 102B. Storage array controllers 110C and 110D facilitate the communication, e.g., send the write request to the appropriate storage drive 171A-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.

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

FIG. 1B illustrates an example system for data storage, in accordance with some implementations. Storage array controller 101 illustrated in FIG. 1B may be similar to the storage array controllers 110A-D described with respect to FIG. 1A. In one example, storage array controller 101 may be similar to storage array controller 110A or storage array controller 110B. 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. 1A may be included below to help illustrate features of storage array controller 101.

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.

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 a direct-mapped flash storage system. In one embodiment, a direct-mapped flash storage system is one that addresses data blocks within flash drives directly and without an address translation performed by the storage controllers of the flash drives.

In some implementations, storage array controller **101** includes one or more host bus adapters **103A-C** that are coupled to the processing device **104** via a data communications link **105A-C**. In some implementations, host bus adapters **103A-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 **103A-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 **103A-C** may be coupled to the processing device **104** via a data communications link **105A-C** such as, for example, a PCIe bus.

In some 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.

In some 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.

In some 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.

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.

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.

In some implementations, storage drive **171A-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 some 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 some implementations utilizing an SSD, the natural size may be based on the erase block size of the SSD. In some implementations, the zones of the zoned storage device may be defined during initialization of the zoned storage device. In some implementations, the zones may be defined dynamically as data is written to the zoned storage device.

In some implementations, zones may be heterogeneous, with some zones each being a page group and other zones being multiple page groups. In some implementations, some zones may correspond to an erase block and other zones may correspond to multiple erase blocks. In an implementation, zones may be any combination of differing numbers of pages in page groups and/or erase blocks, for heterogeneous mixes of programming modes, manufacturers, product types and/or product generations of storage devices, as applied to heterogeneous assemblies, upgrades, distributed storages, etc. In some implementations, zones may be defined as having usage characteristics, such as a property of supporting data with particular kinds of longevity (very short lived or very long lived, for example). These properties could be used by a zoned storage device to determine how the zone will be managed over the zone's expected lifetime.

It should be appreciated that a zone is a virtual construct. Any particular zone may not have a fixed location at a storage device. Until allocated, a zone may not have any location at a storage device. A zone may correspond to a number representing a chunk of virtually allocatable space that is the size of an erase block or other block size in various implementations. When the system allocates or opens a zone, zones get allocated to flash or other solid-state storage memory and, as the system writes to the zone, pages are written to that mapped flash or other solid-state storage memory of the zoned storage device. When the system closes the zone, the associated erase block(s) or other sized block(s) are completed. At some point in the future, the system may delete a zone which will free up the zone's allocated space. During its lifetime, a zone may be moved around to different locations of the zoned storage device, e.g., as the zoned storage device does internal maintenance.

In some implementations, the zones of the zoned storage device may be in different states. A zone may be in an empty state in which data has not been stored at the zone. An empty zone may be opened explicitly, or implicitly by writing data to the zone. This is the initial state for zones on a fresh zoned storage device, but may also be the result of a zone reset. In some implementations, an empty zone may have a designated location within the flash memory of the zoned storage device. In an implementation, the location of the empty zone may be chosen when the zone is first opened or first written to (or later if writes are buffered into memory). A zone may be in an open state either implicitly or explicitly, where a zone that is in an open state may be written to store data with write or append commands. In an implementation, a zone that is in an open state may also be written to using a copy command that copies data from a different zone. In some implementations, a zoned storage device may have a limit on the number of open zones at a particular time.

A zone in a closed state is a zone that has been partially written to, but has entered a closed state after issuing an explicit close operation. A zone in a closed state may be left available for future writes, but may reduce some of the run-time overhead consumed by keeping the zone in an open state. In some implementations, a zoned storage device may have a limit on the number of closed zones at a particular time. A zone in a full state is a zone that is storing data and can no longer be written to. A zone may be in a full state either after writes have written data to the entirety of the zone or as a result of a zone finish operation. Prior to a finish operation, a zone may or may not have been completely written. After a finish operation, however, the zone may not be opened or written to further without first performing a zone reset operation.

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.

In some 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 some 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.

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.

5 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.

Advantages of the process being performed only by the 20 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 25 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.

30 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.

40 FIG. 1C 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 45 same, more, or fewer elements configured in the same or different manner in other implementations.

In one embodiment, system 117 includes a dual Peripheral Component Interconnect ('PCI') flash storage device 118 with separately addressable fast write storage. System 117 50 may include a storage device controller 119. In one embodiment, storage device controller 119A-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 120a-n), operatively coupled to various channels of the storage device controller 119. Flash memory devices 120a-n, may be presented to the controller 119A-D as an addressable collection of Flash pages, erase blocks, and/or control elements sufficient to allow the storage device controller 119A-D to program and retrieve various aspects of the Flash. In one embodiment, storage device controller 119A-D may perform operations on flash memory devices 120a-n including 55 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 60

11

Flash memory, controlling voltage levels associated with programming and retrieving contents of Flash cells, etc.

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 119A-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.

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 120a-120n) for sufficient time to write the contents of RAM to Flash memory. In one embodiment, storage device controller 119A-D may write the contents of RAM to Flash Memory if the storage device controller detects loss of external power.

In one embodiment, system 117 includes two data communications links 123a, 123b. In one embodiment, data communications links 123a, 123b may be PCI interfaces. In another embodiment, data communications links 123a, 123b may be based on other communications standards (e.g., HyperTransport, InfiniBand, etc.). Data communications links 123a, 123b may be based on non-volatile memory express ('NVMe') or NVMe over fabrics ('NVMF') specifications that allow external connection to the storage device controller 119A-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.

System 117 may also include an external power source (not shown), which may be provided over one or both data communications links 123a, 123b, 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 119A-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 119A-D may write stored content associated with the addressable fast-write logical storage to Flash memory (e.g., Flash memory 120a-n) for long-term persistent storage.

In one embodiment, the logical device may include some presentation of some or all of the content of the Flash memory devices 120a-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.

12

In one embodiment, the stored energy device 122 may be sufficient to ensure completion of in-progress operations to the Flash memory devices 120a-120n stored energy device 122 may power storage device controller 119A-D and associated Flash memory devices (e.g., 120a-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 120a-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.

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 stored 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.

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

In one embodiment, two storage controllers (e.g., 125a and 125b) 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 125a, 125b may provide services through some number of network interfaces (e.g., 126a-d) to host computers 127a-n outside of the storage system 124. Storage controllers 125a, 125b may provide integrated services or an application entirely within the storage system 124, forming a converged storage and compute system. The storage controllers 125a, 125b may utilize the fast write memory within or across storage devices 119a-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.

In one embodiment, storage controllers 125a, 125b operate as PCI masters to one or the other PCI buses 128a, 128b. In another embodiment, 128a and 128b may be based on other communications standards (e.g., HyperTransport, InfiniBand, etc.). Other storage system embodiments may operate storage controllers 125a, 125b as multi-masters for both PCI buses 128a, 128b. 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 119a may be operable under direction from a storage controller 125a 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. 1C). 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., **128a**, **128b**) from the storage controllers **125a**, **125b**. In one embodiment, a recalculation may include compressing data, attaching indexing or other metadata, combining multiple data segments together, performing erasure code calculations, etc.

In one embodiment, under direction from a storage controller **125a**, **125b**, a storage device controller **119a**, **119b** may be operable to calculate and transfer data to other storage devices from data stored in RAM (e.g., RAM **121** of FIG. 1C) without involvement of the storage controllers **125a**, **125b**. This operation may be used to mirror data stored in one storage controller **125a** to another storage controller **125b**, 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 **129a**, **129b** to the PCI bus **128a**, **128b**.

A storage device controller **119A-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.

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.

In one embodiment, the storage controllers **125a**, **125b** 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 **125a**, **125b** 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.

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.

The embodiments depicted with reference to FIGS. 2A-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.

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.

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 ('MRAM') that substitutes for DRAM and enables a reduced power hold-up apparatus.

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.

FIG. 2A 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.

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.

Referring to FIG. 2A, 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 152 units or storage nodes 150 within the chassis.

FIG. 2B is a block diagram showing a communications interconnect 173 and power distribution bus 172 coupling multiple storage nodes 150. Referring back to FIG. 2A, the communications interconnect 173 can be included in or implemented with the switch fabric 146 in some embodiments. 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. 2B, 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. 2A. In addition, one or more storage nodes 150 may be a compute only storage node as illustrated in FIG. 2B. Authorities 168 are implemented on the non-volatile solid state storage 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 storage 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.

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 storage 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.

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.

With reference to FIGS. 2A and 2B, 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.

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.

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.

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.

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.

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. 2E and 2G) 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.

5 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 10 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. 15 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 20 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 25 **152** unit 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**.

Data and metadata is stored by a set of underlying storage 30 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 35 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 40 some embodiments, low density parity check ('LDPC') 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 45 a log structured layout.

In order to maintain consistency across multiple copies of 50 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. 55 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 60 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 65 of candidate authority owners. A pseudorandom data distri-

bution 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.

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.

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.

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.

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.

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.

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.

FIG. 2C 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 embodiments. 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. 2C, 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. 2C, 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**.

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 non-volatile solid state storage **152** units 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 non-volatile solid state storage **152** units 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).

FIG. 2D shows a storage server environment, which uses embodiments of the storage nodes **150** and storage **152** units of FIGS. 2A-C. In this version, each non-volatile solid state storage **152** unit has a processor such as controller **212** (see FIG. 2C), an FPGA, flash memory **206**, and NVRAM **204** (which is super-capacitor backed DRAM **216**, see FIGS. 2B and 2C) on a PCIe (peripheral component interconnect express) board in a chassis **138** (see FIG. 2A). The non-volatile solid state storage **152** unit 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 non-volatile solid state storage **152** units may fail and the device will continue with no data loss.

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 non-volatile solid state storage **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**. 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 non-volatile solid state storage **152** unit fails, onboard supercapacitors 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**.

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. 2D 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.

FIG. 2E 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. 2A-C in the storage server environment of FIG. 2D. 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).

In the compute and storage planes **256**, **258** of FIG. 2E, 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**.

FIG. 2F 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. 2F. 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.

Still referring to FIG. 2F, 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.

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.

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.

FIG. 2G 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.

- 5 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, 10 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') 15 protocol, one version of which is also known as Common Internet File System ('CIFS'), may be integrated with the storage systems discussed herein. SMB 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 transaction 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, 20 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 25 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. 30 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 35 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 40 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.
- 45
- 50
- 55
- 60
- 65

FIG. 3A 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. 3A may be similar to the storage systems described above with reference to FIGS. 1A-1D and FIGS. 2A-2G. In some embodiments, the storage system 306 depicted in FIG. 3A 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.

In the example depicted in FIG. 3A, the storage system 306 is coupled to the cloud services provider 302 via a data communications link 304. 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.

The cloud services provider 302 depicted in FIG. 3A 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.

In the example depicted in FIG. 3A, 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.

In the example depicted in FIG. 3A, 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.

Although not explicitly depicted in FIG. 3A, 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 system 306 and remote, cloud-based storage that is utilized by the storage system 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.

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. 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.

In the example depicted in FIG. 3A, 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. 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.

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. 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.

Although the example depicted in FIG. 3A 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.

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.

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.

For further explanation, FIG. 3B 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. 3B may be similar to the storage systems described above with reference to FIGS. 1A-1D and FIGS. 2A-2G as the storage system may include many of the components described above.

The storage system 306 depicted in FIG. 3B 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 (STY) 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. 3A 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.

The storage resources 308 depicted in FIG. 3B 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.

The storage resources 308 depicted in FIG. 3B 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.

The example storage system 306 depicted in FIG. 3B 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.

The example storage system 306 depicted in FIG. 3B 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.

The example storage system 306 depicted in FIG. 3B 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, storage resources within the storage system may be utilized as a read cache, or tiering may be achieved within the storage systems by placing data within the storage system in accordance with one or more tiering policies.

The storage system 306 depicted in FIG. 3B 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.

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 10 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.

15 The storage system 306 depicted in FIG. 3B also includes processing resources 312 that may be 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 20 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.

25 The storage system 306 depicted in FIG. 3B also includes 30 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 35 resources 312 within the storage system 306 are useful in carrying out various data protection techniques. 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 data archiving, data backup, data replication, data snapshotting, data and database cloning, and other data protection techniques.

40 The software resources 314 may also include software 45 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.

45 The software resources 314 may also include software 50 that is useful in facilitating and optimizing I/O operations that are directed to the storage system 306. For example, the software resources 314 may include software modules that perform 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 55 that perform other functions. Such software resources 314 may be embodied as one or more software containers or in many other ways.

For further explanation, FIG. 3C 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. 3C, 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.

The cloud-based storage system 318 depicted in FIG. 3C 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. For example, each of the cloud computing instances 320, 322 may execute on an Azure VM, where each Azure VM may include high speed temporary storage that may be leveraged as a cache (e.g., as a read cache). 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.

In the example method depicted in FIG. 3C, 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 110A, 110B in FIG. 1A described above such as writing data 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, monitoring and reporting of storage device 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. 3C may include identical source code that is executed within different cloud computing instances 320, 322 such as distinct EC2 instances.

Readers will appreciate that other embodiments that do not include a primary and secondary controller 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.

The cloud-based storage system 318 depicted in FIG. 3C includes cloud computing instances 340a, 340b, 340n with local storage 330, 334, 338. The cloud computing instances 340a, 340b, 340n 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 340a, 340b, 340n of FIG. 3C may differ from the cloud computing instances 320, 322 described above as the cloud computing instances 340a, 340b, 340n of FIG. 3C 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 340a, 340b, 340n 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.

In the example depicted in FIG. 3C, each of the cloud computing instances 340a, 340b, 340n with local storage 330, 334, 338 can include a software daemon 328, 332, 336 that, when executed by a cloud computing instance 340a, 340b, 340n can present itself to the storage controller applications 324, 326 as if the cloud computing instance 340a, 340b, 340n 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 340a, 340b, 340n with local storage 330, 334, 338 may utilize iSCSI, NVMe over TCP, messaging, a custom protocol, or in some other mechanism.

In the example depicted in FIG. 3C, each of the cloud computing instances 340a, 340b, 340n 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 such as, for example, as Amazon Elastic Block Store ('EBS') volumes. 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 computing instance 340a, 340b, 340n 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 computing instance 340a, 340b, 340n. 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 340a, 340b, 340n 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. In yet another embodiment, high performance block storage resources such as one or more Azure Ultra Disks may be utilized as the NVRAM.

When a request to write data is received by a particular cloud computing instance 340a, 340b, 340n with local storage 330, 334, 338, the software daemon 328, 332, 336 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 resources, but the software daemon 328, 332, 336 may also be configured to write the data to cloud-based object storage 348 that is attached to the particular cloud computing instance 340a, 340b, 340n. The cloud-based object storage 348 that is attached to the particular cloud computing instance 340a, 340b, 340n may be embodied, for example, as Amazon Simple Storage Service ('S3'). 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 340a, 340b, 340n and the cloud-based object storage 348. In other embodiments, rather than using both the cloud computing instances 340a, 340b, 340n with local storage 330, 334, 338 (also referred to herein as 'virtual drives') and the cloud-based object storage 348 to store data, a persistent storage layer may be implemented in other ways. For example, one or more Azure Ultra disks may be used to persistently store data (e.g., after the data has been written to the NVRAM layer). In an embodiment where one or more Azure Ultra disks may be used to persistently store data, the usage of a cloud-based object storage 348 may be eliminated such that data is only stored persistently in the Azure Ultra disks without also writing the data to an object storage layer.

While the local storage 330, 334, 338 resources and the block storage 342, 344, 346 resources that are utilized by the cloud computing instances 340a, 340b, 340n may support block-level access, the cloud-based object storage 348 that is attached to the particular cloud computing instance 340a, 340b, 340n supports only object-based access. The software daemon 328, 332, 336 may therefore 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 340a, 340b, 340n.

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 340a, 340b, 340n. 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 340a, 340b, 340n 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 340a, 340b, 340n without requiring the cloud computing instances 340a, 340b, 340n 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 340a, 340b, 340n. 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 340a, 340b, 340n.

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 340a, 340b, 340n 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 340a, 340b, 340n 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 340a, 340b, 340n 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.

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. For example, if 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, a 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.

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.

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 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.

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.

Readers will appreciate that the storage systems described in this disclosure may be useful for supporting various types of software applications. In fact, the storage systems may be ‘application aware’ in the sense that the storage systems may obtain, maintain, or otherwise have access to information describing connected applications (e.g., applications that utilize the storage systems) to optimize the operation of the storage system based on intelligence about the applications and their utilization patterns. For example, the storage system may optimize data layouts, optimize caching behaviors, optimize ‘QoS’ levels, or perform some other optimization that is designed to improve the storage performance that is experienced by the application.

As an example of one type of application that may be supported by the storage systems described herein, the storage system 306 may be useful in supporting artificial intelligence (‘AI’) applications, database applications, XOps projects (e.g., DevOps projects, DataOps projects, MLOps projects, ModelOps projects, PlatformOps 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.

In view of the fact that the storage systems include 5 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.

Such AI applications may enable devices to perceive their 20 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 25 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.

In addition to the resources already described, the storage 30 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 35 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 also be independently scalable.

As described above, the storage systems described herein 40 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 45 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.

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 AI techniques have 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.

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.

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. 5 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.

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.

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.

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.

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 and 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

41

storage systems described herein may be leveraged to support on-chain storage of data as well as off-chain storage of data.

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).

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.

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.

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

42

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.

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 ONNX 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.

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 stored 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.

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.

The storage systems described above may alone, or in combination with other computing resources, serve 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.

The storage systems described above may also be optimized for use in big data analytics, including being leveraged as part of a composable data analytics pipeline where containerized analytics architectures, for example, make analytics capabilities more composable. 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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

The storage systems described herein may be used to form a data lake. A data lake may operate as the first place that an organization's data flows to, where such data may be in a raw format. Metadata tagging may be implemented to facilitate searches of data elements in the data lake, especially in embodiments where the data lake contains multiple stores of data, in formats not easily accessible or readable (e.g., unstructured data, semi-structured data, structured data). From the data lake, data may go downstream to a data warehouse where data may be stored in a more processed, packaged, and consumable format. The storage systems described above may also be used to implement such a data warehouse. In addition, a data mart or data hub may allow for data that is even more easily consumed, where the storage systems described above may also be used to provide the underlying storage resources necessary for a data mart or data hub. In embodiments, queries the data lake may require a schema-on-read approach, where data is applied to a plan or schema as it is pulled out of a stored location, rather than as it goes into the stored location.

The storage systems described herein may also be configured implement a recovery point objective ('RPO'), which may be established by a user, established by an administrator, established as a system default, established as part of a storage class or service that the storage system is participating in the delivery of, or in some other way. A "recovery point objective" is a goal for the maximum time difference between the last update to a source dataset and the last recoverable replicated dataset update that would be correctly recoverable, given a reason to do so, from a continuously or frequently updated copy of the source dataset. An update is correctly recoverable if it properly takes into account all updates that were processed on the source dataset prior to the last recoverable replicated dataset update.

In synchronous replication, the RPO would be zero, meaning that under normal operation, all completed updates on the source dataset should be present and correctly recoverable on the copy dataset. In best effort nearly synchronous replication, the RPO can be as low as a few seconds. In snapshot-based replication, the RPO can be roughly calculated as the interval between snapshots plus the time to transfer the modifications between a previous already transferred snapshot and the most recent to-be-replicated snapshot.

If updates accumulate faster than they are replicated, then an RPO can be missed. If more data to be replicated accumulates between two snapshots, for snapshot-based replication, than can be replicated between taking the snapshot and replicating that snapshot's cumulative updates to

the copy, then the RPO can be missed. If, again in snapshot-based replication, data to be replicated accumulates at a faster rate than could be transferred in the time between subsequent snapshots, then replication can start to fall further behind which can extend the miss between the expected recovery point objective and the actual recovery point that is represented by the last correctly replicated update.

The storage systems described above may also be part of a shared nothing storage cluster. In a shared nothing storage cluster, each node of the cluster has local storage and communicates with other nodes in the cluster through networks, where the storage used by the cluster is (in general) provided only by the storage connected to each individual node. A collection of nodes that are synchronously replicating a dataset may be one example of a shared nothing storage cluster, as each storage system has local storage and communicates to other storage systems through a network, where those storage systems do not (in general) use storage from somewhere else that they share access to through some kind of interconnect. In contrast, some of the storage systems described above are themselves built as a shared-storage cluster, since there are drive shelves that are shared by the paired controllers. Other storage systems described above, however, are built as a shared nothing storage cluster, as all storage is local to a particular node (e.g., a blade) and all communication is through networks that link the compute nodes together.

In other embodiments, other forms of a shared nothing storage cluster can include embodiments where any node in the cluster has a local copy of all storage they need, and where data is mirrored through a synchronous style of replication to other nodes in the cluster either to ensure that the data isn't lost or because other nodes are also using that storage. In such an embodiment, if a new cluster node needs some data, that data can be copied to the new node from other nodes that have copies of the data.

In some embodiments, mirror-copy-based shared storage clusters may store multiple copies of all the cluster's stored data, with each subset of data replicated to a particular set of nodes, and different subsets of data replicated to different sets of nodes. In some variations, embodiments may store all of the cluster's stored data in all nodes, whereas in other variations nodes may be divided up such that a first set of nodes will all store the same set of data and a second, different set of nodes will all store a different set of data.

Readers will appreciate that RAFT-based databases (e.g., etcd) may operate like shared-nothing storage clusters where all RAFT nodes store all data. The amount of data stored in a RAFT cluster, however, may be limited so that extra copies don't consume too much storage. A container server cluster might also be able to replicate all data to all cluster nodes, presuming the containers don't tend to be too large and their bulk data (the data manipulated by the applications that run in the containers) is stored elsewhere such as in an S3 cluster or an external file server. In such an example, the container storage may be provided by the cluster directly through its shared-nothing storage model, with those containers providing the images that form the execution environment for parts of an application or service.

For further explanation, FIG. 3D 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. 3D, 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. 3D, the components illustrated in FIG. 3D are not intended to be limiting. Additional or alternative components may be used in other embodiments. Components of computing device **350** shown in FIG. 3D will now be described in additional detail.

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.

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**.

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**.

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.

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**.

For further explanation, FIG. 3E illustrates an example of a fleet of storage systems **376** for providing storage services (also referred to herein as ‘data services’). The fleet of storage systems **376** depicted in FIG. 3 includes a plurality of storage systems **374a, 374b, 374c, 374d, 374n** that may each be similar to the storage systems described herein. The storage systems **374a, 374b, 374c, 374d, 374n** in the fleet of storage systems **376** may be embodied as identical storage systems or as different types of storage systems. For example, two of the storage systems **374a, 374n** depicted in FIG. 3E are depicted as being cloud-based storage systems, as the resources that collectively form each of the storage

systems **374a, 374n** are provided by distinct cloud services providers **370, 372**. For example, the first cloud services provider **370** may be Amazon AWS™ whereas the second cloud services provider **372** is Microsoft Azure™, although in other embodiments one or more public clouds, private clouds, or combinations thereof may be used to provide the underlying resources that are used to form a particular storage system in the fleet of storage systems **376**.

The example depicted in FIG. 3E includes an edge management service **366** for delivering storage services in accordance with some embodiments of the present disclosure. The storage services (also referred to herein as ‘data services’) that are delivered may include, for example, services to provide a certain amount of storage to a consumer, services to provide storage to a consumer in accordance with a predetermined service level agreement, services to provide storage to a consumer in accordance with predetermined regulatory requirements, and many others.

The edge management service **366** depicted in FIG. 3E may be embodied, for example, as one or more modules of computer program instructions executing on computer hardware such as one or more computer processors. Alternatively, the edge management service **366** may be embodied as one or more modules of computer program instructions executing on a virtualized execution environment such as one or more virtual machines, in one or more containers, or in some other way. In other embodiments, the edge management service **366** may be embodied as a combination of the embodiments described above, including embodiments where the one or more modules of computer program instructions that are included in the edge management service **366** are distributed across multiple physical or virtual execution environments.

The edge management service **366** may operate as a gateway for providing storage services to storage consumers, where the storage services leverage storage offered by one or more storage systems **374a, 374b, 374c, 374d, 374n**. For example, the edge management service **366** may be configured to provide storage services to host devices **378a, 378b, 378c, 378d, 378n** that are executing one or more applications that consume the storage services. In such an example, the edge management service **366** may operate as a gateway between the host devices **378a, 378b, 378c, 378d, 378n** and the storage systems **374a, 374b, 374c, 374d, 374n**, rather than requiring that the host devices **378a, 378b, 378c, 378d, 378n** directly access the storage systems **374a, 374b, 374c, 374d, 374n**.

The edge management service **366** of FIG. 3E exposes a storage services module **364** to the host devices **378a, 378b, 378c, 378d, 378n** of FIG. 3E, although in other embodiments the edge management service **366** may expose the storage services module **364** to other consumers of the various storage services. The various storage services may be presented to consumers via one or more user interfaces, via one or more APIs, or through some other mechanism provided by the storage services module **364**. As such, the storage services module **364** depicted in FIG. 3E may be embodied as one or more modules of computer program instructions executing on physical hardware, on a virtualized execution environment, or combinations thereof, where executing such modules causes enables a consumer of storage services to be offered, select, and access the various storage services.

The edge management service **366** of FIG. 3E also includes a system management services module **368**. The system management services module **368** of FIG. 3E includes one or more modules of computer program instruc-

tions that, when executed, perform various operations in coordination with the storage systems 374a, 374b, 374c, 374d, 374n to provide storage services to the host devices 378a, 378b, 378c, 378d, 378n. The system management services module 368 may be configured, for example, to perform tasks such as provisioning storage resources from the storage systems 374a, 374b, 374c, 374d, 374n via one or more APIs exposed by the storage systems 374a, 374b, 374c, 374d, 374n, migrating datasets or workloads amongst the storage systems 374a, 374b, 374c, 374d, 374n via one or more APIs exposed by the storage systems 374a, 374b, 374c, 374d, 374n, setting one or more tunable parameters (i.e., one or more configurable settings) on the storage systems 374a, 374b, 374c, 374d, 374n via one or more APIs exposed by the storage systems 374a, 374b, 374c, 374d, 374n, and so on. For example, many of the services described below relate to embodiments where the storage systems 374a, 374b, 374c, 374d, 374n are configured to operate in some way. In such examples, the system management services module 368 may be responsible for using APIs (or some other mechanism) provided by the storage systems 374a, 374b, 374c, 374d, 374n to configure the storage systems 374a, 374b, 374c, 374d, 374n to operate in the ways described below.

In addition to configuring the storage systems 374a, 374b, 374c, 374d, 374n, the edge management service 366 itself may be configured to perform various tasks required to provide the various storage services. Consider an example in which the storage service includes a service that, when selected and applied, causes personally identifiable information ('PII') contained in a dataset to be obfuscated when the dataset is accessed. In such an example, the storage systems 374a, 374b, 374c, 374d, 374n may be configured to obfuscate PII when servicing read requests directed to the dataset. Alternatively, the storage systems 374a, 374b, 374c, 374d, 374n may service reads by returning data that includes the PII, but the edge management service 366 itself may obfuscate the PII as the data is passed through the edge management service 366 on its way from the storage systems 374a, 374b, 374c, 374d, 374n to the host devices 378a, 378b, 378c, 378d, 378n.

The storage systems 374a, 374b, 374c, 374d, 374n depicted in FIG. 3E may be embodied as one or more of the storage systems described above with reference to FIGS. 1A-3D, including variations thereof. In fact, the storage systems 374a, 374b, 374c, 374d, 374n may serve as a pool of storage resources where the individual components in that pool have different performance characteristics, different storage characteristics, and so on. For example, one of the storage systems 374a may be a cloud-based storage system, another storage system 374b may be a storage system that provides block storage, another storage system 374c may be a storage system that provides file storage, another storage system 374d may be a relatively high-performance storage system while another storage system 374n may be a relatively low-performance storage system, and so on. In alternative embodiments, only a single storage system may be present.

The storage systems 374a, 374b, 374c, 374d, 374n depicted in FIG. 3E may also be organized into different failure domains so that the failure of one storage system 374a should be totally unrelated to the failure of another storage system 374b. For example, each of the storage systems may receive power from independent power systems, each of the storage systems may be coupled for data communications over independent data communications networks, and so on. Furthermore, the storage systems in a

first failure domain may be accessed via a first gateway whereas storage systems in a second failure domain may be accessed via a second gateway. For example, the first gateway may be a first instance of the edge management service 366 and the second gateway may be a second instance of the edge management service 366, including embodiments where each instance is distinct, or each instance is part of a distributed edge management service 366.

As an illustrative example of available storage services, storage services may be presented to a user that are associated with different levels of data protection. For example, storage services may be presented to the user that, when selected and enforced, guarantee the user that data associated with that user will be protected such that various recovery point objectives ('RPO') can be guaranteed. A first available storage service may ensure, for example, that some dataset associated with the user will be protected such that any data that is more than 5 seconds old can be recovered in the event of a failure of the primary data store whereas a second available storage service may ensure that the dataset that is associated with the user will be protected such that any data that is more than 5 minutes old can be recovered in the event of a failure of the primary data store.

An additional example of storage services that may be presented to a user, selected by a user, and ultimately applied to a dataset associated with the user can include one or more data compliance services. Such data compliance services may be embodied, for example, as services that may be provided to consumers (i.e., a user) the data compliance services to ensure that the user's datasets are managed in a way to adhere to various regulatory requirements. For example, one or more data compliance services may be offered to a user to ensure that the user's datasets are managed in a way so as to adhere to the General Data Protection Regulation ('GDPR'), one or more data compliance services may be offered to a user to ensure that the user's datasets are managed in a way so as to adhere to the Sarbanes-Oxley Act of 2002 ('SOX'), or one or more data compliance services may be offered to a user to ensure that the user's datasets are managed in a way so as to adhere to some other regulatory act. In addition, the one or more data compliance services may be offered to a user to ensure that the user's datasets are managed in a way so as to adhere to some non-governmental guidance (e.g., to adhere to best practices for auditing purposes), the one or more data compliance services may be offered to a user to ensure that the user's datasets are managed in a way so as to adhere to a particular clients or organizations requirements, and so on.

In order to provide this particular data compliance service, the data compliance service may be presented to a user (e.g., via a GUI) and selected by the user. In response to receiving the selection of the particular data compliance service, one or more storage services policies may be applied to a dataset associated with the user to carry out the particular data compliance service. For example, a storage services policy may be applied requiring that the dataset be encrypted prior to being stored in a storage system, prior to being stored in a cloud environment, or prior to being stored elsewhere. In order to enforce this policy, a requirement may be enforced not only requiring that the dataset be encrypted when stored, but a requirement may be put in place requiring that the dataset be encrypted prior to transmitting the dataset (e.g., sending the dataset to another party). In such an example, a storage services policy may also be put in place requiring that any encryption keys used to encrypt the dataset are not stored on the same system that stores the dataset itself.

Readers will appreciate that many other forms of data compliance services may be offered and implemented in accordance with embodiments of the present disclosure.

The storage systems 374a, 374b, 374c, 374d, 374n in the fleet of storage systems 376 may be managed collectively, for example, by one or more fleet management modules. The fleet management modules may be part of or separate from the system management services module 368 depicted in FIG. 3E. The fleet management modules may perform tasks such as monitoring the health of each storage system in the fleet, initiating updates or upgrades on one or more storage systems in the fleet, migrating workloads for load balancing or other performance purposes, and many other tasks. As such, and for many other reasons, the storage systems 374a, 374b, 374c, 374d, 374n may be coupled to each other via one or more data communications links in order to exchange data between the storage systems 374a, 374b, 374c, 374d, 374n.

In some embodiments, one or more storage systems or one or more elements of storage systems (e.g., features, services, operations, components, etc. of storage systems), such as any of the illustrative storage systems or storage system elements described herein may be implemented in one or more container systems. A container system may include any system that supports execution of one or more containerized applications or services. Such a service may be software deployed as infrastructure for building applications, for operating a run-time environment, and/or as infrastructure for other services. In the discussion that follows, descriptions of containerized applications generally apply to containerized services as well.

A container may combine one or more elements of a containerized software application together with a runtime environment for operating those elements of the software application bundled into a single image. For example, each such container of a containerized application may include executable code of the software application and various dependencies, libraries, and/or other components, together with network configurations and configured access to additional resources, used by the elements of the software application within the particular container in order to enable operation of those elements. A containerized application can be represented as a collection of such containers that together represent all the elements of the application combined with the various run-time environments needed for all those elements to run. As a result, the containerized application may be abstracted away from host operating systems as a combined collection of lightweight and portable packages and configurations, where the containerized application may be uniformly deployed and consistently executed in different computing environments that use different container-compatible operating systems or different infrastructures. In some embodiments, a containerized application shares a kernel with a host computer system and executes as an isolated environment (an isolated collection of files and directories, processes, system and network resources, and configured access to additional resources and capabilities) that is isolated by an operating system of a host system in conjunction with a container management framework. When executed, a containerized application may provide one or more containerized workloads and/or services.

The container system may include and/or utilize a cluster of nodes. For example, the container system may be configured to manage deployment and execution of containerized applications on one or more nodes in a cluster. The containerized applications may utilize resources of the nodes, such as memory, processing and/or storage resources

provided and/or accessed by the nodes. The storage resources may include any of the illustrative storage resources described herein and may include on-node resources such as a local tree of files and directories, off-node resources such as external networked file systems, databases or object stores, or both on-node and off-node resources. Access to additional resources and capabilities that could be configured for containers of a containerized application could include specialized computation capabilities such as GPUs and AI/ML engines, or specialized hardware such as sensors and cameras.

In some embodiments, the container system may include a container orchestration system (which may also be referred to as a container orchestrator, a container orchestration platform, etc.) designed to make it reasonably simple and for many use cases automated to deploy, scale, and manage containerized applications. In some embodiments, the container system may include a storage management system configured to provision and manage storage resources (e.g., virtual volumes) for private or shared use by cluster nodes and/or containers of containerized applications.

FIG. 3F illustrates an example container system 380. In this example, the container system 380 includes a container storage system 381 that may be configured to perform one or more storage management operations to organize, provision, and manage storage resources for use by one or more containerized applications 382-1 through 382-L of container system 380. In particular, the container storage system 381 may organize storage resources into one or more storage pools 383 of storage resources for use by containerized applications 382-1 through 382-L. The container storage system may itself be implemented as a containerized service.

The container system 380 may include or be implemented by one or more container orchestration systems, including Kubernetes™, Mesos™, Docker Swarm™, among others. The container orchestration system may manage the container system 380 running on a cluster 384 through services implemented by a control node, depicted as 385, and may further manage the container storage system or the relationship between individual containers and their storage, memory and CPU limits, networking, and their access to additional resources or services.

A control plane of the container system 380 may implement services that include: deploying applications via a controller 386, monitoring applications via the controller 386, providing an interface via an API server 387, and scheduling deployments via scheduler 388. In this example, controller 386, scheduler 388, API server 387, and container storage system 381 are implemented on a single node, node 385. In other examples, for resiliency, the control plane may be implemented by multiple, redundant nodes, where if a node that is providing management services for the container system 380 fails, then another, redundant node may provide management services for the cluster 384.

A data plane of the container system 380 may include a set of nodes that provides container runtimes for executing containerized applications. An individual node within the cluster 384 may execute a container runtime, such as Docker™, and execute a container manager, or node agent, such as a kubelet in Kubernetes (not depicted) that communicates with the control plane via a local network-connected agent (sometimes called a proxy), such as an agent 389. The agent 389 may route network traffic to and from containers using, for example, Internet Protocol (IP) port numbers. For example, a containerized application may request a storage class from the control plane, where the request is handled by

the container manager, and the container manager communicates the request to the control plane using the agent 389.

Cluster 384 may include a set of nodes that run containers for managed containerized applications. A node may be a virtual or physical machine. A node may be a host system.

The container storage system 381 may orchestrate storage resources to provide storage to the container system 380. For example, the container storage system 381 may provide persistent storage to containerized applications 382-1-382-L using the storage pool 383. The container storage system 381 may itself be deployed as a containerized application by a container orchestration system.

For example, the container storage system 381 application may be deployed within cluster 384 and perform management functions for providing storage to the containerized applications 382. Management functions may include determining one or more storage pools from available storage resources, provisioning virtual volumes on one or more nodes, replicating data, responding to and recovering from host and network faults, or handling storage operations. The storage pool 383 may include storage resources from one or more local or remote sources, where the storage resources may be different types of storage, including, as examples, block storage, file storage, and object storage.

The container storage system 381 may also be deployed on a set of nodes for which persistent storage may be provided by the container orchestration system. In some examples, the container storage system 381 may be deployed on all nodes in a cluster 384 using, for example, a Kubernetes DaemonSet. In this example, nodes 390-1 through 390-N provide a container runtime where container storage system 381 executes. In other examples, some, but not all nodes in a cluster may execute the container storage system 381.

The container storage system 381 may handle storage on a node and communicate with the control plane of container system 380, to provide dynamic volumes, including persistent volumes. A persistent volume may be mounted on a node as a virtual volume, such as virtual volumes 391-1 and 391-P. After a virtual volume 391 is mounted, containerized applications may request and use, or be otherwise configured to use, storage provided by the virtual volume 391. In this example, the container storage system 381 may install a driver on a kernel of a node, where the driver handles storage operations directed to the virtual volume. In this example, the driver may receive a storage operation directed to a virtual volume, and in response, the driver may perform the storage operation on one or more storage resources within the storage pool 383, possibly under direction from or using additional logic within containers that implement the container storage system 381 as a containerized service.

The container storage system 381 may, in response to being deployed as a containerized service, determine available storage resources. For example, storage resources 392-1 through 392-M may include local storage, remote storage (storage on a separate node in a cluster), or both local and remote storage. Storage resources may also include storage from external sources such as various combinations of block storage systems, file storage systems, and object storage systems. The storage resources 392-1 through 392-M may include any type(s) and/or configuration(s) of storage resources (e.g., any of the illustrative storage resources described above), and the container storage system 381 may be configured to determine the available storage resources in any suitable way, including based on a configuration file. For example, a configuration file may specify account and authentication information for cloud-based object storage

348 or for a cloud-based storage system 318. The container storage system 381 may also determine availability of one or more storage devices 356 or one or more storage systems. An aggregate amount of storage from one or more of storage device(s) 356, storage system(s), cloud-based storage system(s) 318, edge management services 366, cloud-based object storage 348, or any other storage resources, or any combination or sub-combination of such storage resources may be used to provide the storage pool 383. The storage pool 383 is used to provision storage for the one or more virtual volumes mounted on one or more of the nodes 390 within cluster 384.

In some implementations, the container storage system 381 may create multiple storage pools. For example, the container storage system 381 may aggregate storage resources of a same type into an individual storage pool. In this example, a storage type may be one of: a storage device 356, a storage array 102, a cloud-based storage system 318, storage via an edge management service 366, or a cloud-based object storage 348. Or it could be storage configured with a certain level or type of redundancy or distribution, such as a particular combination of striping, mirroring, or erasure coding.

The container storage system 381 may execute within the cluster 384 as a containerized container storage system service, where instances of containers that implement elements of the containerized container storage system service may operate on different nodes within the cluster 384. In this example, the containerized container storage system service 30 may operate in conjunction with the container orchestration system of the container system 380 to handle storage operations, mount virtual volumes to provide storage to a node, aggregate available storage into a storage pool 383, provision storage for a virtual volume from a storage pool 383, generate backup data, replicate data between nodes, clusters, environments, among other storage system operations. In some examples, the containerized container storage system service may provide storage services across multiple clusters operating in distinct computing environments. For example, other storage system operations may include storage system operations described herein. Persistent storage provided by the containerized container storage system service may be used to implement stateful and/or resilient containerized applications.

The container storage system 381 may be configured to perform any suitable storage operations of a storage system. For example, the container storage system 381 may be configured to perform one or more of the illustrative storage management operations described herein to manage storage resources used by the container system.

In some embodiments, one or more storage operations, including one or more of the illustrative storage management operations described herein, may be containerized. For example, one or more storage operations may be implemented as one or more containerized applications configured to be executed to perform the storage operation(s). Such containerized storage operations may be executed in any suitable runtime environment to manage any storage system (s), including any of the illustrative storage systems described herein.

The storage systems described herein may support various forms of data replication. For example, two or more of the storage systems may synchronously replicate a dataset between each other. In synchronous replication, distinct copies of a particular dataset may be maintained by multiple storage systems, but all accesses (e.g., a read) of the dataset should yield consistent results regardless of which storage

system the access was directed to. For example, a read directed to any of the storage systems that are synchronously replicating the dataset should return identical results. As such, while updates to the version of the dataset need not occur at exactly the same time, precautions must be taken to ensure consistent accesses to the dataset. For example, if an update (e.g., a write) that is directed to the dataset is received by a first storage system, the update may only be acknowledged as being completed if all storage systems that are synchronously replicating the dataset have applied the update to their copies of the dataset. In such an example, synchronous replication may be carried out through the use of I/O forwarding (e.g., a write received at a first storage system is forwarded to a second storage system), communications between the storage systems (e.g., each storage system indicating that it has completed the update), or in other ways.

In other embodiments, a dataset may be replicated through the use of checkpoints. In checkpoint-based replication (also referred to as ‘nearly synchronous replication’), a set of updates to a dataset (e.g., one or more write operations directed to the dataset) may occur between different checkpoints, such that a dataset has been updated to a specific checkpoint only if all updates to the dataset prior to the specific checkpoint have been completed. Consider an example in which a first storage system stores a live copy of a dataset that is being accessed by users of the dataset. In this example, assume that the dataset is being replicated from the first storage system to a second storage system using checkpoint-based replication. For example, the first storage system may send a first checkpoint (at time $t=0$) to the second storage system, followed by a first set of updates to the dataset, followed by a second checkpoint (at time $t=1$), followed by a second set of updates to the dataset, followed by a third checkpoint (at time $t=2$). In such an example, if the second storage system has performed all updates in the first set of updates but has not yet performed all updates in the second set of updates, the copy of the dataset that is stored on the second storage system may be up-to-date until the second checkpoint. Alternatively, if the second storage system has performed all updates in both the first set of updates and the second set of updates, the copy of the dataset that is stored on the second storage system may be up-to-date until the third checkpoint. Readers will appreciate that various types of checkpoints may be used (e.g., metadata only checkpoints), checkpoints may be spread out based on a variety of factors (e.g., time, number of operations, an RPO setting), and so on.

In other embodiments, a dataset may be replicated through snapshot-based replication (also referred to as ‘asynchronous replication’). In snapshot-based replication, snapshots of a dataset may be sent from a replication source such as a first storage system to a replication target such as a second storage system. In such an embodiment, each snapshot may include the entire dataset or a subset of the dataset such as, for example, only the portions of the dataset that have changed since the last snapshot was sent from the replication source to the replication target. Readers will appreciate that snapshots may be sent on-demand, based on a policy that takes a variety of factors into consideration (e.g., time, number of operations, an RPO setting), or in some other way.

The storage systems described above may, either alone or in combination, be 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 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.

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.

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.

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.

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).

One or more embodiments may be described herein with the aid of method steps illustrating the performance of specified functions and relationships thereof. The boundaries and sequence of these functional building blocks and method steps have been arbitrarily defined herein for convenience of description. Alternate boundaries and sequences can be defined so long as the specified functions and relationships are appropriately performed. Any such alternate boundaries or sequences are thus within the scope and spirit of the claims. Further, the boundaries of these functional building blocks have been arbitrarily defined for convenience of description. Alternate boundaries could be defined as long as the certain significant functions are appropriately performed. Similarly, flow diagram blocks may also have been arbitrarily defined herein to illustrate certain significant functionality.

To the extent used, the flow diagram block boundaries and sequence could have been defined otherwise and still perform the certain significant functionality. Such alternate definitions of both functional building blocks and flow diagram blocks and sequences are thus within the scope and spirit of the claims. One of average skill in the art will also recognize that the functional building blocks, and other illustrative blocks, modules and components herein, can be implemented as illustrated or by discrete components, application specific integrated circuits, processors executing appropriate software and the like or any combination thereof.

While particular combinations of various functions and features of the one or more embodiments are expressly described herein, other combinations of these features and functions are likewise possible. The present disclosure is not limited by the particular examples disclosed herein and expressly incorporates these other combinations.

For further explanation, FIG. 4 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure. The method of FIG. 4 may be implemented by one or more storage systems such as the storage systems described above with reference to FIGS. 1A-3E. In some embodiments, the steps described herein may be performed by one or more storage controllers that are included within a particular storage system. In other embodiments, the steps may be performed (at least to some extent) by fleet management software or some other software that is responsible for the management of a fleet of storage systems (e.g., two or more storage systems).

The example method depicted in FIG. 4 includes detecting 402 that a portion of the dataset stored in a first storage system has become unavailable. The portion of the dataset stored in a first storage system may become unavailable for a variety of reasons. For example, a storage device that contains a portion of the dataset may fail, a portion (e.g., a flash die) of a storage device may fail, a malicious actor may render some portion of the dataset as being unavailable, or some other event may occur that makes the portion of the dataset unavailable. As such, detecting 402 that a portion of the dataset stored in a first storage system has become

unavailable may be carried out by detecting the occurrence of such an event, by receiving an uncorrectable read error when attempting to read the portion of the dataset, or in some other way.

Readers will appreciate that, as the phrase is used here, a portion of the dataset becoming unavailable is distinguishable from a situation in which some user (e.g., an application that is executing on a host device) is prevented from accessing the dataset. A portion of the dataset may become unavailable when a system-level entity (e.g., a storage system controller, a controller on a particular storage device) cannot access the portion of the dataset without error. A portion of the dataset is not unavailable, however, just because a user-level entity (e.g., an application executing on a host device) cannot access the portion of the dataset. A user-level entity may not be able to access some portion (or the entirety) of the dataset for a variety of reasons including, for example, the user-level entity does not have the required permissions, the portion of the dataset includes personal data that is not to be shared with the user-level entity, or for some other reason. In contrast to the aforementioned scenario, the present disclosure contemplates that a portion of a dataset becomes unavailable even to system-level entities due to, for example, hardware failure, system failure, cyberattack, data corruption, or some other event.

In some implementations, it is expected that data stored within the first storage system can survive certain data loss events, hardware failure events, or similar events. For example, the storage system may be configured to execute data recovery operations, error correction operations, RAID rebuild operations, and so on. Such operations may not be sufficient, however, to restore data in certain circumstances. For example, a system implementing a RAID-6 configuration may encounter failures with more than a certain number of devices (e.g., more than two devices), preventing data recovery operations such as RAID rebuild operations from fully recovering any missing data. Similarly, a device may experience individual block read or data corruption errors during reconstruction of a failed device as well. Additionally, some operations may require that the entirety of a dataset be retrieved from another source, such as a backup site. This may not be feasible in instances where rapid data recovery is required, and may unnecessarily waste processor, network, and storage resources to obtain the entirety of a (e.g., very large) dataset from a remote source even though it may be known that only a small portion of the dataset has been corrupted.

The portion of the dataset that may become unavailable can include any sub-part of the entire dataset. For example, the portion of the dataset that has become unavailable may be embodied as a single data block (e.g., a 1 MB block), as one or more pages within the data block, as one or more bytes in the dataset, as one or more logical portions of the dataset (e.g., a file in a file system), as a segment (or a segment and offset) in a metadata representation of the dataset, or in some other way.

In the example method depicted in FIG. 4, detecting 402 that a portion of the dataset stored in a first storage system has become unavailable may be carried out automatically and without user intervention. For example, a controller within a particular storage system may be configured to detect a device failure, to detect a failed read operation, to detect corrupted data during a scrubbing process, or in some other way. In other embodiments, the controller may receive a message from the underlying storage media (e.g., a storage device) indicating that the portion of the dataset has become

unavailable, or that some event has occurred that would make the portion of the dataset unavailable.

The example method depicted in FIG. 4 also includes obtaining 404 an identifier for the portion of the dataset that has become unavailable. The identifier for the portion of the dataset that has become unavailable may be embodied, for example, as a block identifier, as an identifier associated with a metadata representation of the dataset (e.g., a segment and offset value), as an identifier of a snapshot of the dataset along with an offset identifying a location within the snapshot that includes the portion of the dataset, or in some other way. The identifier for the portion of the dataset that has become unavailable may be obtained 404, for example, by identifying a most recent snapshot that includes the portion of the dataset and using an identifier associated with the snapshot as the identifier (optionally along with an offset or other additional information that specifies the particular portion of the snapshot that includes the portion of the dataset), by identifying a logical identifier associated with the dataset such as a file name, object name, or file name (optionally along with an offset or other additional information that specifies the location of the particular portion of the dataset), or some other identifier. Readers will appreciate that in different embodiments, different types of identifiers may be obtained 404 based on the nature of the data source that will ultimately be to find a replacement for the portion of the dataset. For example, in an embodiment where the portion of the dataset that is included in a first volume will be retrieved from a replica volume, a volume identifier may be obtained 404. In an embodiment where the portion of the dataset that is included in a first volume will be retrieved from a snapshot of the volume, however, a snapshot identifier may be obtained 404.

Readers will appreciate that in some embodiments, obtaining 404 an identifier for the portion of the dataset that has become unavailable may be carried out automatically and without user intervention. For example, applications executing on a controller (e.g., a storage controller application) may be configured with instructions that carry out the methods described herein, such that the applications automatically detect that the portion of the dataset is unavailable and obtain 404 identifiers of the unavailable portion of the dataset without user intervention. For example, such applications may be configured with instructions that include data indicating reference tables or other data stores that store identifiers for portions of the dataset that are stored in a particular storage system. As another example, identifiers for unavailable portions of the dataset may be provided to such applications during a replication process for portions of the dataset, during a process of updating/creating deduplication references, or during any other process that involves the abovementioned portions of the dataset. Accordingly, when such applications detect that the portion of the dataset is unavailable, the applications can automatically, without user intervention, initiate a process of determining, identifying, or otherwise obtaining 404 one or more identifiers for the unavailable portion of the dataset using dataset identifier resources that were previously provided or made accessible to the applications.

The example method depicted in FIG. 4 also includes locating 406, using the identifier, a replacement portion of the dataset that is stored at one or more other storage systems. In some implementations, the dataset is being (e.g., synchronously) replicated across a set of storage systems. In other words, each storage system of the set may store a replica of the dataset such that portions of the dataset can be retrieved from any of the storage systems in the set. Accord-

ingly, a portion of the dataset that is unavailable at one storage system may be available at one or more other storage systems of the abovementioned set. As described above, the unavailable data forms only a portion of the dataset, not the entire dataset. So the abovementioned identifier refers specifically to a portion of the dataset, not the dataset as a whole (although an identifier of the entire dataset may be paired with additional information to identify the portion of the dataset). For example, the identifier may be a block identifier for a dataset that comprises a large number of blocks that are distributed across a plurality of systems.

In certain cases, a replica of the dataset may be an asynchronous replica of the dataset or a snapshot of the dataset. In such cases, it may be necessary to ensure that the data contained in the asynchronous replica or the snapshot corresponds to the data that became unavailable. In other words, the portion of the dataset that is obtained from another storage system is a current version and not an older version of the data is being used to restore the portion of the dataset became unavailable.

The identifier of the portion of the dataset that is unavailable can be used to locate a replacement for the portion of the dataset that corresponds to the identifier. For example, as described above, the identifier may be a logical address. In some implementations, a data block may be referred to or stored with reference to a particular logical address on each storage system of a set of storage systems that are synchronously replicating the dataset. Accordingly, if the portion of the dataset is damaged on one storage system, its replacement may be located with reference to its corresponding logical address on one or more other storage systems. In some embodiments, if the same logical address is used on two different storage systems to store replicas of the same portion of the dataset, both storage systems may be configured to track overwrites on the portion of the dataset at the logical address. This is to ensure that the portion of the dataset stored as a replica on a storage system is the same data as that which became unavailable on another storage system.

In some implementations, locating 406 a replacement portion of the dataset can include identifying the other storage systems that may have replacement portions of the dataset. For example, the storage system may maintain replication information associated with the dataset, where such replication information identifies replication targets associated with the dataset. Such information may be used to identify replication peers (i.e., other storage systems) for the dataset.

In some implementations, locating 406 a replacement portion of the dataset can include determining whether located data is actually a replica of the portion of the dataset that became unavailable. For example, a hash value of the located data may be compared to a previously stored hash value associated with the portion of the dataset to determine whether the located data matches the portion of the dataset that has become unavailable. In other embodiments, other values that are representative of a dataset (or some portion thereof) such as a cyclic redundancy checksum ('CRC') value may be utilized. In still other embodiments, time-based information can be used to make the above-mentioned comparison. For example, if a storage system where a portion of the dataset becomes unavailable is aware that the unavailable portion was not written to or updated more recently than a replica (e.g., from another storage system), then the replica can be said to be the same data as in the portion of the dataset that became unavailable.

If data at a second storage system is determined to be a replacement for the portion of the dataset that became unavailable at a first storage system, the data located at the second storage system may be retrieved by the first storage system or otherwise be made available to the first storage system. Readers will appreciate that only the portion of the dataset that is a replacement for the unavailable portion of the dataset may be retrieved, without retrieving other portions of the dataset or the entire dataset.

Readers will appreciate that locating 406 a replacement portion of the dataset may be carried out automatically and without user intervention. For example, a storage system controller or other entity may be configured with instructions that, when executed, carry out the steps described herein, such that the applications automatically locate 406 the replacement portion of the dataset at one or more other storage systems without user intervention.

The example method depicted in FIG. 4 also includes writing 408, to the dataset that is stored in the first storage system, the replacement portion of the dataset as a replacement of the portion of the dataset that has become unavailable. Writing 408 the replacement portion of the dataset may occur automatically and without user intervention. Writing 408 the replacement portion of the dataset as a replacement of the portion of the dataset that has become unavailable may be carried out, for example, by writing the replacement data to the first storage system and updating a logical address (or even a metadata representation of the dataset) that previously referred to the unavailable portion of the dataset to now refer to location within the storage system where the replacement portion was stored. As such, the replacement portion may serve as a replacement of the portion of the dataset that has become unavailable, regardless of the fact that the replacement portion may be stored at a physical location within the first storage system that differs from the physical location of the unavailable portion of the dataset. As a result, repairing the dataset may occur in a way that is transparent to applications that are accessing the dataset.

For further explanation, FIG. 5 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure. The method of FIG. 5 is similar to the method of FIG.

4 in that the example method depicted in FIG. 5 also includes detecting 402 that a portion of the dataset stored in a first storage system has become unavailable, obtaining 404 an identifier for the portion of the dataset, locating 406, using the identifier, a replacement portion of the dataset that is stored at one or more other storage systems, and writing 408, to the dataset that is stored in the first storage system, the replacement portion of the dataset as a replacement of the portion of the dataset that has become unavailable, where the writing occurs automatically, without user intervention.

The example method of FIG. 5 differs from that of FIG. 4 in that the example method depicted in FIG. 5 also includes obtaining 505 replication metadata that was generated when replicating the dataset between multiple storage systems, wherein the identifier for the portion of the dataset is obtained based on the replication metadata. The replication metadata described herein may be any metadata that pertains to the portion of the dataset that has become unavailable and that is generated or used during a process of replicating the portion of the dataset from the storage system where the portion of the dataset was stored (e.g., a replication source) to or from one or more other storage systems (e.g., a replication target).

In some implementations, replication metadata can include metadata representing the portion of the dataset that

became unavailable. For example, where the portion of the dataset is a data block, the replication metadata can include a block identifier. The replication metadata can include a logical address of the portion of the dataset that has become unavailable. The replication metadata can also (or alternatively) include other representations of the data block, such as a hash value. The replication metadata can also (or alternatively) include sequence information representing the portion of the dataset. For example, the portion of the dataset 10 may be replicated along with many other dataset portions in a particular sequence during a replication process. Accordingly, the replication metadata may include a replication sequence or order identifier identifying the position in the replication sequence that corresponds to the portion of the dataset that has become unavailable.

In some implementations, the replication metadata can include data representing the replication target, such as one or more other storage systems. The replication metadata may include an identifier for one or more target storage systems 20 that is a replication peer of the source storage system that stored the portion of the dataset that became unavailable. The replication metadata may include a storage volume identifier. The replication metadata can also (or alternatively) include a snapshot identifier where the portion of the dataset that became unavailable was part of a snapshot. The replication metadata can also (or alternatively) include a logical address at the replication target that is to be used as a reference for the portion of the dataset that became unavailable during replication or after the replication of the 30 portion of the dataset is completed. The replication metadata can also include other metadata that is generated as a result of the replication process. For example, the replication process may be configured to generate metadata that represents replication checkpoints or milestones. More specifically, the replication process may have included the replication of a number of portions of the dataset (e.g., in response to a particular trigger such as another application confirming an update to the dataset) and generation of 35 checkpoint data indicating a percentage value or count that indicates how many of the specified portions of the dataset have been replicated, how many have not yet been replicated, and so forth.

For further explanation, FIG. 6 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure. The example method of FIG. 6 is similar to the method of FIG. 4 in that the method depicted in FIG. 6 also includes detecting 402 that a portion of the dataset stored in a first storage system has become unavailable, obtaining 404 an identifier for the portion of the dataset, locating 406, using the identifier, a replacement portion of the dataset that is stored at one or more other storage systems, and writing 408, to the dataset that is stored in the first storage system, the replacement portion of the dataset as a replacement of the 50 portion of the dataset that has become unavailable, where the writing occurs automatically, without user intervention.

In some implementations, the example method of FIG. 6 differs from that of FIG. 4 in that the example method depicted in FIG. 6 also includes obtaining 605 a deduplication hash value for the portion of the dataset. In some implementations, deduplication processes are executed to ensure that data that is being written to (or already stored by) one or more storage systems is not a duplicate of other data that currently exists on the one or more storage systems. In 60 some implementations, these deduplication processes generate representative values of the data, such as a deduplication hash value. For example, a deduplication hash value for

some block of data may be generated by using the block of data as input to a hash function. In such an example, the deduplication hash value for the block of data may be compared to hash values that were generated by using blocks of data that are already stored by the storage system as inputs to the same hash function. These deduplication hash values associated with data that is already stored by the storage system can be stored as metadata that is associated with the data, or stored in some other way.

In some implementations, the deduplication hash value may have been previously created for a portion of the dataset that later becomes unavailable. For example, a storage system may experience a failure that causes a portion of the dataset to become unavailable. However, metadata associated with that portion of the dataset, such as the deduplication hash value, may have been unaffected by the failure and still be available. Moreover, the portion of the dataset may have been replicated to or be otherwise available at one or more other storage systems. The one or more other storage systems may have an available replacement of the portion of the dataset that became unavailable, such that there is a stored deduplication hash value for the replacement portion at the one or more other storage systems (or the deduplication hash value can be generated for the replacement portion and compared to the deduplication hash value for the portion of the dataset that became unavailable). Accordingly, the deduplication hash value for the portion of the dataset that has become unavailable can be used to identify and obtain a replacement for the unavailable portion of the dataset.

For further explanation, FIG. 7 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure. The example method of FIG. 7 is similar to the method of FIG. 4 in that the example method depicted in FIG. 7 also includes detecting 402 that a portion of the dataset stored in a first storage system has become unavailable, obtaining 404 an identifier for the portion of the dataset, locating 406, using the identifier, a replacement portion of the dataset that is stored at one or more other storage systems, and writing 408, to the dataset that is stored in the first storage system, the replacement portion of the dataset as a replacement of the portion of the dataset that has become unavailable, where the writing occurs automatically, without user intervention.

In some implementations, the example method of FIG. 7 differs from that of FIG. 4 in that the example method depicted in FIG. 7 also includes obtaining 705 a file system identifier associated with the portion of the dataset that has become unavailable. The portion of the dataset that has become unavailable may be stored in a file system that is supported by an underlying storage system. Accordingly, there may be file system metadata associated with each portion of the dataset. For example, the portion of the dataset that has become unavailable may have been tagged with information such as purpose, owner, environment, and so forth. Other file system metadata can include the file header, file folder data, file path, file owner, author, date created, date modified, unique file identifiers, and so forth. The file system identifier or other file system metadata may include data structures or other resources that store associations between the abovementioned tags and portions of the dataset. Accordingly, the file system identifier for the portion of the dataset that has become unavailable can be used to identify and obtain a replacement for the unavailable portion of the dataset.

For further explanation, FIG. 8 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure.

sure. The example method of FIG. 8 is similar to the method of FIG. 4 in that the example method depicted in FIG. 8 also includes detecting 402 that a portion of the dataset stored in a first storage system has become unavailable, obtaining 404 an identifier for the portion of the dataset, locating 406, using the identifier, a replacement portion of the dataset that is stored at one or more other storage systems, and writing 408, to the dataset that is stored in the first storage system, the replacement portion of the dataset as a replacement of the portion of the dataset that has become unavailable, where the writing occurs automatically, without user intervention.

In some implementations, the method of FIG. 8 differs from that of FIG. 4 in that the example method depicted in FIG. 8 also includes obtaining 805 a cblock identifier associated with the portion of the dataset that has become unavailable. As used herein, the term “cblock” may refer to a compressed block of data that is created using a data compression algorithm. In some implementations, the dataset may be stored as one or more compressed blocks of data such that the portion of the dataset that has become unavailable may be associated with a cblock identifier. Accordingly, the cblock identifier for the portion of the dataset that has become unavailable can be used to identify and obtain a replacement for the unavailable portion of the dataset.

For further explanation, FIG. 9 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure. The method of FIG. 9 is similar to the method of FIG. 4 in that the example method depicted in FIG. 9 also includes detecting 402 that a portion of the dataset stored in a first storage system has become unavailable, obtaining 404 an identifier for the portion of the dataset, locating 406, using the identifier, a replacement portion of the dataset that is stored at one or more other storage systems, and writing 408, to the dataset that is stored in the first storage system, the replacement portion of the dataset as a replacement of the portion of the dataset that has become unavailable, where the writing occurs automatically, without user intervention.

In some implementations, the method of FIG. 9 differs from that of FIG. 4 in that the method also includes obtaining 905 a private identifier that is not visible to applications that utilize the dataset for the portion of the dataset that has become unavailable. The dataset may be stored in different formats and with reference to different logical organizations, some of which may not be visible to a user of the dataset. For example, users of the dataset may see a volume-based interface for accessing the dataset, where blocks of data for the dataset are logically addressed in terms of a volume identifier and a corresponding offset identifier. However, the dataset may be maintained using a different logical organization internally that is not visible to outside users. For example, the dataset may be logically organized using an internal metadata representation of the dataset, such that each data block is accessed using internal metadata constructs. Accordingly, the private identifier for the portion of the dataset that has become unavailable can be used to identify and obtain a replacement for the unavailable portion of the dataset.

For further explanation, FIG. 10 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure. The example method of FIG. 10 is similar to the example method of FIG. 4 in that the method depicted in FIG. 10 also includes detecting 402 that a portion of the dataset stored in a first storage system has become unavailable, obtaining 404 an identifier for the portion of the dataset, locating 406, using the identifier, a replacement portion of the dataset that is stored at one or more other storage systems, and writing 408, to the dataset that is stored in the first storage system, the replacement portion of the dataset as a replacement of the portion of the dataset that has become unavailable, where the writing occurs automatically, without user intervention.

portion of the dataset that is stored at one or more other storage systems, and writing 408, to the dataset that is stored in the first storage system, the replacement portion of the dataset as a replacement of the portion of the dataset that has become unavailable, where the writing occurs automatically, without user intervention.

The example method of FIG. 10 differs from that of FIG. 4 in that the example method depicted in FIG. 10 also includes obtaining 1005 the identifier from an application that was utilizing the dataset prior to the portion of the dataset becoming unavailable. An application may be processing portions of the dataset when a failure occurs, causing a particular portion of the dataset to become unavailable. Readers will appreciate that the application may generate metadata, logs, or other application data that can be used to identify and locate the portion of the dataset that has become unavailable. For example, the application may generate application execution logs that indicate the portions of the dataset that the application was processing. An execution log may indicate, for example, sequence numbers for blocks of data that were being processed. The sequence numbers may indicate that, for example, block 30 of 50 blocks of the dataset had completed processing and block 31 was being processed before the failure occurred. Accordingly, the identifier for the portion of the dataset that became unavailable (e.g., ‘block 31’) may be obtained based on an application’s processing of the dataset. As another example, metadata or logs generated by the application that was utilizing the dataset may indicate that the portion of the dataset that has become unavailable was replicated to another storage system within a threshold period of time prior to the failure. Such information can be used to identify the portion of the dataset that has become unavailable and locate the replica based on replication metadata provided by the application.

For further explanation, FIG. 11 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure. The example method of FIG. 11 is similar to the method of FIG. 4 in that the example method depicted in FIG. 11 also includes detecting 402 that a portion of the dataset stored in a first storage system has become unavailable, obtaining 404 an identifier for the portion of the dataset, locating 406, using the identifier, a replacement portion of the dataset that is stored at one or more other storage systems, and writing 408, to the dataset that is stored in the first storage system, the replacement portion of the dataset as a replacement of the portion of the dataset that has become unavailable, where the writing occurs automatically, without user intervention.

In some implementations, the method of FIG. 11 differs from that of FIG. 4 in that the example method depicted in FIG. 11 also includes querying 1102, using the location, the one or more other storage systems for portions of the dataset that are located at a corresponding location within the one or more other storage systems, wherein the location includes one or more of a segment identifier and a segment offset value. In addition to the methods described above, the location of a portion of the dataset that has become unavailable may be expressed as a segment identifier and offset, which can be used to query other storage systems for replacement portions of the dataset. In some embodiments, a storage device may be logically divided into multiple segments, each identified by a segment identifier. In other embodiments, the logical location of a portion of data may be referred to using a segment identifier and an offset value that is with reference to another value, such as the segment

identifier. In certain cases, each of a number of storage systems may have replica portions of the dataset being stored at the storage system at the same segment identifier and segment offset value at approximately the same time during a replication process. In such cases, a combination of segment identifier and segment offset value plus a time value can represent a globally unique identifier for a portion of a dataset. For example, in cases where the dataset is being replicated (e.g., synchronously or asynchronously) to different storage systems, portions of the dataset may be referenced using the same logical address identifiers across each of the different storage systems. In other words, a portion of the dataset that is stored with reference to a particular segment ID and offset on one storage system may be referenced using that same segment ID and offset on one or more other storage systems. Given this, the segment ID and offset can be used to query other storage systems for a replacement of the portion of the dataset that has become unavailable.

In some implementations, the method also includes determining 1104 that the replacement portion of the dataset is available at the corresponding location at the one or more other storage systems. For example, the abovementioned querying process may return a number of results. For example, some storage systems may return a result of the query indicating that nothing is currently stored with reference to the logical segment ID and offset. Other storage systems may return a number of dataset portions. In some implementations, any of the returned dataset portions may be a replacement for the portion of the dataset that has become unavailable just because these returned dataset portions were referenced by the segment ID and offset that was used for the portion of the dataset that became unavailable. In other implementations, the returned dataset portions may be validated using methods such as comparison of hash values and so on.

For further explanation, FIG. 12 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure. The example method of FIG. 12 is similar to the method of FIG. 4 in that the example method depicted in FIG. 12 also includes detecting 402 that a portion of the dataset stored in a first storage system has become unavailable, obtaining 404 an identifier for the portion of the dataset, locating 406, using the identifier, a replacement portion of the dataset that is stored at one or more other storage systems, and writing 408, to the dataset that is stored in the first storage system, the replacement portion of the dataset as a replacement of the portion of the dataset that has become unavailable, where the writing occurs automatically, without user intervention.

In some implementations, the method of FIG. 12 differs from that of FIG. 4 in that the example method depicted in FIG. 12 also includes identifying 1202 data whose similarity to the portion of the dataset (that has become unavailable) satisfies a threshold similarity. In certain circumstances, a dataset portion that is similar but not identical to the portion of the dataset that has become unavailable may be acceptable. For example, metadata for the portion of the dataset that has become unavailable may define rules or conditions for acceptable replacements of the portion of the dataset. As a more specific example, the metadata for the portion of the dataset that has become unavailable may indicate that, if the portion of the dataset is corrupted or lost, a replacement data portion having a 95% or 99% similarity to the portion of the dataset that has become unavailable may be an acceptable replacement.

In other implementations, a replacement data portion may be convertible into the portion of the dataset that has become unavailable by execution of one or more operations on the replacement data portion. For example, an application may be processing the dataset such that each sequential portion of a dataset is greater than the previous portion of the dataset in the sequence by some fixed value. Accordingly, a replacement portion may be located that is, for example, the immediately preceding portion of the dataset in the particular sequence relative to the portion of the dataset that became unavailable. Such a near-replacement data portion may be incrementable by the fixed value described above, in order to obtain a replacement of the portion of the dataset that became unavailable.

For further explanation, FIG. 13 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure. The example method of FIG. 13 is similar to the method of FIG. 4 in that the example method depicted in FIG. 13 also includes detecting 402 that a portion of the dataset stored in a first storage system has become unavailable, obtaining 404 an identifier for the portion of the dataset, locating 406, using the identifier, a replacement portion of the dataset that is stored at one or more other storage systems, and writing 408, to the dataset that is stored in the first storage system, the replacement portion of the dataset as a replacement of the portion of the dataset that has become unavailable, where the writing occurs automatically, without user intervention.

In some implementations, the method of FIG. 13 differs from that of FIG. 4 in that the example method depicted in FIG. 13 also includes determining 1302 that a first timestamp associated with storage of the replacement portion of the dataset is not earlier than a second timestamp associated with storage of the portion of the dataset that became unavailable. In some implementations, if the replacement data portion has a storage timestamp that is within a threshold period of time after the storage timestamp of the portion of the dataset that has become unavailable, the replacement data portion may be acceptable as a replacement for the portion of the dataset that has become unavailable.

For further explanation, FIG. 14 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure. The example method of FIG. 14 is similar to the method of FIG. 4 in that the example method depicted in FIG. 14 also includes detecting 402 that a portion of the dataset stored in a first storage system has become unavailable, obtaining 404 an identifier for the portion of the dataset, locating 406, using the identifier, a replacement portion of the dataset that is stored at one or more other storage systems, and writing 408, to the dataset that is stored in the first storage system, the replacement portion of the dataset as a replacement of the portion of the dataset that has become unavailable, where the writing occurs automatically, without user intervention.

In some implementations, the method of FIG. 14 differs from that of FIG. 4 in that the example method depicted in FIG. 14 also includes determining 1402 that a first checksum value associated with the replacement portion of the dataset equals a second checksum value associated with the portion of the dataset. For example, a checksum (e.g., a cyclic redundancy checksum (CRC)) value for the replacement data portion may be compared to a stored checksum value for the portion of the dataset that has become unavailable. If the values match, the replacement data portion may be stored as a replacement for the portion of the dataset that has

become unavailable. In some embodiments, the CRC value may be a cryptographic checksum value.

For further explanation, FIG. 15 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure. The method of FIG. 15 is similar to the method of FIG. 4 in that the method depicted in FIG. 15 also includes detecting 402 that a portion of the dataset stored in a first storage system has become unavailable, obtaining 404 an identifier for the portion of the dataset, locating 406, using the identifier, a replacement portion of the dataset that is stored at one or more other storage systems, and writing 408, to the dataset that is stored in the first storage system, the replacement portion of the dataset as a replacement of the portion of the dataset that has become unavailable, where the writing occurs automatically, without user intervention.

In some implementations, the method of FIG. 15 differs from that of FIG. 4 in that the example method depicted in FIG. 15 also includes determining 1508 that a first error correction code value associated with the replacement portion of the dataset equals a second error correction code value associated with the portion of the dataset. Similar to the method described above with respect to FIG. 14, for example, an error correction code for the replacement data portion may be compared to a stored error correction code for the portion of the dataset that has become unavailable. If the values match, the replacement data portion may be stored as a replacement for the portion of the dataset that has become unavailable.

For further explanation, FIG. 16 sets forth a flow chart illustrating an example method of restoring lost data in accordance with some embodiments of the present disclosure. The example method of FIG. 16 is similar to the method of FIG. 4 in that the example method depicted in FIG. 16 also includes detecting 402 that a portion of the dataset stored in a first storage system has become unavailable, obtaining 404 an identifier for the portion of the dataset, locating 406, using the identifier, a replacement portion of the dataset that is stored at one or more other storage systems, and writing 408, to the dataset that is stored in the first storage system, the replacement portion of the dataset as a replacement of the portion of the dataset that has become unavailable, where the writing occurs automatically, without user intervention.

In some implementations, the example method of FIG. 16 differs from that of FIG. 4 in that the example method depicted in FIG. 16 also includes determining 1610 a location at the first storage system that was associated with the portion of the dataset. As described above, the portion of the dataset that has become unavailable may be referred to using a logical address (e.g., <volume, offset> or <segment, offset> and so forth) or a physical address (physical memory address, sector location, and so forth). In some implementations, the logical address may be used to identify the portion of the dataset that has become unavailable and to locate its replacement at another storage system. Such information may be used to identify the portion of the dataset that has become unavailable and locate its replacement at another storage system.

In some implementations, the method also includes writing 1612 the replacement dataset portion to the determined location, such as by updating the logical address referencing the portion of the dataset that became unavailable to now reference the replacement dataset portion. In other words, the portion of the dataset that became unavailable loses the logical address reference that could previously be used to access the portion of the dataset that has become unavail-

able. After the reference is updated, the logical address can be used to refer to the replacement portion of the dataset. In other implementations, the replacement dataset portion may be written to a different logical address rather than the old logical address for the portion of the dataset that became unavailable. In such a case, additional updates may be made to ensure consistency, such as updating a mapping of logical addresses to physical addresses by mapping the different logical address to a same physical address as the old logical address.

For further explanation, FIG. 17 sets forth a flow chart illustrating an example method of recovering corrupted data through speculative bitflip and cross-validation. The example method of FIG. 17 may be implemented by one or more storage systems such as the storage systems described above with reference to FIGS. 1A-3E. In some embodiments, the steps described herein may be performed by one or more storage controllers (e.g., storage controllers 110A-D, shown in FIG. 1A, or storage controller 101, shown in FIG. 1B) that are included within a particular storage system. In other embodiments, the steps may be performed (at least to some extent) by fleet management software or some other software that is responsible for the management of a fleet of storage systems (e.g., two or more storage systems).

As discussed above, data stored in storage devices (e.g., storage drives 171A-F, shown in FIG. 1A) can be stored in one or more portions. As used herein, the term ‘data portion’ may refer to a data block, object, file, page, shard, stripe, or the like. In an example embodiment, data may be stored as Flash pages, which may be a few kilobytes in size. A Flash block may include a large number of Flash pages. A Flash page may be further subdivided into a header portion and content portion. There may be more than one header for a Flash page. In addition, there may be another header within a Flash block that stores information associated with each Flash page.

In some implementations, a storage device may implement error detection and/or error correction methods to detect or correct errors in stored data. These methods may run automatically or on request by an administrator. For example, a storage device (such as a NAND flash chip) may include redundancy schemes such as error correction codes that may tolerate corrupted bits within a storage device, or erasure codes that tolerate the failure of multiple storage nodes, and so on. Additionally, a low density parity check (‘LDPC’) code may be used, or a checkbit, parity bit, checksum, and/or other error detection or error correction methods.

However, readers will appreciate that the error detection or correction methods may fail to identify all errors that may exist in data. For example, a data loss event (e.g., a cyber-attack or other event requiring disaster recovery) may occur that results in some data blocks being irretrievably lost. In other circumstances, certain data blocks can be retrieved but their contents cannot be verified as being accurate to a time prior to the data loss event. Independent of data loss events or large-scale data corruption events, small errors may be introduced into the data over time. A storage device may provide these data blocks and even perform error detection and correction but be unable to verify whether all errors that were introduced have been identified and corrected.

Moreover, existing error detection methods may be able to identify the presence of an error, but not its precise location. Similarly, a data loss event may result in the loss of blocks of data whose retrieval relies on the accuracy of other data blocks. For example, in certain RAID configura-

tions, lost data blocks can be retrieved by performing parity calculations that use other data blocks, which must themselves include accurate data if the lost data blocks are to be correctly reconstructed. If the other data blocks include errors, they may not be usable to correctly reconstruct the lost data blocks. In other words, even where the data is retrievable from a storage device, error correction methods may indicate corruption in enough data blocks such that an entire section of data (such as a RAID stripe) is unusable. In other circumstances, a storage system administrator or customer may wish to detect the presence of errors in data before any errors affect system performance. For example, during a storage system scrubbing process, an administrator may wish to check for the presence of errors in the data.

One example of an error may be a bit flip error, where one or more bits in a data block change in value from 0 to 1 or from 1 to 0. Bit flip errors can occur unexpectedly and the location of the bit flip within a data block, such as in a Flash page, may not be known. Bit flip errors may occur during a data loss event, or even due to factors such as repeated use of the storage medium (e.g., a large number of P/E cycles occurring in Flash memory), environmental factors such as heat or radiation, or inaccuracies introduced during data storage or copying, among other reasons. Multiple bit flip errors may occur within a single data block, and can occur in a header portion or content portion. Bit flip errors can even occur within an additional portion that is appended to or otherwise added to the data block for purposes of error detection or error correction, such as within a parity bit portion or cyclic redundancy check (CRC) block that is appended to a data portion. Since the bit position of a bit flip error may not be obviously identifiable, the corruption in the data may become evident only when a consumer’s (e.g., an application’s) use of the data produces erroneous results.

The disclosed systems and methods are directed to correcting such errors by altering the data portion in a speculative manner in order to reverse the effects of the data corruption. Accordingly, the example method depicted in FIG. 17 includes altering 1702 a first data portion stored in a storage device, wherein the first data portion contains an error. Altering 1702 the first data portion stored in the storage device, wherein the first data portion contains an error can be performed in a variety of ways. As described above, the storage device (e.g., storage drives 171A-F, shown in FIG. 1A) may store data in data portions. Each data portion may be of a fixed size, such as 512 bytes or 4 kilobytes. As described above, errors may be introduced into the data portion over time such that an error is detected but the nature of the error or the precise location of the error inside the data portion is unknown. Accordingly, the method includes altering the first data portion in an attempt to undo the error, given that, for example, a data loss event can cause corruptions within data. Since the location of the error within the data portion is initially not known, altering the first data portion to undo the error can include, for example, changing a sub-portion (e.g., a first sub-portion) of the data portion in a speculative manner. For example, where the data portion comprises a set of bytes, altering the first data portion can include altering the value of one or more bits of the first data portion. For example, the storage controller’s described herein may copy the data portion into a register or to another location in memory and then attempt to modify a sub-portion of the data portion. Such a modification may be part of a scheme that speculates that the error is within that sub-portion and that modifying that sub-portion will resolve the error.

For any data portion, such as a Flash page of size 4 KB, a hash function can be used to generate a hash value that represents the data contained in the 4 KB page. Readers of skill in the art will understand that any suitable hash function can be used to generate the hash value, where a hash function is designed to generate a fixed-size output value from an input while minimizing collisions, or the probability that the same output value is generated from two different inputs. Hash values for a data portion from different points in time can then be compared to each other to determine whether the data portion's contents have changed based on whether the hash values from different points in time match each other or not.

An example of a hash function may be the use of a cyclic redundancy check (or checksum) (CRC) method. Readers will appreciate that the first data portion, when initially stored, may have been stored with a CRC value that is generated from the first data portion and then appended to the first data portion. CRC values may be generated, for example, by performing a mathematical calculation using the data of the first data portion. As a more particular example, the mathematical calculations can include polynomial long division, in which a chosen generator polynomial becomes the divisor, the data of the first data portion becomes the dividend, and the remainder is the result, or the obtained CRC value. The obtained CRC value may itself be of a certain number of bits, or n bits. As an even more specific example, a 1-bit CRC, also referred to as a parity bit, may be a single CRC bit added to the first data portion, where the single bit indicates a count of 1-bits (or 0-bits) within the first data portion. In other words, the CRC value can be said to be representative of the data of the first data portion.

Readers will appreciate that corruption within the first data portion (e.g., a bitflip error) can cause the appended CRC value to no longer be representative of the first data portion. For example, prior to any data corruption, the first data portion may be 7 bits of data that are all zeroes, i.e., '0000000'. For this data portion, an appended CRC value, such as a 1-bit CRC or parity bit, represents a count of 1-bits in the first data portion. There are no 1-bits in this data portion so, in an even parity bit scheme, the parity bit may also be set to 0. This is because the parity bit is set to 0 in an even parity bit scheme if the count of 1-bits is an even number (in this case, also zero) or 1 if the count is an odd number. More generally, an 'even parity' scheme is a special case of a CRC, where the 1-bit CRC is generated by the polynomial $x+1$. The parity bit is also counted toward the count of 1-bits in the parity bit scheme. The final result with the CRC value being appended to the first data portion stored will be '00000000'. As another example, the CRC value itself may be 4 bits (e.g., 1010) or a byte (e.g., 10101101) that is generated using a different method (e.g., using a specific generator polynomial) that is different from the parity bit example outlined above.

Subsequently, data corruption in the first data portion may result in changed values for one or more bits of the first data portion. As a result, the CRC value may no longer be representative of the data of the first data portion. As an example, the CRC value of 1010 may have been initially calculated using the data of the first data portion and then appended to the first data portion. Subsequent to data corruption occurring within the first data portion, the data in the first data portion will have changed and the CRC value of 1010 may or may not be representative of the data of the first data portion. As a result, if a CRC value of the first data portion is calculated after data corruption is known to have

occurred within the first data portion, the newly calculated CRC value for the corrupted data portion may be different (i.e., not 1010 but 1110 or some other value). If the CRC value appended to the first data portion is different from the CRC value calculated for the first data portion, it may be concluded that data corruption exists within the first data portion.

The systems and methods described herein involve a speculative data alteration scheme where it is speculated that if a certain sub-portion of the data is altered, the errors introduced due to data corruption may be resolved and the corruption may be reversed. Accordingly, the first data portion is altered, as described above in the step of altering 1702, in an attempt to resolve the error. More specifically, it is speculated that if a specific alteration is made, then a CRC value that is calculated using the altered first data portion should match the stored CRC value that was initially appended to the first data portion. Accordingly, the example method depicted in FIG. 17 also includes calculating 1704 a hash value 1705 using the altered first data portion. Calculating 1704 the hash value 1705 using the altered first data portion can be carried out in a variety of ways. As discussed above, a CRC value may be calculated using data of the altered first data portion.

The example method depicted in FIG. 17 also includes comparing 1706 the hash value 1705 calculated using the altered first data portion to a stored hash value for the first data portion. As discussed above, the CRC value may be said to represent the data of the first data portion such that the CRC value (or any hash value in general) may be used to detect the presence of errors within the first data portion. Supposed that the stored CRC value for the first data portion is termed CRC-A, the value being 1010. Then a portion is altered (e.g., a bit is flipped) within the first data portion and a CRC value, termed CRC-B, is calculated using data of the altered first data portion.

The example method depicted in FIG. 17 also includes determining 1708, based on the comparing, whether the altering corrects the error in the first data portion. Continuing with the example above, if CRC-B=CRC-A (i.e., CRC-B is also calculated to be 1010), then it may be determined that the specific alteration made to the first data portion (i.e., the specific bit that was flipped) resolved the error. The altered first data portion can be considered to be a restored version of the first data portion, prior to data corruption having been introduced into the first data portion.

Conversely, if CRC-B≠CRC-A (i.e., CRC-B is calculated to be not 1010 but some other value like 1000), then it may be determined that the specific alteration made to the first data portion (i.e., the specific bit that was flipped) did not resolve the error. If so, the method may return to the step of altering 1702 and alter a different sub-portion (e.g., a different bit) of the first data portion, then re-calculate the CRC (or other hash value) for the re-altered first data portion, and determine whether the re-calculated CRC value matches the stored CRC value in the first data portion.

For further explanation, FIG. 18 sets forth a flow chart illustrating an example method of recovering corrupted data through speculative bitflip and cross-validation. The example method of FIG. 18 is similar to the example method of FIG. 17 in that the example method of FIG. 18 also includes altering 1702 a first data portion stored in a storage device, wherein the first data portion contains an error, calculating 1704 a hash value 1705 using the altered first data portion, comparing 1706 the hash value 1705 calculated using the altered first data portion to a stored hash value for

the first data portion, and based on the comparing, determining 1708 whether the altering corrects the error in the first data portion.

The example method of FIG. 18 differs from the example method of FIG. 17 in that in the example method of FIG. 18, altering the first data portion also includes flipping 1802 at least one bit in the first data portion. As described above, altering the first data portion can include making changes to a sub-portion of the first data portion, such as a single bit. Altering the single bit can include changing its value from 0 to 1 or from 1 to 0. For example, a storage controller can copy out data from a storage device that corresponds to the first data portion into another storage area or memory area and alter the data for the first data portion.

As discussed above, this process involves speculation as to the bit that initially got flipped due to corruption such that if the bit were ‘flipped back’, the corruption would be resolved. Since the precise bit position may not be known, the speculative bit flipping may proceed by flipping, for example, the very first bit in the first data portion. For example, for the bits in an 8-bit byte having index values 0-7, flipping the bit can include flipping the bit at position 0, or the leftmost bit in the byte, or flipping the bit can include flipping the bit at position 7, or the rightmost bit in the byte, and so on.

Bit flipping as described above can be performed in a variety of ways. Various bitwise manipulation functions or operators can be used. As an example, the first data portion can be processed with an XOR function that includes a mask. As a more specific example, suppose that the binary representation of the first data portion (or the first four bits of the first data portion) is 1101. Then flipping the first bit of this first data portion can include XOR-ing 1101 with a mask of 1000 whereby 1101 XOR 1000=0101. In other words, the first bit of this first data portion has been deliberately altered, or flipped from 1 to 0. So 1101 is now 0101. As mentioned above, this is done as part of a speculative bit flipping that speculates that earlier in time (e.g., due to data corruption), that first bit was erroneously flipped from 0 to 1.

Altering the first data portion can also include flipping more than one bit, such as flipping two bits or more bits at a time in the first data portion. Similar to the example described above, flipping two or more bits can also involve various bitwise manipulation functions such as XOR or other logic functions and additional calculation components such as masks. Readers will appreciate that bit flipping may be performed in parallel such that multiple bits are flipped during a speculative process of identifying the bit that was corrupted (i.e., whose value was initially flipped due to data corruption, leading to an error).

The example method of FIG. 18 further differs from the example method of FIG. 17 in that in the example method of FIG. 18, flipping 1802 the at least one bit also includes flipping 1804 a bit within a header portion of the first data portion.

Readers will further appreciate that where the first data portion is, for example, a Flash memory page, the Flash page may include a header portion and a content portion. The header portion may include details regarding the Flash page, such as whether the Flash page is a data page or a map page. The header portion may include the CRC value that corresponds to that Flash page. Accordingly, flipping 1802 the at least one bit can include flipping a bit in the header portion of the Flash page. Flipping 1802 the at least one bit can also include flipping a bit in the CRC value stored in the header portion as well. Moreover, there may be multiple CRC

values stored in the header or elsewhere in the first data portion. Flipping 1802 the at least one bit can include flipping bits at any location within the first data portion. As described above, once any bit or set of bits is flipped, the CRC value for the altered first data portion is calculated and compared to the CRC value stored within the first data portion.

Readers will appreciate that the speculative bit flipping process can be optimized or improved over time based on prior experience. For example, it may be identified that, for a particular storage device, a bit in a specific position is always or nearly always the source of a bit flip error. If so, that same bit may be flipped first during a subsequent data loss or data corruption event, based on prior knowledge that this bit is commonly the source of bit flip errors. In other words, the bit flipping process may not commence at the first bit or some preset position each time but may be updated to initiate at the specific bit position that is observed to be the source of most or all errors in such a storage device. Moreover, a bit or set of bits at the same position(s) may be targeted in other storage devices that are of the same type, model, or configuration, or even storage devices that are of different types.

For further explanation, FIG. 19 sets forth a flow chart illustrating an example method of recovering corrupted data through speculative bitflip and cross-validation. The example method of FIG. 19 is similar to the example method of FIG. 17 in that the example method of FIG. 19 also includes altering 1702 a first data portion stored in a storage device, wherein the first data portion contains an error, calculating 1704 a hash value 1705 using the altered first data portion, comparing 1706 the hash value 1705 calculated using the altered first data portion to a stored hash value for the first data portion, and based on the comparing, determining 1708 whether the altering corrects the error in the first data portion.

The example method of FIG. 19 differs from the example method of FIG. 17 in that in the example method of FIG. 19, calculating the hash value using the altered first data portion includes calculating 1902 a cyclic redundancy check (CRC) value using the altered first data portion. As described above, CRC may be a method for detecting errors in stored data, whereby a CRC value is generated from data, such as a set of bytes, and stored with the data at a first point in time. The CRC value can be recalculated at a second point in time with the same set of bytes and the recalculated CRC value is compared to the stored CRC value. If there is a match, it provides a strong indication that no corruption was introduced into the data between the first point in time and the second point in time. If the values differ, it may be determined that the data has been corrupted (e.g., a bit flip error has occurred in the data).

Accordingly, the method of FIG. 19 includes calculating a CRC value for the altered first data portion. More specifically, the first data portion is altered, such as by flipping one of the bits of the first data portion. After the bit flip or data alteration, a new CRC value is calculated for the altered first data portion. If this new CRC value for the altered first data portion matches the stored CRC value (e.g., both CRC values are 1110) that is obtained from the stored first data portion, it may provide a strong indication (though not necessarily a guarantee, as discussed later) that there is no data corruption in the first data portion. Otherwise, if the new CRC value for the altered first data portion does not match the stored CRC value that is obtained from the stored first data portion (e.g., the stored CRC is 1110 but the

recalculated CRC value is 1001), it may indicate the presence of data corruption in the first data portion, such as one or more bit flip errors.

For further explanation, FIG. 20 sets forth a flow chart illustrating an example method of recovering corrupted data through speculative bitflip and cross-validation. The example method of FIG. 20 is similar to the example method of FIG. 17 in that the example method of FIG. 20 also includes altering 1702 a first data portion stored in a storage device, wherein the first data portion contains an error, calculating 1704 a hash value 1705 using the altered first data portion, comparing 1706 the hash value 1705 calculated using the altered first data portion to a stored hash value for the first data portion, and based on the comparing, determining 1708 whether the altering corrects the error in the first data portion.

Readers of skill in the art will understand that the use of a hash function, such as a CRC, may not always provide detection of data corruption such as bit flip errors with a high degree of probability. As with other hash functions, CRCs may suffer from aliasing or collision problems. In other words, two different inputs may yield the same CRC value. A comparison between CRC values of a stored first data portion and an altered first data portion can certainly provide a strong indication of whether corruption—such as bit flip errors—exists in the first data portion. But CRC comparison may not necessarily guarantee that a data portion is free of bit flip errors. This is because the speculative bit flip may, coincidentally, result in an altered first data portion whose CRC value is the same as the stored CRC for the data portion. But the actual bit flip error may be in a different position. So the speculative bit flip at one position may coincidentally create an altered first data portion whose CRC value matches the stored CRC value, but the bit flip error may still exist at another position within the first data portion.

To cross-validate the CRC value generated using the altered first data portion, further processing using RAID parity calculations may be used. Readers of skill in the art will understand that a RAID configuration may be used to protect data in a storage device (e.g., storage drives 171A-F). The RAID configuration may be applicable to the abovementioned first data portion, such that the first data portion may be considered a RAID shard that is a member of a collection of data portions or shards that is collectively termed a RAID stripe. In RAID configurations, data that is stored in a storage device is protected through the calculation and storage of parity data that is generated using the data. Where the first data portion is a shard of a RAID stripe, the first data portion can be used as part of RAID parity calculations to generate the parity data, stored as parity data shards. Taken together, data shards and parity shards can be used to reconstruct data portions that are lost, corrupted, or otherwise unavailable. RAID parity calculations can thus be used to cross-validate the altered first data portion. More specifically, the altered first data portion can be used in RAID parity calculations to reconstruct another data portion (e.g., another data portion that is part of the same RAID stripe as the first data portion). The contents of the reconstructed data portion can be further validated through CRC comparison, memory comparison, or other methods. If such validation succeeds for the reconstructed data portion, this may provide a stronger indication that the alteration in the first data portion served to resolve the bit flip error. This is because RAID parity calculations using the altered first data portion would not result in a validated reconstructed data

portion if the alteration in the first data portion did not resolve errors within the first data portion.

The example method of FIG. 20 differs from the example method of FIG. 17 in that in the example method of FIG. 20 5 also includes reconstructing 2002 a second data portion of a RAID stripe with parity calculations that use the altered first data portion. Reconstructing 2002 a second data portion of a RAID stripe with parity calculations that use the altered first data portion may be performed in a variety of ways. For 10 example, FIG. 20 shows a RAID stripe 2050 that includes shard A1 2010, shard A2 2012, shard A3 2014, shard A_P 2016, and shard A_Q 2018. As shown in FIG. 20, shard A1 2010, shard A2 2012, shard A3 2014, shard A_P 2016, and shard A_Q 2018 may be stored in storage devices 2001, 2003, 15 2005, 2007, and 2009, respectively. In the example shown in FIG. 20, shard A1 2010, shard A2 2012, shard A3 2014 may include data (such as user data) that is stored in the storage devices by another device (e.g., an application host), whereas shard A_P 2016 and shard A_Q 2018 may include 20 parity data that is generated with RAID parity calculations.

In this example, shard A1 2010 may represent the first data portion. An altered version of the shard A1 2010 may be generated at step 1702, creating an altered shard A1' 2011. The altered shard A1' 2011 is then used in RAID parity 25 calculations to reconstruct another shard in the RAID stripe 2050. For example, the altered shard A1' 2011, along with shard A3 2014, shard A_P 2016, and shard A_Q 2018, may be used to reconstruct shard A2 2012, resulting in a reconstructed shard A2' 2013.

The example method of FIG. 20 also includes calculating 30 2004 a second hash value using the reconstructed second data portion. Calculating 2004 a second hash value using the reconstructed second data portion can include, for example, calculating a hash value using shard A2' 2013. For example, 35 a CRC value can be generated using the reconstructed shard A2' 2013.

The example method of FIG. 20 also includes determining 40 2006 that the calculated second hash value matches a stored hash value within the second data portion. Determining 2006 that the calculated second hash value matches a stored hash value within the second data portion can include determining that the hash value generated using shard A2' 2013 matches a stored hash value stored within shard A2 2012. More specifically, determining 2006 that the calculated 45 second hash value matches a stored hash value within the second data portion can include determining that the CRC value generated using shard A2' 2013 matches a stored CRC value stored within shard A2 2012.

The example method of FIG. 20 also includes determining 50 2008 that the altering corrects the error in the first data portion, based on the determination 2006 that the calculated second hash value matches a stored hash value within the second data portion. Particularly, if there is a match between the CRC value generated using shard A2' 2013 and the stored CRC value stored within shard A2 2012, this may 55 indicate that the reconstructed shard A2' 2013 and the original shard A2 2012 include the same data. As a result, because reconstructed shard A2' 2013 matches original shard A2 2012, then altered shard A1' 2011 is also likely to include the same data that was originally stored in storage device 2010 prior to a data loss or data corruption event. In other words, altered shard A1' 2011 may represent an uncorrupted shard A1 2010. More specifically, the RAID 60 parity calculations that use altered shard A1' 2011 to reconstruct shard A2' 2013 resulted in a shard A2' that was the same as the original shard A2. Therefore, it may be concluded that the shard used to reconstruct shard A2 (i.e., shard 65

A1') also includes the data that was originally stored within it prior to data corruption. Accordingly, in some implementations, it may be further concluded that the alteration of shard A1' 2010 in step 1702 to generate altered shard A1' 2011 served to resolve the bit flip error that originally may have occurred in shard A1' 2010.

For further explanation, FIG. 21 sets forth a flow chart illustrating an example method of recovering corrupted data through speculative bitflip and cross-validation. The example method of FIG. 21 is similar to the example method of FIG. 20 in that the example method of FIG. 21 also includes altering 1702 a first data portion stored in a storage device, wherein the first data portion contains an error, calculating 1704 a hash value 1705 using the altered first data portion, comparing 1706 the hash value 1705 calculated using the altered first data portion to a stored hash value for the first data portion, based on the comparing, determining 1708 whether the altering corrects the error in the first data portion, reconstructing 2002 a second data portion of the RAID stripe with parity calculations that use the altered first data portion; calculating 2004 a second hash value using the reconstructed second data portion; determining 2006 that the calculated second hash value matches a stored hash value within the second data portion; and based on the determination, determining 2008 that the altering corrects the error in the first data portion.

The example method of FIG. 21 differs from the example method of FIG. 20 in that in the example method of FIG. 21 also includes, determining 2102 that the second data portion is unavailable. Continuing with the example above in FIG. 20, RAID stripe 2050 should include shard A1' 2010, shard A2' 2012, shard A3' 2014, shard A_P' 2016, and shard A_Q' 2018. But shard A2' 2012 may be lost during a data loss event. For example, some circumstance such as a severe power surge or natural disaster may cause a partial or total loss of data stored in storage device 2003. As a result, the data of shard A2' 2012 may be partially or wholly unretrievable. As one example, the entirety of the shard A2' 2012 may be lost. As another example, the header portion of shard A2' 2012 may be available but the remaining contents may be unavailable. In a related example, while the some or all of the contents of shard A2' 2012 may be unavailable, the storage device 2003 may still be able to provide metadata or other additional data associated with shard A2' 2012. This metadata or other additional data may be accessible from a primary header, or primary page header, or another header, or another storage location in storage device 2003 or another storage device. Readers will appreciate that, as an example of this metadata, the CRC values or other hash values that were generated and stored for the shard A2' 2012 may still be accessible.

The example method of FIG. 21 also includes, obtaining 2104 the stored hash value for the second data portion from the storage device based on the determination 2102 that the second data portion is unavailable, wherein the storage device stores the stored hash value in a primary header associated with the second data portion. As discussed above, stored hash values (such as stored CRC values) for a data portion may be available at a storage device that stores the data portion, even if other contents of the data portion are unavailable. Accordingly, in one example, where the second data portion is a Flash page, the stored hash value may be in a primary header, or primary page header for the Flash page. Even if the Flash page itself is unavailable, a storage controller may still be able to obtain, from the primary page header, a stored hash value or CRC value generated using data of the Flash page. Additionally, CRC values may be

mirrored across multiple storage locations within the data portion, within the storage device, or elsewhere in a storage system. In response to a determination that the second data portion is unavailable, the storage controller may be configured to search through these different locations to obtain the CRC value associated with the unavailable second data portion.

For further explanation, FIG. 22 sets forth a flow chart illustrating an example method of recovering corrupted data through speculative bitflip and cross-validation. The example method of FIG. 22 is similar to the example method of FIG. 17 in that the example method of FIG. 22 also includes altering 1702 a first data portion stored in a storage device, wherein the first data portion contains an error, calculating 1704 a hash value 1705 using the altered first data portion, comparing 1706 the hash value 1705 calculated using the altered first data portion to a stored hash value for the first data portion, and based on the comparing, determining 1708 whether the altering corrects the error in the first data portion.

The example method of FIG. 22 differs from the example method of FIG. 17 in that in the example method of FIG. 22 also includes determining 2202 that a first number of the plurality of data portions of the RAID stripe does not contain an error. Continuing with the example above of FIG. 20, an initial determination may be made that altered shard A1' 2011 and shard A_P' 2016 do not contain errors. As a more specific example, the storage controller may determine that shard A_P' 2016 does not contain errors because no errors have been identified in usage of shard A_P' 2016. By contrast, the storage controller may generate altered shard A1' 2011 based on knowledge that original shard A1' 2010 contained bit flip errors, compare the stored CRC value in shard A1' 2010 to a CRC value generated from altered shard A1' 2011 and, based on a match in values during the comparison, make an initial determination that altered shard A1' 2011 is likely to be free of errors.

As stated above, this may be an initial determination with respect to altered shard A1' 2011, not a final conclusion that this data portion is free of bit flip errors. This is because there may be a small but non-zero probability that two different inputs result in the same CRC value (i.e., a corrupted data portion and its altered version might both return the same CRC value in rare cases, even though the altered version still contains a bit flip error). For a more conclusive determination that altered shard A1' 2011 is free of bit flip errors, the example method of FIG. 22 also includes reconstructing 2204 a third data portion with parity calculations that are performed using the third data portion and a first subset of the plurality of data portions that includes the first data portion.

In the example of FIG. 21, where shard A2' 2012 was unavailable, CRC comparison was used to determine 55 whether the altering corrects the error in the first data portion. By contrast, here in FIG. 22 data may be available for shard A2' 2012 and for all other shards, or at least a sufficient number of shards that are needed to perform a RAID rebuild of a shard. More specifically, the first subset 60 of the plurality of data portions may refer to altered shard A1' 2011 and any additional shards that are required for parity calculations depending on the RAID configuration that is in effect. The third data portion may refer to, for example, shard A_Q' 2018. In other words, altered shard A1' 2011 and additional shards may be used in a RAID parity calculation that results in a reconstructed shard A_Q', referred to in FIG. 22 as shard A_Q' 2019.

81

The example method of FIG. 22 also includes performing 2206 a memory comparison between the third data portion and the reconstructed third data portion. Continuing with the example above, the method can include a comparison of the contents of shard A_Q 2018 and shard A_{Q'} 2019. If the memory comparison of shard A_Q 2018 and shard A_{Q'} 2019 indicates that these two shards include the same data, then it may be determined that the altering of shard A1 2010 to generate altered shard A1' 2011 resolved the bit flip error that resulted in a corrected shard (i.e., altered shard A1' 2011).

One or more embodiments may be described herein with the aid of method steps illustrating the performance of specified functions and relationships thereof. The boundaries and sequence of these functional building blocks and method steps have been arbitrarily defined herein for convenience of description. Alternate boundaries and sequences can be defined so long as the specified functions and relationships are appropriately performed. Any such alternate boundaries or sequences are thus within the scope and spirit of the claims. Further, the boundaries of these functional building blocks have been arbitrarily defined for convenience of description. Alternate boundaries could be defined as long as the certain significant functions are appropriately performed. Similarly, flow diagram blocks may also have been arbitrarily defined herein to illustrate certain significant functionality.

To the extent used, the flow diagram block boundaries and sequence could have been defined otherwise and still perform the certain significant functionality. Such alternate definitions of both functional building blocks and flow diagram blocks and sequences are thus within the scope and spirit of the claims. One of average skill in the art will also recognize that the functional building blocks, and other illustrative blocks, modules and components herein, can be implemented as illustrated or by discrete components, application specific integrated circuits, processors executing appropriate software and the like or any combination thereof.

While particular combinations of various functions and features of the one or more embodiments are expressly 40 described herein, other combinations of these features and functions are likewise possible. The present disclosure is not limited by the particular examples disclosed herein and expressly incorporates these other combinations.

What is claimed is:

1. A method comprising:

generating, after detection of an error caused by corrupted data in a first data portion in a storage device, an altered first data portion, wherein the first data portion is included within a plurality of data portions of a RAID stripe; 50 calculating a hash value using the altered first data portion; comparing the hash value calculated using the altered first data portion to a stored hash value for the first data portion; based on the comparing, determining that the altering corrects the error in the first data portion by reconstructing a second data portion of the RAID stripe using the altered first data portion to cross-validate the altered first data portion; and overwriting the first data portion using the altered first data portion.

2. The method of claim 1, wherein generating the altered first data portion further comprises flipping at least one bit in the first data portion.

82

3. The method of claim 2, wherein flipping the at least one bit further comprises flipping the at least one bit within a header portion of the first data portion.

4. The method of claim 1, wherein calculating the hash value using the altered first data portion further comprises calculating a cyclic redundancy check (CRC) value using the altered first data portion.

5. The method of claim 1 further comprising:
reconstructing a second data portion of the RAID stripe
with parity calculations that use the altered first data
portion;
calculating a second hash value using the reconstructed
second data portion;
determining that the calculated second hash value
matches a stored hash value within the second data
portion; and
based on the determination, determining that the altering
corrects the error in the first data portion.

6. The method of claim 5, further comprising:
determining that the second data portion is unavailable;
and
based on the determination, obtaining the stored hash
value for the second data portion from the storage
device, wherein the storage device stores the stored
hash value in a primary header associated with the
second data portion.

7. The method of claim 5, further comprising:
determining that a first number of the plurality of data
portions of the RAID stripe does not contain an error;
based on the determination, reconstructing a third data
portion with parity calculations that are performed
using the third data portion and a first subset of the
plurality of data portions that includes the altered first
data portion; and
performing a memory comparison between the third data
portion and the reconstructed third data portion.

8. An apparatus comprising:
a memory; and
a processing device operatively coupled to the memory,
the processing device configured to:
generate, after detection of an error caused by corrupted
data in a first data portion in a storage device, an altered
first data portion, wherein the first data portion is
included within a plurality of data portions of a RAID
stripe;
calculate a hash value using the altered first data
portion;
compare the hash value calculated using the altered first
data portion to a stored hash value for the first data
portion;
based on the comparing, determine that the altering
corrects the error in the first data portion by recon-
structing a second data portion of the RAID stripe
using the altered first data portion to cross-validate
the altered first data portion; and
overwrite the first data portion using the altered first
data portion.

9. The apparatus of claim 8, wherein the processing
device is further configured to flip at least one bit in the first
data portion.

10. The apparatus of claim 9, wherein the processing
device is further configured to flip the at least one bit within
a header portion of the first data portion.

11. The apparatus of claim 8, wherein the processing
device is further configured to calculate a cyclic redundancy
check (CRC) value using the altered first data portion.

83

12. The apparatus of claim **8**, wherein the processing device is further configured to:

reconstruct a second data portion of the RAID stripe with parity calculations that use the altered first data portion; calculate a second hash value using the reconstructed second data portion; determine that the calculated second hash value matches a stored hash value within the second data portion; and based on the determination, determine that the altering corrects the error in the first data portion.

13. The apparatus of claim **12**, wherein the processing device is further configured to:

determine that the second data portion is unavailable; and based on the determination, obtain the stored hash value for the second data portion from the storage device, wherein the storage device stores the stored hash value in a primary header associated with the second data portion.

14. The apparatus of claim **12**, wherein the processing device is further configured to:

determine that a first number of the plurality of data portions of the RAID stripe does not contain an error; based on the determination, reconstruct a third data portion with parity calculations that are performed using the third data portion and a first subset of the plurality of data portions that includes the altered first data portion; and

perform a memory comparison between the third data portion and the reconstructed third data portion.

15. A non-transitory computer readable storage medium storing instructions which, when executed, cause a processing device to:

generate, after detection of an error caused by corrupted data in a first data portion in a storage device, an altered first data portion, wherein the first data portion is included within a plurality of data portions of a RAID stripe;

calculate a hash value using the altered first data portion; compare the hash value calculated using the altered first data portion to a stored hash value for the first data portion;

based on the comparing, determine that the altering corrects the error in the first data portion by reconstructing

84

a second data portion of the RAID stripe using the altered first data portion to cross-validate the altered first data portion; and overwrite the first data portion using the altered first data portion.

16. The non-transitory computer readable storage medium of claim **15**, wherein the processing device is further configured to flip at least one bit in the first data portion.

17. The non-transitory computer readable storage medium of claim **15**, wherein the processing device is further configured to calculate a cyclic redundancy check (CRC) value using the altered first data portion.

18. The non-transitory computer readable storage medium of claim **15**, wherein the processing device is further configured to:

reconstruct a second data portion of the RAID stripe with parity calculations that use the altered first data portion; calculate a second hash value using the reconstructed second data portion; determine that the calculated second hash value matches a stored hash value within the second data portion; and based on the determination, determine that the altering corrects the error in the first data portion.

19. The non-transitory computer readable storage medium of claim **18**, wherein the processing device is further configured to:

determine that the second data portion is unavailable; and based on the determination, obtain the stored hash value for the second data portion from the storage device, wherein the storage device stores the stored hash value in a primary header associated with the second data portion.

20. The non-transitory computer readable storage medium of claim **18**, wherein the processing device is further configured to:

determine that a first number of the plurality of data portions of the RAID stripe does not contain an error; based on the determination, reconstruct a third data portion with parity calculations that are performed using the third data portion and a first subset of the plurality of data portions that includes the altered first data portion; and

perform a memory comparison between the third data portion and the reconstructed third data portion.

* * * * *