United States Patent	12393340
Kind Code	B2
Date of Patent	August 19, 2025
Inventor(s)	Karr; Ronald et al.

Latency reduction of flash-based devices using programming interrupts

Abstract

A read request with a high priority indication is received. A determination as to whether an in progress flash programming operation would delay processing the read request for a threshold amount of time is made. In response to determining that the in progress flash programming operation delays processing the read request for the threshold amount of time, the in progress flash programming operation is interrupted.

Inventors:	**Karr; Ronald (Palo Alto, CA), Kannan; Hari (Sunnyvale, CA), Lee; Robert (Pebble Beach, CA), Kirkpatrick; Peter E. (Los Altos, CA)**
Applicant:	**PURE STORAGE, INC.** (Santa Clara, CA)
Family ID:	**1000008762834**
Assignee:	**PURE STORAGE, INC. (Santa Clara, CA)**
Appl. No.:	**18/329341**
Filed:	**June 05, 2023**

Prior Publication Data

Document Identifier	Publication Date
US 20230393742 A1	Dec. 07, 2023

Related U.S. Application Data

continuation-in-part parent-doc US 17865854 20220715 US 11947795 child-doc US 18329341
continuation-in-part parent-doc US 16249745 20190116 US 11934322 child-doc US 18329341
us-provisional-application US 63471227 20230605
us-provisional-application US 63487868 20230301
us-provisional-application US 63436351 20221230
us-provisional-application US 63349488 20220606

Publication Classification

Int. Cl.: **G06F12/00** (20060101); **G06F3/06** (20060101)

U.S. Cl.:

CPC **G06F3/061** (20130101); **G06F3/0614** (20130101); **G06F3/0665** (20130101); **G06F3/067** (20130101);

Field of Classification Search

CPC: G06F (3/061); G06F (3/0614); G06F (3/0665); G06F (3/067); G06F (12/0238); G06F (2212/72)

References Cited

U.S. PATENT DOCUMENTS

Patent No.	Issued Date	Patentee Name	U.S. Cl.	CPC
5390327	12/1994	Lubbers et al.	N/A	N/A
5450581	12/1994	Bergen et al.	N/A	N/A
5479653	12/1994	Jones	N/A	N/A
5488731	12/1995	Mendelsohn	N/A	N/A
5504858	12/1995	Ellis et al.	N/A	N/A
5564113	12/1995	Bergen et al.	N/A	N/A
5574882	12/1995	Menon et al.	N/A	N/A
5649093	12/1996	Hanko et al.	N/A	N/A
5883909	12/1998	DeKoning et al.	N/A	N/A
6000010	12/1998	Legg	N/A	N/A
6260156	12/2000	Garvin et al.	N/A	N/A
6269453	12/2000	Krantz	N/A	N/A
6275898	12/2000	DeKoning	N/A	N/A
6453428	12/2001	Stephenson	N/A	N/A
6523087	12/2002	Busser	N/A	N/A
6535417	12/2002	Tsuda et al.	N/A	N/A
6643748	12/2002	Wieland	N/A	N/A
6725392	12/2003	Frey et al.	N/A	N/A
6763455	12/2003	Hall	N/A	N/A
6836816	12/2003	Kendall	N/A	N/A
6985995	12/2005	Holland et al.	N/A	N/A
7032125	12/2005	Holt et al.	N/A	N/A
7047358	12/2005	Lee et al.	N/A	N/A
7051155	12/2005	Talagala et al.	N/A	N/A
7055058	12/2005	Lee et al.	N/A	N/A
7065617	12/2005	Wang	N/A	N/A
7069383	12/2005	Yamamoto et al.	N/A	N/A
7076606	12/2005	Orsley	N/A	N/A
7107480	12/2005	Moshayedi et al.	N/A	N/A
7159150	12/2006	Kenchammana-Hosekote et al.	N/A	N/A

7162575	12/2006	Dalal et al.	N/A	N/A
7164608	12/2006	Lee	N/A	N/A
7188270	12/2006	Nanda et al.	N/A	N/A
7334156	12/2007	Land et al.	N/A	N/A
7370220	12/2007	Nguyen et al.	N/A	N/A
7386666	12/2007	Beauchamp et al.	N/A	N/A
7398285	12/2007	Kisley	N/A	N/A
7424498	12/2007	Patterson	N/A	N/A
7424592	12/2007	Karr et al.	N/A	N/A
7444532	12/2007	Masuyama et al.	N/A	N/A
7480658	12/2008	Heinla et al.	N/A	N/A
7484056	12/2008	Madnani et al.	N/A	N/A
7484057	12/2008	Madnani et al.	N/A	N/A
7484059	12/2008	Ofer et al.	N/A	N/A
7536506	12/2008	Ashmore et al.	N/A	N/A
7558859	12/2008	Kasiolas et al.	N/A	N/A
7565446	12/2008	Talagala et al.	N/A	N/A
7613947	12/2008	Coatney et al.	N/A	N/A
7634617	12/2008	Misra	N/A	N/A
7634618	12/2008	Misra	N/A	N/A
7681104	12/2009	Sim-Tang et al.	N/A	N/A
7681105	12/2009	Sim-Tang et al.	N/A	N/A
7681109	12/2009	Litsyn et al.	N/A	N/A
7730257	12/2009	Franklin	N/A	N/A
7730258	12/2009	Smith et al.	N/A	N/A
7730274	12/2009	Usgaonkar	N/A	N/A
7743276	12/2009	Jacobson et al.	N/A	N/A
7752489	12/2009	Deenadhayalan et al.	N/A	N/A
7757038	12/2009	Kitahara	N/A	N/A
7757059	12/2009	Ofer et al.	N/A	N/A
7778960	12/2009	Chatterjee et al.	N/A	N/A
7783955	12/2009	Murin	N/A	N/A
7814272	12/2009	Barrall et al.	N/A	N/A
7814273	12/2009	Barrall	N/A	N/A
7818531	12/2009	Barrall	N/A	N/A
7827351	12/2009	Suetsugu et al.	N/A	N/A
7827439	12/2009	Mathew et al.	N/A	N/A
7831768	12/2009	Ananthamurthy et al.	N/A	N/A
7856583	12/2009	Smith	N/A	N/A
7870105	12/2010	Arakawa et al.	N/A	N/A
7873878	12/2010	Belluomini et al.	N/A	N/A
7885938	12/2010	Greene et al.	N/A	N/A
7886111	12/2010	Klemm et al.	N/A	N/A
7908448	12/2010	Chatterjee et al.	N/A	N/A
7916538	12/2010	Jeon et al.	N/A	N/A
7921268	12/2010	Jakob	N/A	N/A
7930499	12/2010	Duchesne	N/A	N/A
7941697	12/2010	Mathew et al.	N/A	N/A

7958303	12/2010	Shuster	N/A	N/A
7971129	12/2010	Watson et al.	N/A	N/A
7975115	12/2010	Wayda et al.	N/A	N/A
7984016	12/2010	Kisley	N/A	N/A
7991822	12/2010	Bish et al.	N/A	N/A
8006126	12/2010	Deenadhayalan et al.	N/A	N/A
8010485	12/2010	Chatterjee et al.	N/A	N/A
8010829	12/2010	Chatterjee et al.	N/A	N/A
8020047	12/2010	Courtney	N/A	N/A
8046548	12/2010	Chatterjee et al.	N/A	N/A
8051361	12/2010	Sim-Tang et al.	N/A	N/A
8051362	12/2010	Li et al.	N/A	N/A
8074038	12/2010	Lionetti et al.	N/A	N/A
8082393	12/2010	Galloway et al.	N/A	N/A
8086603	12/2010	Nasre et al.	N/A	N/A
8086634	12/2010	Mimatsu	N/A	N/A
8086911	12/2010	Taylor	N/A	N/A
8090837	12/2011	Shin et al.	N/A	N/A
8108502	12/2011	Tabbara et al.	N/A	N/A
8117388	12/2011	Jernigan, IV	N/A	N/A
8117521	12/2011	Parker et al.	N/A	N/A
8140821	12/2011	Raizen et al.	N/A	N/A
8145838	12/2011	Miller et al.	N/A	N/A
8145840	12/2011	Koul et al.	N/A	N/A
8175012	12/2011	Chu et al.	N/A	N/A
8176360	12/2011	Frost et al.	N/A	N/A
8176405	12/2011	Hafner et al.	N/A	N/A
8180855	12/2011	Aiello et al.	N/A	N/A
8200922	12/2011	McKean et al.	N/A	N/A
8209469	12/2011	Carpenter et al.	N/A	N/A
8225006	12/2011	Karamcheti	N/A	N/A
8239618	12/2011	Kotzur et al.	N/A	N/A
8244999	12/2011	Chatterjee et al.	N/A	N/A
8261016	12/2011	Goel	N/A	N/A
8271455	12/2011	Kesselman	N/A	N/A
8285686	12/2011	Kesselman	N/A	N/A
8305811	12/2011	Jeon	N/A	N/A
8315999	12/2011	Chatley et al.	N/A	N/A
8327080	12/2011	Der	N/A	N/A
8335769	12/2011	Kesselman	N/A	N/A
8341118	12/2011	Drobychev et al.	N/A	N/A
8351290	12/2012	Huang et al.	N/A	N/A
8364920	12/2012	Parkison et al.	N/A	N/A
8365041	12/2012	Olbrich et al.	N/A	N/A
8375146	12/2012	Sinclair	N/A	N/A
8397016	12/2012	Talagala et al.	N/A	N/A
8402152	12/2012	Duran	N/A	N/A
8412880	12/2012	Leibowitz et al.	N/A	N/A
8423739	12/2012	Ash et al.	N/A	N/A

8429436	12/2012	Fillingim et al.	N/A	N/A
8452928	12/2012	Ofer et al.	N/A	N/A
8473698	12/2012	Lionetti et al.	N/A	N/A
8473778	12/2012	Simitci et al.	N/A	N/A
8473815	12/2012	Chung et al.	N/A	N/A
8479037	12/2012	Chatterjee et al.	N/A	N/A
8484414	12/2012	Sugimoto et al.	N/A	N/A
8498967	12/2012	Chatterjee et al.	N/A	N/A
8504797	12/2012	Mimatsu	N/A	N/A
8522073	12/2012	Cohen	N/A	N/A
8533408	12/2012	Madnani et al.	N/A	N/A
8533527	12/2012	Daikokuya et al.	N/A	N/A
8539177	12/2012	Madnani et al.	N/A	N/A
8544029	12/2012	Bakke et al.	N/A	N/A
8549224	12/2012	Zeryck et al.	N/A	N/A
8583861	12/2012	Ofer et al.	N/A	N/A
8589625	12/2012	Colgrove et al.	N/A	N/A
8595455	12/2012	Chatterjee et al.	N/A	N/A
8615599	12/2012	Takefman et al.	N/A	N/A
8627136	12/2013	Shankar et al.	N/A	N/A
8627138	12/2013	Clark et al.	N/A	N/A
8639669	12/2013	Douglis et al.	N/A	N/A
8639863	12/2013	Kanapathippillai et al.	N/A	N/A
8640000	12/2013	Cypher	N/A	N/A
8650343	12/2013	Kanapathippillai et al.	N/A	N/A
8660131	12/2013	Vermunt et al.	N/A	N/A
8661218	12/2013	Piszczek et al.	N/A	N/A
8671072	12/2013	Shah et al.	N/A	N/A
8689042	12/2013	Kanapathippillai et al.	N/A	N/A
8700875	12/2013	Barron et al.	N/A	N/A
8706694	12/2013	Chatterjee et al.	N/A	N/A
8706914	12/2013	Duchesneau	N/A	N/A
8706932	12/2013	Kanapathippillai et al.	N/A	N/A
8712963	12/2013	Douglis et al.	N/A	N/A
8713405	12/2013	Healey, Jr. et al.	N/A	N/A
8719621	12/2013	Karmarkar	N/A	N/A
8725730	12/2013	Keeton et al.	N/A	N/A
8751859	12/2013	Becker-Szendy et al.	N/A	N/A
8756387	12/2013	Frost et al.	N/A	N/A
8762793	12/2013	Grube et al.	N/A	N/A
8769232	12/2013	Suryabudi et al.	N/A	N/A
8775858	12/2013	Gower et al.	N/A	N/A
8775868	12/2013	Colgrove et al.	N/A	N/A
8788913	12/2013	Xin et al.	N/A	N/A
8793447	12/2013	Usgaonkar et al.	N/A	N/A
8799746	12/2013	Baker et al.	N/A	N/A

8819311	12/2013	Liao	N/A	N/A
8819383	12/2013	Jobanputra et al.	N/A	N/A
8822155	12/2013	Sukumar et al.	N/A	N/A
8824261	12/2013	Miller et al.	N/A	N/A
8832528	12/2013	Thatcher et al.	N/A	N/A
8838541	12/2013	Camble et al.	N/A	N/A
8838892	12/2013	Li	N/A	N/A
8843700	12/2013	Salessi et al.	N/A	N/A
8850108	12/2013	Hayes et al.	N/A	N/A
8850288	12/2013	Lazier et al.	N/A	N/A
8856593	12/2013	Eckhardt et al.	N/A	N/A
8856619	12/2013	Cypher	N/A	N/A
8862617	12/2013	Kesselman	N/A	N/A
8862847	12/2013	Feng et al.	N/A	N/A
8862928	12/2013	Xavier et al.	N/A	N/A
8868825	12/2013	Hayes et al.	N/A	N/A
8874836	12/2013	Hayes et al.	N/A	N/A
8880793	12/2013	Nagineni	N/A	N/A
8880825	12/2013	Lionetti et al.	N/A	N/A
8886778	12/2013	Nedved et al.	N/A	N/A
8898383	12/2013	Yamamoto et al.	N/A	N/A
8898388	12/2013	Kimmel	N/A	N/A
8904231	12/2013	Coatney et al.	N/A	N/A
8918478	12/2013	Ozzie et al.	N/A	N/A
8930307	12/2014	Colgrove et al.	N/A	N/A
8930633	12/2014	Amit et al.	N/A	N/A
8943357	12/2014	Atzmony	N/A	N/A
8949502	12/2014	McKnight et al.	N/A	N/A
8959110	12/2014	Smith et al.	N/A	N/A
8959388	12/2014	Kuang et al.	N/A	N/A
8972478	12/2014	Storer et al.	N/A	N/A
8972779	12/2014	Lee et al.	N/A	N/A
8977597	12/2014	Ganesh et al.	N/A	N/A
8996828	12/2014	Kalos et al.	N/A	N/A
9003144	12/2014	Hayes et al.	N/A	N/A
9009724	12/2014	Gold et al.	N/A	N/A
9021053	12/2014	Bembo et al.	N/A	N/A
9021215	12/2014	Meir et al.	N/A	N/A
9025393	12/2014	Wu et al.	N/A	N/A
9043372	12/2014	Makkar et al.	N/A	N/A
9047214	12/2014	Northcott	N/A	N/A
9053808	12/2014	Sprouse et al.	N/A	N/A
9058155	12/2014	Cepulis et al.	N/A	N/A
9063895	12/2014	Madnani et al.	N/A	N/A
9063896	12/2014	Madnani et al.	N/A	N/A
9098211	12/2014	Madnani et al.	N/A	N/A
9110898	12/2014	Chamness et al.	N/A	N/A
9110964	12/2014	Shilane et al.	N/A	N/A
9116819	12/2014	Cope et al.	N/A	N/A
9117536	12/2014	Yoon et al.	N/A	N/A

9122401	12/2014	Zaltsman et al.	N/A	N/A
9123422	12/2014	Yu et al.	N/A	N/A
9124300	12/2014	Sharon et al.	N/A	N/A
9134908	12/2014	Horn et al.	N/A	N/A
9153337	12/2014	Sutardja	N/A	N/A
9158472	12/2014	Kesselman et al.	N/A	N/A
9159422	12/2014	Lee et al.	N/A	N/A
9164891	12/2014	Karamcheti et al.	N/A	N/A
9183136	12/2014	Kawamura et al.	N/A	N/A
9189650	12/2014	Jaye et al.	N/A	N/A
9201733	12/2014	Verma et al.	N/A	N/A
9207876	12/2014	Shu et al.	N/A	N/A
9229656	12/2015	Contreras et al.	N/A	N/A
9229810	12/2015	He et al.	N/A	N/A
9235475	12/2015	Shilane et al.	N/A	N/A
9244626	12/2015	Shah et al.	N/A	N/A
9250999	12/2015	Barroso	N/A	N/A
9251066	12/2015	Colgrove et al.	N/A	N/A
9268648	12/2015	Barash et al.	N/A	N/A
9268806	12/2015	Kesselman	N/A	N/A
9280678	12/2015	Redberg	N/A	N/A
9286002	12/2015	Karamcheti et al.	N/A	N/A
9292214	12/2015	Kalos et al.	N/A	N/A
9298760	12/2015	Li et al.	N/A	N/A
9304908	12/2015	Karamcheti et al.	N/A	N/A
9311969	12/2015	Sharon et al.	N/A	N/A
9311970	12/2015	Sharon et al.	N/A	N/A
9323663	12/2015	Karamcheti et al.	N/A	N/A
9323667	12/2015	Bennett	N/A	N/A
9323681	12/2015	Apostolides et al.	N/A	N/A
9335942	12/2015	Kumar et al.	N/A	N/A
9348538	12/2015	Mallaiah et al.	N/A	N/A
9355022	12/2015	Ravimohan et al.	N/A	N/A
9384082	12/2015	Lee et al.	N/A	N/A
9384252	12/2015	Akirav et al.	N/A	N/A
9389958	12/2015	Sundaram et al.	N/A	N/A
9390019	12/2015	Patterson et al.	N/A	N/A
9395922	12/2015	Nishikido et al.	N/A	N/A
9396202	12/2015	Drobychev et al.	N/A	N/A
9400828	12/2015	Kesselman et al.	N/A	N/A
9405478	12/2015	Koseki et al.	N/A	N/A
9411685	12/2015	Lee	N/A	N/A
9417960	12/2015	Cai et al.	N/A	N/A
9417963	12/2015	He et al.	N/A	N/A
9430250	12/2015	Hamid et al.	N/A	N/A
9430542	12/2015	Akirav et al.	N/A	N/A
9432541	12/2015	Ishida	N/A	N/A
9454434	12/2015	Sundaram et al.	N/A	N/A
9471579	12/2015	Natanzon	N/A	N/A
9477554	12/2015	Hayes et al.	N/A	N/A

9477632	12/2015	Du	N/A	N/A
9501398	12/2015	George et al.	N/A	N/A
9525737	12/2015	Friedman	N/A	N/A
9529542	12/2015	Friedman et al.	N/A	N/A
9535631	12/2016	Fu et al.	N/A	N/A
9552248	12/2016	Miller et al.	N/A	N/A
9552291	12/2016	Munetoh et al.	N/A	N/A
9552299	12/2016	Stalzer	N/A	N/A
9563517	12/2016	Natanzon et al.	N/A	N/A
9588698	12/2016	Karamcheti et al.	N/A	N/A
9588712	12/2016	Kalos et al.	N/A	N/A
9594652	12/2016	Sathiamoorthy et al.	N/A	N/A
9600193	12/2016	Ahrens et al.	N/A	N/A
9619321	12/2016	Haratsch et al.	N/A	N/A
9619430	12/2016	Kannan et al.	N/A	N/A
9645754	12/2016	Li et al.	N/A	N/A
9667720	12/2016	Bent et al.	N/A	N/A
9710535	12/2016	Aizman et al.	N/A	N/A
9733840	12/2016	Karamcheti et al.	N/A	N/A
9734225	12/2016	Akirav et al.	N/A	N/A
9740403	12/2016	Storer et al.	N/A	N/A
9740700	12/2016	Chopra et al.	N/A	N/A
9740762	12/2016	Horowitz et al.	N/A	N/A
9747319	12/2016	Bestler et al.	N/A	N/A
9747320	12/2016	Kesselman	N/A	N/A
9767130	12/2016	Bestler et al.	N/A	N/A
9781227	12/2016	Friedman et al.	N/A	N/A
9785498	12/2016	Misra et al.	N/A	N/A
9798486	12/2016	Singh	N/A	N/A
9804925	12/2016	Carmi et al.	N/A	N/A
9811285	12/2016	Karamcheti et al.	N/A	N/A
9811546	12/2016	Bent et al.	N/A	N/A
9818478	12/2016	Chung	N/A	N/A
9829066	12/2016	Thomas et al.	N/A	N/A
9836245	12/2016	Hayes et al.	N/A	N/A
9891854	12/2017	Munetoh et al.	N/A	N/A
9891860	12/2017	Delgado et al.	N/A	N/A
9892005	12/2017	Kedem et al.	N/A	N/A
9892186	12/2017	Akirav et al.	N/A	N/A
9904589	12/2017	Donlan et al.	N/A	N/A
9904717	12/2017	Anglin et al.	N/A	N/A
9910748	12/2017	Pan	N/A	N/A
9910904	12/2017	Anglin et al.	N/A	N/A
9934237	12/2017	Shilane et al.	N/A	N/A
9940065	12/2017	Kalos et al.	N/A	N/A
9946604	12/2017	Glass	N/A	N/A
9952809	12/2017	Shah	N/A	N/A
9959167	12/2017	Donlan et al.	N/A	N/A
9965539	12/2017	D'Halluin et al.	N/A	N/A
9998539	12/2017	Brock et al.	N/A	N/A

10007457	12/2017	Hayes et al.	N/A	N/A
10013177	12/2017	Liu et al.	N/A	N/A
10013311	12/2017	Sundaram et al.	N/A	N/A
10019314	12/2017	Yang et al.	N/A	N/A
10019317	12/2017	Usvyatsky et al.	N/A	N/A
10031703	12/2017	Natanzon et al.	N/A	N/A
10061512	12/2017	Lin	N/A	N/A
10073626	12/2017	Karamcheti et al.	N/A	N/A
10082985	12/2017	Hayes et al.	N/A	N/A
10089012	12/2017	Chen et al.	N/A	N/A
10089174	12/2017	Yang	N/A	N/A
10089176	12/2017	Donlan et al.	N/A	N/A
10108819	12/2017	Donlan et al.	N/A	N/A
10146787	12/2017	Bashyam et al.	N/A	N/A
10152268	12/2017	Chakraborty et al.	N/A	N/A
10157098	12/2017	Yang et al.	N/A	N/A
10162704	12/2017	Kirschner et al.	N/A	N/A
10180875	12/2018	Klein	N/A	N/A
10185730	12/2018	Bestler et al.	N/A	N/A
10235065	12/2018	Miller et al.	N/A	N/A
10324639	12/2018	Seo	N/A	N/A
10567406	12/2019	Astigarraga et al.	N/A	N/A
10846137	12/2019	Vallala et al.	N/A	N/A
10877683	12/2019	Wu et al.	N/A	N/A
11076509	12/2020	Alissa et al.	N/A	N/A
11106810	12/2020	Natanzon et al.	N/A	N/A
11194707	12/2020	Stalzer	N/A	N/A
2001/0010605	12/2000	Aoki	N/A	G11B 20/1816
2002/0144059	12/2001	Kendall	N/A	N/A
2003/0105984	12/2002	Masuyama et al.	N/A	N/A
2003/0110205	12/2002	Johnson	N/A	N/A
2004/0161086	12/2003	Buntin et al.	N/A	N/A
2005/0001652	12/2004	Malik et al.	N/A	N/A
2005/0076228	12/2004	Davis et al.	N/A	N/A
2005/0235132	12/2004	Karr et al.	N/A	N/A
2005/0278460	12/2004	Shin et al.	N/A	N/A
2005/0283649	12/2004	Turner et al.	N/A	N/A
2006/0015683	12/2005	Ashmore et al.	N/A	N/A
2006/0114930	12/2005	Lucas et al.	N/A	N/A
2006/0174157	12/2005	Barrall et al.	N/A	N/A
2006/0248294	12/2005	Nedved et al.	N/A	N/A
2007/0079068	12/2006	Draggon	N/A	N/A
2007/0214194	12/2006	Reuter	N/A	N/A
2007/0214314	12/2006	Reuter	N/A	N/A
2007/0234016	12/2006	Davis et al.	N/A	N/A
2007/0239926	12/2006	Gyl et al.	N/A	N/A
2007/0268905	12/2006	Baker et al.	N/A	N/A
2008/0080709	12/2007	Michtchenko et al.	N/A	N/A
2008/0107274	12/2007	Worthy	N/A	N/A

2008/0155191	12/2007	Anderson et al.	N/A	N/A
2008/0256141	12/2007	Wayda et al.	N/A	N/A
2008/0295118	12/2007	Liao	N/A	N/A
2009/0077208	12/2008	Nguyen et al.	N/A	N/A
2009/0138654	12/2008	Sutardja	N/A	N/A
2009/0216910	12/2008	Duchesneau	N/A	N/A
2009/0216920	12/2008	Lauterbach et al.	N/A	N/A
2010/0017444	12/2009	Chatterjee et al.	N/A	N/A
2010/0023673	12/2009	Struk	711/E12.008	G06F 12/08
2010/0042636	12/2009	Lu	N/A	N/A
2010/0094806	12/2009	Apostolides et al.	N/A	N/A
2010/0115070	12/2009	Missimilly	N/A	N/A
2010/0125695	12/2009	Wu et al.	N/A	N/A
2010/0162076	12/2009	Sim-Tang et al.	N/A	N/A
2010/0169707	12/2009	Mathew et al.	N/A	N/A
2010/0174576	12/2009	Naylor	N/A	N/A
2010/0268908	12/2009	Ouyang et al.	N/A	N/A
2010/0306500	12/2009	Mimatsu	N/A	N/A
2011/0035540	12/2010	Fitzgerald et al.	N/A	N/A
2011/0040925	12/2010	Frost et al.	N/A	N/A
2011/0055453	12/2010	Bennett	711/E12.001	G11C 16/26
2011/0060927	12/2010	Fillingim et al.	N/A	N/A
2011/0119462	12/2010	Leach et al.	N/A	N/A
2011/0219170	12/2010	Frost et al.	N/A	N/A
2011/0238625	12/2010	Hamaguchi et al.	N/A	N/A
2011/0264843	12/2010	Haines et al.	N/A	N/A
2011/0302369	12/2010	Goto et al.	N/A	N/A
2012/0011398	12/2011	Eckhardt et al.	N/A	N/A
2012/0079318	12/2011	Colgrove et al.	N/A	N/A
2012/0089567	12/2011	Takahashi et al.	N/A	N/A
2012/0110249	12/2011	Jeong et al.	N/A	N/A
2012/0131253	12/2011	McKnight et al.	N/A	N/A
2012/0158923	12/2011	Mohamed et al.	N/A	N/A
2012/0191900	12/2011	Kunimatsu et al.	N/A	N/A
2012/0198152	12/2011	Terry et al.	N/A	N/A
2012/0198261	12/2011	Brown et al.	N/A	N/A
2012/0209943	12/2011	Jung	N/A	N/A
2012/0226934	12/2011	Rao	N/A	N/A
2012/0246435	12/2011	Meir et al.	N/A	N/A
2012/0260055	12/2011	Murase	N/A	N/A
2012/0311557	12/2011	Resch	N/A	N/A
2013/0022201	12/2012	Glew et al.	N/A	N/A
2013/0036314	12/2012	Glew et al.	N/A	N/A
2013/0042056	12/2012	Shats et al.	N/A	N/A
2013/0060884	12/2012	Bernbo et al.	N/A	N/A
2013/0067188	12/2012	Mehra et al.	N/A	N/A
2013/0073894	12/2012	Xavier et al.	N/A	N/A
2013/0124776	12/2012	Hallak et al.	N/A	N/A
2013/0132800	12/2012	Healey, Jr. et al.	N/A	N/A
2013/0151653	12/2012	Sawicki et al.	N/A	N/A

2013/0151771	12/2012	Tsukahara et al.	N/A	N/A
2013/0173853	12/2012	Ungureanu et al.	N/A	N/A
2013/0238554	12/2012	Yucel et al.	N/A	N/A
2013/0339314	12/2012	Carpentier et al.	N/A	N/A
2013/0339635	12/2012	Amit et al.	N/A	N/A
2013/0339818	12/2012	Baker et al.	N/A	N/A
2014/0040535	12/2013	Lee et al.	N/A	N/A
2014/0040702	12/2013	He et al.	N/A	N/A
2014/0047263	12/2013	Coatney et al.	N/A	N/A
2014/0047269	12/2013	Kim	N/A	N/A
2014/0063721	12/2013	Herman et al.	N/A	N/A
2014/0064048	12/2013	Cohen et al.	N/A	N/A
2014/0068224	12/2013	Fan et al.	N/A	N/A
2014/0075252	12/2013	Luo et al.	N/A	N/A
2014/0122510	12/2013	Namkoong et al.	N/A	N/A
2014/0136880	12/2013	Shankar et al.	N/A	N/A
2014/0181402	12/2013	White	N/A	N/A
2014/0220561	12/2013	Sukumar et al.	N/A	N/A
2014/0237164	12/2013	Le et al.	N/A	N/A
2014/0279936	12/2013	Bernbo et al.	N/A	N/A
2014/0280025	12/2013	Eidson et al.	N/A	N/A
2014/0289588	12/2013	Nagadomi et al.	N/A	N/A
2014/0330785	12/2013	Isherwood et al.	N/A	N/A
2014/0372838	12/2013	Lou et al.	N/A	N/A
2014/0380125	12/2013	Calder et al.	N/A	N/A
2014/0380126	12/2013	Yekhanin et al.	N/A	N/A
2015/0032720	12/2014	James	N/A	N/A
2015/0039645	12/2014	Lewis	N/A	N/A
2015/0039849	12/2014	Lewis	N/A	N/A
2015/0089283	12/2014	Kermarrec et al.	N/A	N/A
2015/0100746	12/2014	Rychlik et al.	N/A	N/A
2015/0134824	12/2014	Mickens et al.	N/A	N/A
2015/0153800	12/2014	Lucas et al.	N/A	N/A
2015/0154111	12/2014	D'Abreu	714/6.11	G06F 12/0246
2015/0154418	12/2014	Redberg	N/A	N/A
2015/0180714	12/2014	Chunn et al.	N/A	N/A
2015/0280959	12/2014	Vincent	N/A	N/A
2016/0026397	12/2015	Nishikido et al.	N/A	N/A
2016/0182542	12/2015	Staniford	N/A	N/A
2016/0191508	12/2015	Bestler et al.	N/A	N/A
2016/0246537	12/2015	Kim	N/A	N/A
2016/0248631	12/2015	Duchesneau	N/A	N/A
2016/0378612	12/2015	Hipsh et al.	N/A	N/A
2017/0091236	12/2016	Hayes et al.	N/A	N/A
2017/0103092	12/2016	Hu et al.	N/A	N/A
2017/0103094	12/2016	Hu et al.	N/A	N/A
2017/0103098	12/2016	Hu et al.	N/A	N/A
2017/0103116	12/2016	Hu et al.	N/A	N/A
2017/0177236	12/2016	Haratsch et al.	N/A	N/A

2017/0262202	12/2016	Seo	N/A	N/A
2018/0039442	12/2017	Shadrin et al.	N/A	N/A
2018/0054454	12/2017	Astigarraga et al.	N/A	N/A
2018/0081958	12/2017	Akirav et al.	N/A	N/A
2018/0101441	12/2017	Hyun et al.	N/A	N/A
2018/0101587	12/2017	Anglin et al.	N/A	N/A
2018/0101588	12/2017	Anglin et al.	N/A	N/A
2018/0129445	12/2017	Shin	N/A	G06F 3/0611
2018/0217756	12/2017	Liu et al.	N/A	N/A
2018/0307560	12/2017	Vishnumolakala et al.	N/A	N/A
2018/0321874	12/2017	Li et al.	N/A	N/A
2019/0036703	12/2018	Bestler	N/A	N/A
2019/0080773	12/2018	Kondo	N/A	N/A
2019/0220315	12/2018	Vallala et al.	N/A	N/A
2020/0034560	12/2019	Natanzon et al.	N/A	N/A
2020/0326871	12/2019	Wu et al.	N/A	N/A
2021/0360833	12/2020	Alissa et al.	N/A	N/A

FOREIGN PATENT DOCUMENTS

Patent No.	**Application Date**	**Country**	**CPC**
2164006	12/2009	EP	N/A
2256621	12/2009	EP	N/A
0213033	12/2001	WO	N/A
2008103569	12/2007	WO	N/A
2008157081	12/2007	WO	N/A
2013032825	12/2012	WO	N/A

OTHER PUBLICATIONS

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. cited by applicant
International Search Report and Written Opinion, PCT/US2015/018169, May 15, 2015, 10 pages. cited by applicant
International Search Report and Written Opinion, PCT/US2015/034291, Sep. 30, 2015, 3 pages. cited by applicant
International Search Report and Written Opinion, PCT/US2015/034302, Sep. 11, 2015, 10 pages. cited by applicant
International Search Report and Written Opinion, PCT/US2015/039135, Sep. 18, 2015, 8 pages. cited by applicant
International Search Report and Written Opinion, PCT/US2015/039136, Sep. 23, 2015, 7 pages. cited by applicant
International Search Report and Written Opinion, PCT/US2015/039137, Oct. 1, 2015, 8 pages. cited by applicant
International Search Report and Written Opinion, PCT/US2015/039142, Sep. 24, 2015, 3 pages. cited by applicant
International Search Report and Written Opinion, PCT/US2015/044370, Dec. 15, 2015, 3 pages.

cited by applicant
International Search Report and Written Opinion, PCT/US2016/014356, Jun. 28, 2016, 3 pages. cited by applicant
International Search Report and Written Opinion, PCT/US2016/014357, Jun. 29, 2016, 3 pages. cited by applicant
International Search Report and Written Opinion, PCT/US2016/014361, May 30, 2016, 3 pages. cited by applicant
International Search Report and Written Opinion, PCT/US2016/014604, May 19, 2016, 3 pages. cited by applicant
International Search Report and Written Opinion, PCT/US2016/016504, Jul. 6, 2016, 7 pages. cited by applicant
International Search Report and Written Opinion, PCT/US2016/023485, Jul. 21, 2016, 13 pages. cited by applicant
International Search Report and Written Opinion, PCT/US2016/024391, Jul. 12, 2016, 11 pages. cited by applicant
International Search Report and Written Opinion, PCT/US2016/026529, Jul. 19, 2016, 9 pages. cited by applicant
International Search Report and Written Opinion, PCT/US2016/031039, Aug. 18, 2016, 7 pages. cited by applicant
International Search Report and Written Opinion, PCT/US2016/033306, Aug. 19, 2016, 11 pages. cited by applicant
International Search Report and Written Opinion, PCT/US2016/047808, Nov. 25, 2016, 14 pages. cited by applicant
Kim et al., “Data Access Frequency based Data Replication Method using Erasure Codes in Cloud Storage System”, Journal of the Institute of Electronics and Information Engineers, Feb. 2014, vol. 51, No. 2, 7 pages. cited by applicant
Schmid, “RAID Scaling Charts, Part 3:4-128 kB Stripes Compared”, Tom's Hardware, Nov. 27, 2007, URL: http://www.tomshardware.com/reviews/RAID-SCALING-CHARTS.1735-4.html, 24 pages. cited by applicant
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. cited by applicant
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. cited by applicant
International Search Report and Written Opinion, PCT/US2023/024557, Sep. 27, 2023, 9 pages. cited by applicant

---

Primary Examiner: Gu; Shawn X

Background/Summary

CROSS REFERENCE TO RELATED APPLICATIONS (1) This is a non-provisional application for patent entitled to a filing date and claiming the benefit of earlier-filed U.S. Provisional Patent Application No. 63/349,488, filed Jun. 6, 2022, herein incorporated by reference in its entirety.

BRIEF DESCRIPTION OF DRAWINGS
(1) FIG. **1**A illustrates a first example system for data storage in accordance with some

implementations.
(2) FIG. **1**B illustrates a second example system for data storage in accordance with some implementations.
(3) FIG. **1**C illustrates a third example system for data storage in accordance with some implementations.
(4) FIG. **1**D illustrates a fourth example system for data storage in accordance with some implementations.
(5) FIG. **2**A is a perspective view of a storage cluster with multiple storage nodes and internal storage coupled to each storage node to provide network attached storage, in accordance with some embodiments.
(6) FIG. **2**B is a block diagram showing an interconnect switch coupling multiple storage nodes in accordance with some embodiments.
(7) FIG. **2**C is a multiple level block diagram, showing contents of a storage node and contents of one of the non-volatile solid state storage units in accordance with some embodiments.
(8) FIG. **2**D shows a storage server environment, which uses embodiments of the storage nodes and storage units of some previous figures in accordance with some embodiments.
(9) FIG. **2**E is a blade hardware block diagram, showing a control plane, compute and storage planes, and authorities interacting with underlying physical resources, in accordance with some embodiments.
(10) FIG. **2**F depicts elasticity software layers in blades of a storage cluster, in accordance with some embodiments.
(11) FIG. **2**G depicts authorities and storage resources in blades of a storage cluster, in accordance with some embodiments.
(12) FIG. **3**A sets forth a diagram of a storage system that is coupled for data communications with a cloud services provider in accordance with some embodiments of the present disclosure.
(13) FIG. **3**B sets forth a diagram of a storage system in accordance with some embodiments of the present disclosure.
(14) FIG. **3**C sets forth an example of a cloud-based storage system in accordance with some embodiments of the present disclosure.
(15) FIG. **3**D illustrates an exemplary computing device that may be specifically configured to perform one or more of the processes described herein.
(16) FIG. **3**E illustrates an example of a fleet of storage systems for providing storage services, in accordance with embodiments of the disclosure.
(17) FIG. **3**F illustrates an example container system, in accordance with embodiments of the disclosure.
(18) FIG. **4** is a block diagram illustration an example storage system in accordance with some embodiments of the disclosure.
(19) FIG. **5**A is an illustration of an example of a scheduler of a storage system receiving a request to read data from a storage device of the storage system, in accordance with embodiments of the disclosure.
(20) FIG. **5**B is an illustration of an example of a scheduler of a storage system receiving queue information in accordance with embodiments of the disclosure.
(21) FIG. **6**A is an illustration of an example of a scheduler of a storage system inserting a read operation into a queue of a storage device, in accordance with some embodiments of the disclosure.
(22) FIG. **6**B is an illustration of an example of a scheduler of a storage system inserting a read operation into a queue of a storage device, in accordance with embodiments of the disclosure.
(23) FIG. **7** is an illustration of a timeline of a storage device of a storage system performing read operations during a multiple stage write operation, in accordance with embodiments of the disclosure.
(24) FIG. **8** is an illustration of a timeline of a storage device of a storage system suspending a

write operation to perform a read operation, in accordance with embodiments of the disclosure.
(25) FIG. **9** is an example method to intelligently process read requests in accordance with embodiments of the disclosure.
(26) FIG. **10** is an illustration of an example of reducing latency of a storage system using a die-aware scheduler, in accordance with embodiments of the disclosure.
(27) FIG. **11** is an example method to improve storage system latency using differing programming modes in accordance with embodiments of the disclosure.
(28) FIG. **12** is an example method to optimize data storage using different flash programming modes, in accordance with embodiments of the disclosure.
(29) FIG. **13** is an example method to intelligently direct flash operations of a modular storage system, in accordance with embodiments of the disclosure.

Description

DESCRIPTION OF EMBODIMENTS
(1) Example methods, apparatus, and products for intelligently scheduling operations of a storage system in accordance with embodiments of the present disclosure are described with reference to the accompanying drawings, beginning with FIG. **1**A. FIG. **1**A illustrates an example system for data storage, in accordance with some implementations. System **100** (also referred to as “storage system” herein) includes numerous elements for purposes of illustration rather than limitation. It may be noted that system **100** may include the same, more, or fewer elements configured in the same or different manner in other implementations.
(2) System **100** includes a number of computing devices **164**A-B. Computing devices (also referred to as “client devices” herein) may be embodied, for example, a server in a data center, a workstation, a personal computer, a notebook, or the like. Computing devices **164**A-B may be coupled for data communications to one or more storage arrays **102**A-B through a storage area network (‘SAN’) **158** or a local area network (‘LAN’) **160**.
(3) The SAN **158** may be implemented with a variety of data communications fabrics, devices, and protocols. For example, the fabrics for SAN **158** may include Fibre Channel, Ethernet, Infiniband, Serial Attached Small Computer System Interface (‘SAS’), or the like. Data communications protocols for use with SAN **158** may include Advanced Technology Attachment (‘ATA’), Fibre Channel Protocol, Small Computer System Interface (‘SCSI’), Internet Small Computer System Interface (‘iSCSI’), HyperSCSI, Non-Volatile Memory Express (‘NVMe’) over Fabrics, or the like. It may be noted that SAN **158** is provided for illustration, rather than limitation. Other data communication couplings may be implemented between computing devices **164**A-B and storage arrays **102**A-B.
(4) 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.
(5) Storage arrays **102**A-B may provide persistent data storage for the computing devices **164**A-B. Storage array **102**A may be contained in a chassis (not shown), and storage array **102**B may be contained in another chassis (not shown), in some implementations. Storage array **102**A and **102**B may include one or more storage array controllers **110**A-D (also referred to as “controller” herein). A storage array controller **110**A-D may be embodied as a module of automated computing machinery comprising computer hardware, computer software, or a combination of computer hardware and software. In some implementations, the storage array controllers **110**A-D may be

configured to carry out various storage tasks. Storage tasks may include writing data received from the computing devices **164**A-B to storage array **102**A-B, erasing data from storage array **102**A-B, retrieving data from storage array **102**A-B and providing data to computing devices **164**A-B, monitoring and reporting of 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.
(6) Storage array controller **110**A-D may be implemented in a variety of ways, including as a Field Programmable Gate Array (‘FPGA’), a Programmable Logic Chip (‘PLC’), an Application Specific Integrated Circuit (‘ASIC’), System-on-Chip (‘SOC’), or any computing device that includes discrete components such as a processing device, central processing unit, computer memory, or various adapters. Storage array controller **110**A-D may include, for example, a data communications adapter configured to support communications via the SAN **158** or LAN **160**. In some implementations, storage array controller **110**A-D may be independently coupled to the LAN **160**. In some implementations, storage array controller **110**A-D may include an I/O controller or the like that couples the storage array controller **110**A-D for data communications, through a midplane (not shown), to a persistent storage resource **170**A-B (also referred to as a “storage resource” herein). The persistent storage resource **170**A-B may include any number of storage drives **171**A-F (also referred to as “storage devices” herein) and any number of non-volatile Random Access Memory (‘NVRAM’) devices (not shown).
(7) In some implementations, the NVRAM devices of a persistent storage resource **170**A-B may be configured to receive, from the storage array controller **110**A-D, data to be stored in the storage drives **171**A-F. In some examples, the data may originate from computing devices **164**A-B. In some examples, writing data to the NVRAM device may be carried out more quickly than directly writing data to the storage drive **171**A-F. In some implementations, the storage array controller **110**A-D may be configured to utilize the NVRAM devices as a quickly accessible buffer for data destined to be written to the storage drives **171**A-F. Latency for write requests using NVRAM devices as a buffer may be improved relative to a system in which a storage array controller **110**A-D writes data directly to the storage drives **171**A-F. In some implementations, the NVRAM devices may be implemented with computer memory in the form of high bandwidth, low latency RAM. The NVRAM device is referred to as “non-volatile” because the NVRAM device may receive or include a unique power source that maintains the state of the RAM after main power loss to the NVRAM device. Such a power source may be a battery, one or more capacitors, or the like. In response to a power loss, the NVRAM device may be configured to write the contents of the RAM to a persistent storage, such as the storage drives **171**A-F.
(8) In some implementations, storage drive **171**A-F may refer to any device configured to record data persistently, where “persistently” or “persistent” refers as to a device's ability to maintain recorded data after loss of power. In some implementations, storage drive **171**A-F may correspond to non-disk storage media. For example, the storage drive **171**A-F may be one or more solid-state drives (‘SSDs’), flash memory based storage, any type of solid-state non-volatile memory, or any other type of non-mechanical storage device. In other implementations, storage drive **171**A-F may include mechanical or spinning hard disk, such as hard-disk drives (‘HDD’).
(9) In some implementations, the storage array controllers **110**A-D may be configured for offloading device management responsibilities from storage drive **171**A-F in storage array **102**A-B. For example, storage array controllers **110**A-D may manage control information that may describe the state of one or more memory blocks in the storage drives **171**A-F. The control information may indicate, for example, that a particular memory block has failed and should no longer be written to, that a particular memory block contains boot code for a storage array controller **110**A-D, the number of program-erase (‘P/E’) cycles that have been performed on a particular memory block, the age of data stored in a particular memory block, the type of data that is stored in a particular memory block, and so forth. In some implementations, the control information may be stored with

an associated memory block as metadata. In other implementations, the control information for the storage drives **171**A-F may be stored in one or more particular memory blocks of the storage drives **171**A-F that are selected by the storage array controller **110**A-D. The selected memory blocks may be tagged with an identifier indicating that the selected memory block contains control information. The identifier may be utilized by the storage array controllers **110**A-D in conjunction with storage drives **171**A-F to quickly identify the memory blocks that contain control information. For example, the storage controllers **110**A-D may issue a command to locate memory blocks that contain control information. It may be noted that control information may be so large that parts of the control information may be stored in multiple locations, that the control information may be stored in multiple locations for purposes of redundancy, for example, or that the control information may otherwise be distributed across multiple memory blocks in the storage drives **171**A-F.
(10) In some implementations, storage array controllers **110**A-D may offload device management responsibilities from storage drives **171**A-F of storage array **102**A-B by retrieving, from the storage drives **171**A-F, control information describing the state of one or more memory blocks in the storage drives **171**A-F. Retrieving the control information from the storage drives **171**A-F may be carried out, for example, by the storage array controller **110**A-D querying the storage drives **171**A-F for the location of control information for a particular storage drive **171**A-F. The storage drives **171**A-F may be configured to execute instructions that enable the storage drives **171**A-F to identify the location of the control information. The instructions may be executed by a controller (not shown) associated with or otherwise located on the storage drive **171**A-F and may cause the storage drive **171**A-F to scan a portion of each memory block to identify the memory blocks that store control information for the storage drives **171**A-F. The storage drives **171**A-F may respond by sending a response message to the storage array controller **110**A-D that includes the location of control information for the storage drive **171**A-F. Responsive to receiving the response message, storage array controllers **110**A-D may issue a request to read data stored at the address associated with the location of control information for the storage drives **171**A-F.
(11) In other implementations, the storage array controllers **110**A-D may further offload device management responsibilities from storage drives **171**A-F by performing, in response to receiving the control information, a storage drive management operation. A storage drive management operation may include, for example, an operation that is typically performed by the storage drive **171**A-F (e.g., the controller (not shown) associated with a particular storage drive **171**A-F). A storage drive management operation may include, for example, ensuring that data is not written to failed memory blocks within the storage drive **171**A-F, ensuring that data is written to memory blocks within the storage drive **171**A-F in such a way that adequate wear leveling is achieved, and so forth.
(12) In some implementations, storage array **102**A-B may implement two or more storage array controllers **110**A-D. For example, storage array **102**A may include storage array controllers **110**A and storage array controllers **110**B. At a given instant, a single storage array controller **110**A-D (e.g., storage array controller **110**A) of a storage system **100** may be designated with primary status (also referred to as "primary controller" herein), and other storage array controllers **110**A-D (e.g., storage array controller **110**A) may be designated with secondary status (also referred to as "secondary controller" herein). The primary controller may have particular rights, such as permission to alter data in persistent storage resource **170**A-B (e.g., writing data to persistent storage resource **170**A-B). At least some of the rights of the primary controller may supersede the rights of the secondary controller. For instance, the secondary controller may not have permission to alter data in persistent storage resource **170**A-B when the primary controller has the right. The status of storage array controllers **110**A-D may change. For example, storage array controller **110**A may be designated with secondary status, and storage array controller **110**B may be designated with primary status.

(13) In some implementations, a primary controller, such as storage array controller **110**A, may serve as the primary controller for one or more storage arrays **102**A-B, and a second controller, such as storage array controller **110**B, may serve as the secondary controller for the one or more storage arrays **102**A-B. For example, storage array controller **110**A may be the primary controller for storage array **102**A and storage array **102**B, and storage array controller **110**B may be the secondary controller for storage array **102**A and **102**B. In some implementations, storage array controllers **110**C and **110**D (also referred to as "storage processing modules") may neither have primary or secondary status. Storage array controllers **110**C and **110**D, implemented as storage processing modules, may act as a communication interface between the primary and secondary controllers (e.g., storage array controllers **110**A and **110**B, respectively) and storage array **102**B. For example, storage array controller **110**A of storage array **102**A may send a write request, via SAN **158**, to storage array **102**B. The write request may be received by both storage array controllers **110**C and **110**D of storage array **102**B. Storage array controllers **110**C and **110**D facilitate the communication, e.g., send the write request to the appropriate storage drive **171**A-F. It may be noted that in some implementations storage processing modules may be used to increase the number of storage drives controlled by the primary and secondary controllers.
(14) In some implementations, storage array controllers **110**A-D are communicatively coupled, via a midplane (not shown), to one or more storage drives **171**A-F and to one or more NVRAM devices (not shown) that are included as part of a storage array **102**A-B. The storage array controllers **110**A-D may be coupled to the midplane via one or more data communication links and the midplane may be coupled to the storage drives **171**A-F and the NVRAM devices via one or more data communications links. The data communications links described herein are collectively illustrated by data communications links **108**A-D and may include a Peripheral Component Interconnect Express ('PCIe') bus, for example.
(15) FIG. **1**B illustrates an example system for data storage, in accordance with some implementations. Storage array controller **101** illustrated in FIG. **1**B may be similar to the storage array controllers **110**A-D described with respect to FIG. **1**A. In one example, storage array controller **101** may be similar to storage array controller **110**A or storage array controller **110**B. Storage array controller **101** includes numerous elements for purposes of illustration rather than limitation. It may be noted that storage array controller **101** may include the same, more, or fewer elements configured in the same or different manner in other implementations. It may be noted that elements of FIG. **1**A may be included below to help illustrate features of storage array controller **101**.
(16) 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.
(17) 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.
(18) 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 **105**A-C. In some implementations, host bus adapters **103**A-C may be computer hardware that connects a host system (e.g., the storage array controller) to other network and storage arrays. In some examples, host bus adapters **103**A-C may be a Fibre Channel adapter that enables the storage array controller **101** to connect to a SAN, an Ethernet adapter that enables the storage array controller **101** to connect to a LAN, or the like. Host bus adapters **103**A-C may be coupled to the processing device **104** via a data communications link **105**A-C such as, for example, a PCIe bus.
(19) 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.
(20) 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.
(21) 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.
(22) 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.
(23) 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.
(24) In some implementations, storage drive **171**A-F may be one or more zoned storage devices. In some implementations, the one or more zoned storage devices may be a shingled HDD. In 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.
(25) 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.
(26) 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.
(27) 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.
(28) 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 a written to further without first performing a zone reset operation.
(29) 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.
(30) 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.
(31) In some implementations, storage drive **171**A-F may support an abstraction that organizes virtual address space of the storage drive such that virtual segments have a size that enables the virtual segments to be written to sequentially within a same erase block of the storage device. The virtual segments are intended to be large enough to aid in reducing the overhead of managing flash memory at the erase block level, while allowing flexibility in how the storage drive **171**A-F is implemented. For example, erase blocks might have additional overhead that is not included in the virtually addressable segment. These segments might be large, but smaller than entire erase blocks, particularly if the erase blocks are a larger size. In embodiments, storage drive **171**A-F may first write a data segment in single level cell (SLC) mode and then later migrate it to a quad-level cell (QLC) mode segment if the data segment has survived for a threshold amount of time without being overwritten. In some embodiments, storage drive **171**A-F may have internal redundancy, such as internal RAID across erase blocks or across chips, to increase reliability. This may cause there to be an internal RAID-oriented segment that is much larger even than the erase blocks of storage drive **171**A-F. In an embodiment, RAID may be added within the storage drive **171**A-F to a set of written erase blocks after the written erase blocks have been idle for a threshold amount of time so that the RAID may not have much of an effect on the size of the sequentially written segments. However, there could be other overhead, such as the storage device controller writing additional redundancy or check codes or index metadata within the erase blocks that is not accounted for in the reported sequentially writable segment size.
(32) 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.
(33) 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.
(34) Advantages of the process being performed only by the operating system of the flash storage system include increased reliability of the flash drives of the flash storage system as unnecessary or redundant write operations are not being performed during the process. One possible point of novelty here is the concept of initiating and controlling the process at the operating system of the flash storage system. In addition, the process can be controlled by the operating system across multiple flash drives. This is in contrast to the process being performed by a storage controller of a flash drive.
(35) 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.
(36) FIG. **1**C illustrates a third example system **117** for data storage in accordance with some implementations. System **117** (also referred to as “storage system” herein) includes numerous

elements for purposes of illustration rather than limitation. It may be noted that system **117** may include the same, more, or fewer elements configured in the same or different manner in other implementations.
(37) In one embodiment, system **117** includes a dual Peripheral Component Interconnect ('PCI') flash storage device **118** with separately addressable fast write storage. System **117** may include a storage device controller **119**. In one embodiment, storage device controller **119**A-D may be a CPU, ASIC, FPGA, or any other circuitry that may implement control structures necessary according to the present disclosure. In one embodiment, system **117** includes flash memory devices (e.g., including flash memory devices **120***a-n*), operatively coupled to various channels of the storage device controller **119**. Flash memory devices **120***a-n* may be presented to the controller **119**A-D as an addressable collection of Flash pages, erase blocks, and/or control elements sufficient to allow the storage device controller **119**A-D to program and retrieve various aspects of the Flash. In one embodiment, storage device controller **119**A-D may perform operations on flash memory devices **120***a-n* including storing and retrieving data content of pages, arranging and erasing any blocks, tracking statistics related to the use and reuse of Flash memory pages, erase blocks, and cells, tracking and predicting error codes and faults within the Flash memory, controlling voltage levels associated with programming and retrieving contents of Flash cells, etc.
(38) In one embodiment, system **117** may include RAM **121** to store separately addressable fast-write data. In one embodiment, RAM **121** may be one or more separate discrete devices. In another embodiment, RAM **121** may be integrated into storage device controller **119**A-D or multiple storage device controllers. The RAM **121** may be utilized for other purposes as well, such as temporary program memory for a processing device (e.g., a CPU) in the storage device controller **119**.
(39) In one embodiment, system **117** may include a stored energy device **122**, such as a rechargeable battery or a capacitor. Stored energy device **122** may store energy sufficient to power the storage device controller **119**, some amount of the RAM (e.g., RAM **121**), and some amount of Flash memory (e.g., Flash memory **120***a*-**120***n*) for sufficient time to write the contents of RAM to Flash memory. In one embodiment, storage device controller **119**A-D may write the contents of RAM to Flash Memory if the storage device controller detects loss of external power.
(40) In one embodiment, system **117** includes two data communications links **123***a*, **123***b*. In one embodiment, data communications links **123***a*, **123***b* may be PCI interfaces. In another embodiment, data communications links **123***a*, **123***b* may be based on other communications standards (e.g., HyperTransport, InfiniB and, etc.). Data communications links **123***a*, **123***b* may be based on non-volatile memory express ('NVMe') or NVMe over fabrics ('NVMf') specifications that allow external connection to the storage device controller **119**A-D from other components in the storage system **117**. It should be noted that data communications links may be interchangeably referred to herein as PCI buses for convenience.
(41) System **117** may also include an external power source (not shown), which may be provided over one or both data communications links **123***a*, **123***b*, or which may be provided separately. An alternative embodiment includes a separate Flash memory (not shown) dedicated for use in storing the content of RAM **121**. The storage device controller **119**A-D may present a logical device over a PCI bus which may include an addressable fast-write logical device, or a distinct part of the logical address space of the storage device **118**, which may be presented as PCI memory or as persistent storage. In one embodiment, operations to store into the device are directed into the RAM **121**. On power failure, the storage device controller **119**A-D may write stored content associated with the addressable fast-write logical storage to Flash memory (e.g., Flash memory **120***a-n*) for long-term persistent storage.
(42) In one embodiment, the logical device may include some presentation of some or all of the content of the Flash memory devices **120***a-n*, where that presentation allows a storage system including a storage device **118** (e.g., storage system **117**) to directly address Flash memory pages

and directly reprogram erase blocks from storage system components that are external to the storage device through the PCI bus. The presentation may also allow one or more of the external components to control and retrieve other aspects of the Flash memory including some or all of: tracking statistics related to use and reuse of Flash memory pages, erase blocks, and cells across all the Flash memory devices; tracking and predicting error codes and faults within and across the Flash memory devices; controlling voltage levels associated with programming and retrieving contents of Flash cells; etc.
(43) In one embodiment, the stored energy device **122** may be sufficient to ensure completion of in-progress operations to the Flash memory devices **120***a*-**120***n* stored energy device **122** may power storage device controller **119**A-D and associated Flash memory devices (e.g., **120***a-n*) for those operations, as well as for the storing of fast-write RAM to Flash memory. Stored energy device **122** may be used to store accumulated statistics and other parameters kept and tracked by the Flash memory devices **120***a-n* and/or the storage device controller **119**. Separate capacitors or stored energy devices (such as smaller capacitors near or embedded within the Flash memory devices themselves) may be used for some or all of the operations described herein.
(44) 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.
(45) FIG. **1**D illustrates a third example storage system **124** for data storage in accordance with some implementations. In one embodiment, storage system **124** includes storage controllers **125***a*, **125***b*. In one embodiment, storage controllers **125***a*, **125***b* are operatively coupled to Dual PCI storage devices. Storage controllers **125***a*, **125***b* may be operatively coupled (e.g., via a storage network **130**) to some number of host computers **127***a-n.*
(46) In one embodiment, two storage controllers (e.g., **125***a* and **125***b*) provide storage services, such as a SCS) block storage array, a file server, an object server, a database or data analytics service, etc. The storage controllers **125***a*, **125***b* may provide services through some number of network interfaces (e.g., **126***a-d*) to host computers **127***a-n* outside of the storage system **124**. Storage controllers **125***a*, **125***b* may provide integrated services or an application entirely within the storage system **124**, forming a converged storage and compute system. The storage controllers **125***a*, **125***b* may utilize the fast write memory within or across storage devices **119***a-d* to journal in progress operations to ensure the operations are not lost on a power failure, storage controller removal, storage controller or storage system shutdown, or some fault of one or more software or hardware components within the storage system **124**.
(47) In one embodiment, storage controllers **125***a*, **125***b* operate as PCI masters to one or the other PCI buses **128***a*, **128***b*. In another embodiment, **128***a* and **128***b* may be based on other communications standards (e.g., HyperTransport, InfiniB and, etc.). Other storage system embodiments may operate storage controllers **125***a*, **125***b* as multi-masters for both PCI buses **128***a*, **128***b*. Alternately, a PCI/NVMe/NVMf switching infrastructure or fabric may connect multiple storage controllers. Some storage system embodiments may allow storage devices to communicate with each other directly rather than communicating only with storage controllers. In one embodiment, a storage device controller **119***a* may be operable under direction from a storage controller **125***a* to synthesize and transfer data to be stored into Flash memory devices from data that has been stored in RAM (e.g., RAM **121** of FIG. **1**C). For example, a recalculated version of RAM content may be transferred after a storage controller has determined that an operation has fully committed across the storage system, or when fast-write memory on the device has reached a certain used capacity, or after a certain amount of time, to ensure improve safety of the data or to release addressable fast-write capacity for reuse. This mechanism may be used, for example, to avoid a second transfer over a bus (e.g., **128***a*, **128***b*) from the storage controllers **125***a*, **125***b*. In

one embodiment, a recalculation may include compressing data, attaching indexing or other metadata, combining multiple data segments together, performing erasure code calculations, etc.
(48) In one embodiment, under direction from a storage controller **125***a*, **125***b*, a storage device controller **119***a*, **119***b* may be operable to calculate and transfer data to other storage devices from data stored in RAM (e.g., RAM **121** of FIG. **1**C) without involvement of the storage controllers **125***a*, **125***b*. This operation may be used to mirror data stored in one storage controller **125***a* to another storage controller **125***b*, or it could be used to offload compression, data aggregation, and/or erasure coding calculations and transfers to storage devices to reduce load on storage controllers or the storage controller interface **129***a*, **129***b* to the PCI bus **128***a*, **128***b*.
(49) A storage device controller **119**A-D may include mechanisms for implementing high availability primitives for use by other parts of a storage system external to the Dual PCI storage device **118**. For example, reservation or exclusion primitives may be provided so that, in a storage system with two storage controllers providing a highly available storage service, one storage controller may prevent the other storage controller from accessing or continuing to access the storage device. This could be used, for example, in cases where one controller detects that the other controller is not functioning properly or where the interconnect between the two storage controllers may itself not be functioning properly.
(50) 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.
(51) In one embodiment, the storage controllers **125***a*, **125***b* may initiate the use of erase blocks within and across storage devices (e.g., **118**) in accordance with an age and expected remaining lifespan of the storage devices, or based on other statistics. The storage controllers **125***a*, **125***b* may initiate garbage collection and data migration data between storage devices in accordance with pages that are no longer needed as well as to manage Flash page and erase block lifespans and to manage overall system performance.
(52) 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.
(53) The embodiments depicted with reference to FIGS. **2**A-G illustrate a storage cluster that stores user data, such as user data originating from one or more user or client systems or other sources external to the storage cluster. The storage cluster distributes user data across storage nodes housed within a chassis, or across multiple chassis, using erasure coding and redundant copies of metadata. Erasure coding refers to a method of data protection or reconstruction in which data is stored across a set of different locations, such as disks, storage nodes or geographic locations. Flash memory is one type of solid-state memory that may be integrated with the embodiments, although the embodiments may be extended to other types of solid-state memory or other storage medium, including non-solid state memory. Control of storage locations and workloads are distributed across

the storage locations in a clustered peer-to-peer system. Tasks such as mediating communications between the various storage nodes, detecting when a storage node has become unavailable, and balancing I/Os (inputs and outputs) across the various storage nodes, are all handled on a distributed basis. Data is laid out or distributed across multiple storage nodes in data fragments or stripes that support data recovery in some embodiments. Ownership of data can be reassigned within a cluster, independent of input and output patterns. This architecture described in more detail below allows a storage node in the cluster to fail, with the system remaining operational, since the data can be reconstructed from other storage nodes and thus remain available for input and output operations. In various embodiments, a storage node may be referred to as a cluster node, a blade, or a server.
(54) 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, InfiniB and, 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.
(55) 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.
(56) 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.
(57) FIG. **2**A is a perspective view of a storage cluster **161**, with multiple storage nodes **150** and internal solid-state memory coupled to each storage node to provide network attached storage or storage area network, in accordance with some embodiments. A network attached storage, storage area network, or a storage cluster, or other storage memory, could include one or more storage clusters **161**, each having one or more storage nodes **150**, in a flexible and reconfigurable arrangement of both the physical components and the amount of storage memory provided thereby. The storage cluster **161** is designed to fit in a rack, and one or more racks can be set up and populated as desired for the storage memory. The storage cluster **161** has a chassis **138** having multiple slots **142**. It should be appreciated that chassis **138** may be referred to as a housing, enclosure, or rack unit. In one embodiment, the chassis **138** has fourteen slots **142**, although other numbers of slots are readily devised. For example, some embodiments have four slots, eight slots, sixteen slots, thirty-two slots, or other suitable number of slots. Each slot **142** can accommodate one storage node **150** in some embodiments. Chassis **138** includes flaps **148** that can be utilized to mount the chassis **138** on a rack. Fans **144** provide air circulation for cooling of the storage nodes **150** and components thereof, although other cooling components could be used, or an embodiment could be devised without cooling components. A switch fabric **146** couples storage nodes **150** within chassis **138** together and to a network for communication to the memory. In an embodiment depicted in herein, the slots **142** to the left of the switch fabric **146** and fans **144** are shown occupied by storage nodes **150**, while the slots **142** to the right of the switch fabric **146** and fans **144** are empty and available for insertion of storage node **150** for illustrative purposes. This configuration is one example, and one or more storage nodes **150** could occupy the slots **142** in various further arrangements. The storage node arrangements need not be sequential or adjacent in some embodiments. Storage nodes **150** are hot pluggable, meaning that a storage node **150** can be inserted into a slot **142** in the chassis **138**, or removed from a slot **142**, without stopping or powering down the system. Upon insertion or removal of storage node **150** from slot **142**, the system automatically reconfigures in order to recognize and adapt to the change. Reconfiguration, in some embodiments, includes restoring redundancy and/or rebalancing data or load.
(58) 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.
(59) Referring to FIG. **2**A, storage cluster **161** is scalable, meaning that storage capacity with non-uniform storage sizes is readily added, as described above. One or more storage nodes **150** can be plugged into or removed from each chassis and the storage cluster self-configures in some embodiments. Plug-in storage nodes **150**, whether installed in a chassis as delivered or later added, can have different sizes. For example, in one embodiment a storage node **150** can have any

multiple of 4 TB, e.g., 8 TB, 12 TB, 16 TB, 32 TB, etc. In further embodiments, a storage node **150** could have any multiple of other storage amounts or capacities. Storage capacity of each storage node **150** is broadcast, and influences decisions of how to stripe the data. For maximum storage efficiency, an embodiment can self-configure as wide as possible in the stripe, subject to a predetermined requirement of continued operation with loss of up to one, or up to two, non-volatile solid state storage **152** units or storage nodes **150** within the chassis.

(60) FIG. **2**B is a block diagram showing a communications interconnect **173** and power distribution bus **172** coupling multiple storage nodes **150**. Referring back to FIG. **2**A, the communications interconnect **173** can be included in or implemented with the switch fabric **146** in some 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. **2**B, storage cluster **161** is enclosed within a single chassis **138**. External port **176** is coupled to storage nodes **150** through communications interconnect **173**, while external port **174** is coupled directly to a storage node. External power port **178** is coupled to power distribution bus **172**. Storage nodes **150** may include varying amounts and differing capacities of non-volatile solid state storage **152** as described with reference to FIG. **2**A. In addition, one or more storage nodes **150** may be a compute only storage node as illustrated in FIG. **2**B. Authorities **168** are implemented on the non-volatile solid state 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.

(61) 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.
(62) 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.
(63) With reference to FIGS. **2**A and **2**B, two of the many tasks of the CPU **156** on a storage node **150** are to break up write data, and reassemble read data. When the system has determined that data is to be written, the authority **168** for that data is located as above. When the segment ID for data is already determined the request to write is forwarded to the non-volatile solid state storage **152** currently determined to be the host of the authority **168** determined from the segment. The host CPU **156** of the storage node **150**, on which the non-volatile solid state storage **152** and corresponding authority **168** reside, then breaks up or shards the data and transmits the data out to various non-volatile solid state storage **152**. The transmitted data is written as a data stripe in accordance with an erasure coding scheme. In some embodiments, data is requested to be pulled, and in other embodiments, data is pushed. In reverse, when data is read, the authority **168** for the segment ID containing the data is located as described above. The host CPU **156** of the storage node **150** on which the non-volatile solid state storage **152** and corresponding authority **168** reside requests the data from the non-volatile solid state storage and corresponding storage nodes pointed to by the authority. In some embodiments the data is read from flash storage as a data stripe. The host CPU **156** of storage node **150** then reassembles the read data, correcting any errors (if present) according to the appropriate erasure coding scheme, and forwards the reassembled data to the network. In further embodiments, some or all of these tasks can be handled in the non-volatile solid state storage **152**. In some embodiments, the segment host requests the data be sent to storage node **150** by requesting pages from storage and then sending the data to the storage node making the original request.
(64) 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.

(65) 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.
(66) 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.
(67) 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.
(68) A segment is a logical container of data in accordance with some embodiments. A segment is an address space between medium address space and physical flash locations, i.e., the data segment number, are in this address space. Segments may also contain meta-data, which enable data redundancy to be restored (rewritten to different flash locations or devices) without the involvement of higher level software. In one embodiment, an internal format of a segment contains client data and medium mappings to determine the position of that data. Each data segment is protected, e.g., from memory and other failures, by breaking the segment into a number of data and parity shards, where applicable. The data and parity shards are distributed, i.e., striped, across non-volatile solid state storage **152** coupled to the host CPUs **156** (See FIGS. **2**E and **2**G) in accordance with an erasure coding scheme. Usage of the term segments refers to the container and its place in the address space of segments in some embodiments. Usage of the term stripe refers to the same set of shards as a segment and includes how the shards are distributed along with redundancy or parity information in accordance with some embodiments.
(69) A series of address-space transformations takes place across an entire storage system. At the top are the directory entries (file names) which link to an inode. Modes point into medium address space, where data is logically stored. Medium addresses may be mapped through a series of indirect mediums to spread the load of large files, or implement data services like deduplication or snapshots. Medium addresses may be mapped through a series of indirect mediums to spread the load of large files, or implement data services like deduplication or snapshots. Segment addresses are then translated into physical flash locations. Physical flash locations have an address range bounded by the amount of flash in the system in accordance with some embodiments. Medium addresses and segment addresses are logical containers, and in some embodiments use a 128 bit or larger identifier so as to be practically infinite, with a likelihood of reuse calculated as longer than the expected life of the system. Addresses from logical containers are allocated in a hierarchical fashion in some embodiments. Initially, each non-volatile solid state storage **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**.
(70) Data and metadata is stored by a set of underlying storage layouts that are optimized for varying workload patterns and storage devices. These layouts incorporate multiple redundancy schemes, compression formats and index algorithms. Some of these layouts store information about authorities and authority masters, while others store file metadata and file data. The redundancy schemes include error correction codes that tolerate corrupted bits within a single storage device (such as a NAND flash chip), erasure codes that tolerate the failure of multiple storage nodes, and replication schemes that tolerate data center or regional failures. In some embodiments, low density parity check ('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 a log structured layout.
(71) In order to maintain consistency across multiple copies of an entity, the storage nodes agree implicitly on two things through calculations: (1) the authority that contains the entity, and (2) the storage node that contains the authority. The assignment of entities to authorities can be done by pseudo randomly assigning entities to authorities, by splitting entities into ranges based upon an externally produced key, or by placing a single entity into each authority. Examples of pseudorandom schemes are linear hashing and the Replication Under Scalable Hashing ('RUSH') family of hashes, including Controlled Replication Under Scalable Hashing ('CRUSH'). In some embodiments, pseudo-random assignment is utilized only for assigning authorities to nodes because the set of nodes can change. The set of authorities cannot change so any subjective function may be applied in these embodiments. Some placement schemes automatically place authorities on storage nodes, while other placement schemes rely on an explicit mapping of authorities to storage nodes. In some embodiments, a pseudorandom scheme is utilized to map from each authority to a set of candidate authority owners. A pseudorandom data distribution function related to CRUSH may assign authorities to storage nodes and create a list of where the authorities are assigned. Each storage node has a copy of the pseudorandom data distribution function, and can arrive at the same calculation for distributing, and later finding or locating an authority. Each of the pseudorandom schemes requires the reachable set of storage nodes as input in some embodiments in order to conclude the same target nodes. Once an entity has been placed in an authority, the entity may be stored on physical devices so that no expected failure will lead to unexpected data loss. In some embodiments, rebalancing algorithms attempt to store the copies of all entities within an authority in the same layout and on the same set of machines.
(72) 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.
(73) 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.
(74) 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.
(75) 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.
(76) 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.
(77) 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.
(78) FIG. **2**C is a multiple level block diagram, showing contents of a storage node **150** and contents of a non-volatile solid state storage **152** of the storage node **150**. Data is communicated to and from the storage node **150** by a network interface controller (‘NIC’) **202** in some 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. **2**C, each non-volatile solid state storage **152** has a relatively fast non-volatile solid state memory, such as nonvolatile random access memory (‘NVRAM’) **204**, and flash memory **206**. In some embodiments, NVRAM **204** may be a component that does not require program/erase cycles (DRAM, MRAM, PCM), and can be a memory that can support being written vastly more often than the memory is read from. Moving

down another level in FIG. 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**.

(79) 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 sold 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).

(80) FIG. **2**D shows a storage server environment, which uses embodiments of the storage nodes **150** and storage **152** units of FIGS. **2**A-C. In this version, each non-volatile solid state storage **152** unit has a processor such as controller **212** (see FIG. **2**C), an FPGA, flash memory **206**, and NVRAM **204** (which is super-capacitor backed DRAM **216**, see FIGS. **2**B and **2**C) on a PCIe (peripheral component interconnect express) board in a chassis **138** (see FIG. **2**A). The 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.

(81) 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**. That authority **168** further manages lifetimes and allocations within that space. Examples of a spool include distributed transactions or notions. When the primary power to a non-volatile solid state storage **152** unit fails, onboard super-capacitors provide a short duration of power hold up. During this holdup interval, the contents of the NVRAM **204** are flushed to flash memory **206**. On the next power-on, the contents of the NVRAM **204** are recovered from the flash memory **206**.
(82) As for the storage unit controller, the responsibility of the logical “controller” is distributed across each of the blades containing authorities **168**. This distribution of logical control is shown in FIG. **2**D as a host controller **242**, mid-tier controller **244** and storage unit controller(s) **246**. Management of the control plane and the storage plane are treated independently, although parts may be physically co-located on the same blade. Each authority **168** effectively serves as an independent controller. Each authority **168** provides its own data and metadata structures, its own background workers, and maintains its own lifecycle.
(83) FIG. **2**E is a blade **252** hardware block diagram, showing a control plane **254**, compute and storage planes **256**, **258**, and authorities **168** interacting with underlying physical resources, using embodiments of the storage nodes **150** and storage units **152** of FIGS. **2**A-C in the storage server environment of FIG. **2**D. The control plane **254** is partitioned into a number of authorities **168** which can use the compute resources in the compute plane **256** to run on any of the blades **252**. The storage plane **258** is partitioned into a set of devices, each of which provides access to flash **206** and NVRAM **204** resources. In one embodiment, the compute plane **256** may perform the operations of a storage array controller, as described herein, on one or more devices of the storage plane **258** (e.g., a storage array).
(84) In the compute and storage planes **256**, **258** of FIG. **2**E, the authorities **168** interact with the underlying physical resources (i.e., devices). From the point of view of an authority **168**, its resources are striped over all of the physical devices. From the point of view of a device, it provides resources to all authorities **168**, irrespective of where the authorities happen to run. Each authority **168** has allocated or has been allocated one or more partitions **260** of storage memory in the storage units **152**, e.g., partitions **260** in flash memory **206** and NVRAM **204**. Each authority **168** uses those allocated partitions **260** that belong to it, for writing or reading user data. Authorities can be associated with differing amounts of physical storage of the system. For example, one authority **168** could have a larger number of partitions **260** or larger sized partitions **260** in one or more storage units **152** than one or more other authorities **168**.
(85) FIG. **2**F depicts elasticity software layers in blades **252** of a storage cluster, in accordance with some embodiments. In the elasticity structure, elasticity software is symmetric, i.e., each blade's compute module **270** runs the three identical layers of processes depicted in FIG. **2**F. Storage managers **274** execute read and write requests from other blades **252** for data and metadata stored in local storage unit **152** NVRAM **204** and flash **206**. Authorities **168** fulfill client requests by issuing the necessary reads and writes to the blades **252** on whose storage units **152** the corresponding data or metadata resides. Endpoints **272** parse client connection requests received from switch fabric **146** supervisory software, relay the client connection requests to the authorities **168** responsible for fulfillment, and relay the authorities' **168** responses to clients. The symmetric three-layer structure enables the storage system's high degree of concurrency. Elasticity scales out efficiently and reliably in these embodiments. In addition, elasticity implements a unique scale-out technique that balances work evenly across all resources regardless of client access pattern, and maximizes concurrency by eliminating much of the need for inter-blade coordination that typically

occurs with conventional distributed locking.
(86) Still referring to FIG. **2**F, authorities **168** running in the compute modules **270** of a blade **252** perform the internal operations required to fulfill client requests. One feature of elasticity is that authorities **168** are stateless, i.e., they cache active data and metadata in their own blades' **252** DRAMs for fast access, but the authorities store every update in their NVRAM **204** partitions on three separate blades **252** until the update has been written to flash **206**. All the storage system writes to NVRAM **204** are in triplicate to partitions on three separate blades **252** in some embodiments. With triple-mirrored NVRAM **204** and persistent storage protected by parity and Reed-Solomon RAID checksums, the storage system can survive concurrent failure of two blades **252** with no loss of data, metadata, or access to either.
(87) 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.
(88) 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.
(89) FIG. **2**G depicts authorities **168** and storage resources in blades **252** of a storage cluster, in accordance with some embodiments. Each authority **168** is exclusively responsible for a partition of the flash **206** and NVRAM **204** on each blade **252**. The authority **168** manages the content and integrity of its partitions independently of other authorities **168**. Authorities **168** compress incoming data and preserve it temporarily in their NVRAM **204** partitions, and then consolidate, RAID-protect, and persist the data in segments of the storage in their flash **206** partitions. As the authorities **168** write data to flash **206**, storage managers **274** perform the necessary flash translation to optimize write performance and maximize media longevity. In the background, authorities **168** “garbage collect,” or reclaim space occupied by data that clients have made obsolete by overwriting the data. It should be appreciated that since authorities' **168** partitions are disjoint, there is no need for distributed locking to execute client and writes or to perform background functions.
(90) The embodiments described herein may utilize various software, communication and/or networking protocols. In addition, the configuration of the hardware and/or software may be adjusted to accommodate various protocols. For example, the embodiments may utilize Active Directory, which is a database based system that provides authentication, directory, policy, and other services in a WINDOWS' environment. In these embodiments, LDAP (Lightweight Directory Access Protocol) is one example application protocol for querying and modifying items in directory service providers such as Active Directory. In some embodiments, a network lock manager (‘NLM’) is utilized as a facility that works in cooperation with the Network File System (‘NFS’) to provide a System V style of advisory file and record locking over a network. The Server Message Block (‘SMB’) protocol, one version of which is also known as Common Internet File System (‘CIFS’), may be integrated with the storage systems discussed herein. 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, may include utilization of an access control list ('ACL'). The ACL is a list of permissions attached to an object and the ACL specifies which users or system processes are granted access to objects, as well as what operations are allowed on given objects. The systems may utilize Internet Protocol version 6 ('IPv6'), as well as IPv4, for the communications protocol that provides an identification and location system for computers on networks and routes traffic across the Internet. The routing of packets between networked systems may include Equal-cost multi-path routing ('ECMP'), which is a routing strategy where next-hop packet forwarding to a single destination can occur over multiple "best paths" which tie for top place in routing metric calculations. Multi-path routing can be used in conjunction with most routing protocols, because it is a per-hop decision limited to a single router. The software may support Multi-tenancy, which is an architecture in which a single instance of a software application serves multiple customers. Each customer may be referred to as a tenant. Tenants may be given the ability to customize some parts of the application, but may not customize the application's code, in some embodiments. The embodiments may maintain audit logs. An audit log is a document that records an event in a computing system. In addition to documenting what resources were accessed, audit log entries typically include destination and source addresses, a timestamp, and user login information for compliance with various regulations. The embodiments may support various key management policies, such as encryption key rotation. In addition, the system may support dynamic root passwords or some variation dynamically changing passwords.
(91) FIG. **3**A sets forth a diagram of a storage system **306** that is coupled for data communications with a cloud services provider **302** in accordance with some embodiments of the present disclosure. Although depicted in less detail, the storage system **306** depicted in FIG. **3**A may be similar to the storage systems described above with reference to FIGS. **1**A-**1**D and FIGS. **2**A-**2**G. In some embodiments, the storage system **306** depicted in FIG. **3**A may be embodied as a storage system that includes imbalanced active/active controllers, as a storage system that includes balanced active/active controllers, as a storage system that includes active/active controllers where less than all of each controller's resources are utilized such that each controller has reserve resources that may be used to support failover, as a storage system that includes fully active/active controllers, as a storage system that includes dataset-segregated controllers, as a storage system that includes dual-layer architectures with front-end controllers and back-end integrated storage controllers, as a storage system that includes scale-out clusters of dual-controller arrays, as well as combinations of such embodiments.
(92) In the example depicted in FIG. **3**A, the storage system **306** is coupled to the cloud services provider **302** via a data communications link **304**. 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 ('IF), real-time transfer protocol ('RTP'), transmission control protocol ('TCP'), user datagram protocol ('UDP'), wireless application protocol ('WAP'), or other protocol.
(93) The cloud services provider **302** depicted in FIG. **3**A may be embodied, for example, as a system and computing environment that provides a vast array of services to users of the cloud services provider **302** through the sharing of computing resources via the data communications link

304. The cloud services provider **302** may provide on-demand access to a shared pool of configurable computing resources such as computer networks, servers, storage, applications and services, and so on.
(94) In the example depicted in FIG. **3**A, the cloud services provider **302** may be configured to provide a variety of services to the storage system **306** and users of the storage system **306** through the implementation of various service models. For example, the cloud services provider **302** may be configured to provide services through the implementation of an infrastructure as a service ('IaaS') service model, through the implementation of a platform as a service ('PaaS') service model, through the implementation of a software as a service ('SaaS') service model, through the implementation of an authentication as a service ('AaaS') service model, through the implementation of a storage as a service model where the cloud services provider **302** offers access to its storage infrastructure for use by the storage system **306** and users of the storage system **306**, and so on.
(95) In the example depicted in FIG. **3**A, the cloud services provider **302** may be embodied, for example, as a private cloud, as a public cloud, or as a combination of a private cloud and public cloud. In an embodiment in which the cloud services provider **302** is embodied as a private cloud, the cloud services provider **302** may be dedicated to providing services to a single organization rather than providing services to multiple organizations. In an embodiment where the cloud services provider **302** is embodied as a public cloud, the cloud services provider **302** may provide services to multiple organizations. In still alternative embodiments, the cloud services provider **302** may be embodied as a mix of a private and public cloud services with a hybrid cloud deployment.
(96) Although not explicitly depicted in FIG. **3**A, readers will appreciate that a vast amount of additional hardware components and additional software components may be necessary to facilitate the delivery of cloud services to the storage system **306** and users of the storage system **306**. For example, the storage system **306** may be coupled to (or even include) a cloud storage gateway. Such a cloud storage gateway may be embodied, for example, as hardware-based or software-based appliance that is located on premises 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**.
(97) 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.
(98) In the example depicted in FIG. **3**A, and as described briefly above, the cloud services provider **302** may be configured to provide services to the storage system **306** and users of the

storage system **306** through the usage of a SaaS service model. 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**.
(99) 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.
(100) Although the example depicted in FIG. **3**A illustrates the storage system **306** being coupled for data communications with the cloud services provider **302**, in other embodiments the storage system **306** may be part of a hybrid cloud deployment in which private cloud elements (e.g., private cloud services, on-premises infrastructure, and so on) and public cloud elements (e.g., public cloud services, infrastructure, and so on that may be provided by one or more cloud services providers) are combined to form a single solution, with orchestration among the various platforms. Such a hybrid cloud deployment may leverage hybrid cloud management software such as, for example, Azure™ Arc from Microsoft™, that centralize the management of the hybrid cloud deployment to any infrastructure and enable the deployment of services anywhere. In such an example, the hybrid cloud management software may be configured to create, update, and delete resources (both physical and virtual) that form the hybrid cloud deployment, to allocate compute and storage to specific workloads, to monitor workloads and resources for performance, policy compliance, updates and patches, security status, or to perform a variety of other tasks.
(101) 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.
(102) 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.
(103) For further explanation, FIG. **3**B sets forth a diagram of a storage system **306** in accordance with some embodiments of the present disclosure. Although depicted in less detail, the storage

system **306** depicted in FIG. **3**B may be similar to the storage systems described above with reference to FIGS. **1**A-**1**D and FIGS. **2**A-**2**G as the storage system may include many of the components described above.
(104) The storage system **306** depicted in FIG. **3**B may include a vast amount of storage resources **308**, which may be embodied in many forms. For example, the storage resources **308** can include nano-RAM or another form of nonvolatile random access memory that utilizes carbon nanotubes deposited on a substrate, 3D crosspoint non-volatile memory, flash memory including single-level cell ('SLC') NAND flash, multi-level cell ('MLC') NAND flash, triple-level cell ('TLC') NAND flash, quad-level cell ('QLC') NAND flash, or others. Likewise, the storage resources **308** may include non-volatile magnetoresistive random-access memory ('MRAM'), including spin transfer torque ('STT') MRAM. The example storage resources **308** may alternatively include non-volatile phase-change memory ('PCM'), quantum memory that allows for the storage and retrieval of photonic quantum information, resistive random-access memory ('ReRAM'), storage class memory ('SCM'), or other form of storage resources, including any combination of resources described herein. Readers will appreciate that other forms of computer memories and storage devices may be utilized by the storage systems described above, including DRAM, SRAM, EEPROM, universal memory, and many others. The storage resources **308** depicted in FIG. **3**A may be embodied in a variety of form factors, including but not limited to, dual in-line memory modules ('DIMMs'), non-volatile dual in-line memory modules ('NVDIMMs'), M.2, U.2, and others.
(105) The storage resources **308** depicted in FIG. **3**B may include various forms of SCM. SCM may effectively treat fast, non-volatile memory (e.g., NAND flash) as an extension of DRAM such that an entire dataset may be treated as an in-memory dataset that resides entirely in DRAM. SCM may include non-volatile media such as, for example, NAND flash. Such NAND flash may be accessed utilizing NVMe that can use the PCIe bus as its transport, providing for relatively low access latencies compared to older protocols. In fact, the network protocols used for SSDs in all-flash arrays can include NVMe using Ethernet (ROCE, NVME TCP), Fibre Channel (NVMe FC), InfiniBand (iWARP), and others that make it possible to treat fast, non-volatile memory as an extension of DRAM. In view of the fact that DRAM is often byte-addressable and fast, non-volatile memory such as NAND flash is block-addressable, a controller software/hardware stack may be needed to convert the block data to the bytes that are stored in the media. Examples of media and software that may be used as SCM can include, for example, 3D XPoint, Intel Memory Drive Technology, Samsung's Z-SSD, and others.
(106) The storage resources **308** depicted in FIG. **3**B may also include racetrack memory (also referred to as domain-wall memory). Such racetrack memory may be embodied as a form of non-volatile, solid-state memory that relies on the intrinsic strength and orientation of the magnetic field created by an electron as it spins in addition to its electronic charge, in solid-state devices. Through the use of spin-coherent electric current to move magnetic domains along a nanoscopic permalloy wire, the domains may pass by magnetic read/write heads positioned near the wire as current is passed through the wire, which alter the domains to record patterns of bits. In order to create a racetrack memory device, many such wires and read/write elements may be packaged together.
(107) The example storage system **306** depicted in FIG. **3**B may implement a variety of storage architectures. For example, storage systems in accordance with some embodiments of the present disclosure may utilize block storage where data is stored in blocks, and each block essentially acts as an individual hard drive. Storage systems in accordance with some embodiments of the present disclosure may utilize object storage, where data is managed as objects. Each object may include the data itself, a variable amount of metadata, and a globally unique identifier, where object storage can be implemented at multiple levels (e.g., device level, system level, interface level). Storage systems in accordance with some embodiments of the present disclosure utilize file storage in which data is stored in a hierarchical structure. Such data may be saved in files and folders, and

presented to both the system storing it and the system retrieving it in the same format.
(108) The example storage system **306** depicted in FIG. **3**B may be embodied as a storage system in which additional storage resources can be added through the use of a scale-up model, additional storage resources can be added through the use of a scale-out model, or through some combination thereof. In a scale-up model, additional storage may be added by adding additional storage devices. In a scale-out model, however, additional storage nodes may be added to a cluster of storage nodes, where such storage nodes can include additional processing resources, additional networking resources, and so on.
(109) The example storage system **306** depicted in FIG. **3**B may leverage the storage resources described above in a variety of different ways. For example, some portion of the storage resources may be utilized to serve as a write cache, 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.
(110) The storage system **306** depicted in FIG. **3**B also includes communications resources **310** that may be useful in facilitating data communications between components within the storage system **306**, as well as data communications between the storage system **306** and computing devices that are outside of the storage system **306**, including embodiments where those resources are separated by a relatively vast expanse. The communications resources **310** may be configured to utilize a variety of different protocols and data communication fabrics to facilitate data communications between components within the storage systems as well as computing devices that are outside of the storage system. For example, the communications resources **310** can include fibre channel ('FC') technologies such as FC fabrics and FC protocols that can transport SCSI commands over FC network, FC over ethernet ('FCoE') technologies through which FC frames are encapsulated and transmitted over Ethernet networks, InfiniB and ('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.
(111) The communications resources **310** can also include mechanisms for accessing storage resources **308** within the storage system **306** utilizing serial attached SCSI ('SAS'), serial ATA ('SATA') bus interfaces for connecting storage resources **308** within the storage system **306** to host bus adapters within the storage system **306**, internet small computer systems interface ('iSCSI') technologies to provide block-level access to storage resources **308** within the storage system **306**, and other communications resources that that may be useful in facilitating data communications between components within the storage system **306**, as well as data communications between the storage system **306** and computing devices that are outside of the storage system **306**.
(112) The storage system **306** depicted in FIG. **3**B also includes processing resources **312** that may be useful in useful in executing computer program instructions and performing other computational tasks within the storage system **306**. The processing resources **312** may include one or more ASICs that are customized for some particular purpose as well as one or more CPUs. The processing resources **312** may also include one or more DSPs, one or more FPGAs, one or more systems on a chip ('SoCs'), or other form of processing resources **312**. The storage system **306** may utilize the storage resources **312** to perform a variety of tasks including, but not limited to, supporting the execution of software resources **314** that will be described in greater detail below.
(113) The storage system **306** depicted in FIG. **3**B also includes software resources **314** that, when executed by processing resources **312** within the storage system **306**, may perform a vast array of tasks. The software resources **314** may include, for example, one or more modules of computer program instructions that when executed by processing resources **312** within the storage system

306 are useful in carrying out various data protection techniques. 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.

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

(115) The software resources **314** may also include software 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 that perform other functions. Such software resources **314** may be embodied as one or more software containers or in many other ways.

(116) For further explanation, FIG. **3**C 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. **3**C, 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.

(117) The cloud-based storage system **318** depicted in FIG. **3**C 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**.

(118) In the example method depicted in FIG. **3**C, the storage controller application **324**, **326** may be embodied as a module of computer program instructions that, when executed, carries out various storage tasks. For example, the storage controller application **324**, **326** may be embodied as a module of computer program instructions that, when executed, carries out the same tasks as the controllers **110**A, **110**B in FIG. **1**A described above such as writing data 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. **3**C may include identical source code that is executed within different cloud computing instances **320**, **322** such as distinct EC2 instances.
(119) 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.
(120) The cloud-based storage system **318** depicted in FIG. **3**C includes cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338**. The cloud computing instances **340***a*, **340***b*, **340***n* may be embodied, for example, as instances of cloud computing resources that may be provided by the cloud computing environment **316** to support the execution of software applications. The cloud computing instances **340***a*, **340***b*, **340***n* of FIG. **3**C may differ from the cloud computing instances **320**, **322** described above as the cloud computing instances **340***a*, **340***b*, **340***n* of FIG. **3**C have local storage **330**, **334**, **338** resources whereas the cloud computing instances **320**, **322** that support the execution of the storage controller application **324**, **326** need not have local storage resources. The cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338** may be embodied, for example, as EC2 M5 instances that include one or more SSDs, as EC2 R5 instances that include one or more SSDs, as EC2 I3 instances that include one or more SSDs, and so on. In some embodiments, the local storage **330**, **334**, **338** must be embodied as solid-state storage (e.g., SSDs) rather than storage that makes use of hard disk drives.
(121) In the example depicted in FIG. **3**C, each of the cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338** can include a software daemon **328**, **332**, **336** that, when executed by a cloud computing instance **340***a*, **340***b*, **340***n* can present itself to the storage controller applications **324**, **326** as if the cloud computing instance **340***a*, **340***b*, **340***n* were a physical storage device (e.g., one or more SSDs). In such an example, the software daemon **328**, **332**, **336** may include computer program instructions similar to those that would normally be contained on a storage device such that the storage controller applications **324**, **326** can send and receive the same commands that a storage controller would send to storage devices. In such a way, the storage controller applications **324**, **326** may include code that is identical to (or substantially identical to) the code that would be executed by the controllers in the storage systems described above. In these and similar embodiments, communications between the storage controller applications **324**, **326** and the cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338** may utilize iSCSI, NVMe over TCP, messaging, a custom protocol, or in some other mechanism.
(122) In the example depicted in FIG. **3**C, each of the cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338** may also be coupled to block storage **342**, **344**, **346** that is offered by the cloud computing environment **316** 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 comping instance **340***a*, **340***b*, **340***n* may, upon receiving a request to write data, initiate a write of the data to its attached EBS volume as well as a write of the data to its local storage **330**, **334**, **338** resources. In some alternative embodiments, data may only be written to the local storage **330**, **334**, **338** resources within a particular cloud comping instance **340***a*, **340***b*, **340***n*. In an alternative embodiment, rather than using the block storage **342**, **344**, **346** that is offered by the cloud computing environment **316** as NVRAM, actual RAM on each

of the cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338** may be used as NVRAM, thereby decreasing network utilization costs that would be associated with using an EBS volume as the NVRAM. In yet another embodiment, high performance block storage resources such as one or more Azure Ultra Disks may be utilized as the NVRAM.

(123) When a request to write data is received by a particular cloud computing instance **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338**, the software daemon **328**, **332**, **336** 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 **340***a*, **340***b*, **340***n*. The cloud-based object storage **348** that is attached to the particular cloud computing instance **340***a*, **340***b*, **340***n* may be embodied, for example, as Amazon Simple Storage Service (‘S3’). In other embodiments, the cloud computing instances **320**, **322** that each include the storage controller application **324**, **326** may initiate the storage of the data in the local storage **330**, **334**, **338** of the cloud computing instances **340***a*, **340***b*, **340***n* and the cloud-based object storage **348**. In other embodiments, rather than using both the cloud computing instances **340***a*, **340***b*, **340***n* 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.

(124) While the local storage **330**, **334**, **338** resources and the block storage **342**, **344**, **346** resources that are utilized by the cloud computing instances **340***a*, **340***b*, **340***n* may support block-level access, the cloud-based object storage **348** that is attached to the particular cloud computing instance **340***a*, **340***b*, **340***n* supports only object-based access. 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 **340***a*, **340***b*, **340***n*.

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

(126) One or more modules of computer program instructions that are executing within the cloud-based storage system **318** (e.g., a monitoring module that is executing on its own EC2 instance) may be designed to handle the failure of one or more of the cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338**. In such an example, the monitoring module may handle the

failure of one or more of the cloud computing instances **340***a*, **340***b*, **340***n* with local storage **330**, **334**, **338** by creating one or more new cloud computing instances with local storage, retrieving data that was stored on the failed cloud computing instances **340***a*, **340***b*, **340***n* from the cloud-based object storage **348**, and storing the data retrieved from the cloud-based object storage **348** in local storage on the newly created cloud computing instances. Readers will appreciate that many variants of this process may be implemented.
(127) 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.
(128) 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.
(129) Readers will further appreciate that the backups (often in the form of one or more snapshots) may also be utilized to perform rapid recovery of the storage system. Consider an example in which the storage system is infected with ransomware that locks users out of the storage system. In such an example, software resources **314** within the storage system may be configured to detect the presence of ransomware and may be further configured to restore the storage system to a point-in-time, using the retained backups, prior to the point-in-time at which the ransomware infected the storage system. In such an example, the presence of ransomware may be explicitly detected through the use of software tools utilized by the system, through the use of a key (e.g., a USB drive) that is inserted into the storage system, or in a similar way. Likewise, the presence of ransomware may be inferred in response to system activity meeting a predetermined fingerprint such as, for example, no reads or writes coming into the system for a predetermined period of time.
(130) 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.

(131) 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.

(132) As an example of one type of application that may be supported by the storage systems describe herein, the storage system **306** may be useful in supporting artificial intelligence ('AI') applications, database applications, XOps projects (e.g., DevOps projects, DataOps projects, MLOp s 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.

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

(134) Such AI applications may enable devices to perceive their environment and take actions that maximize their chance of success at some goal. Examples of such AI applications can include IBM Watson″, Microsoft Oxford™, Google DeepMind™, Baidu Minwa™, and others.

(135) The storage systems described above may also be well suited to support other types of applications that are resource intensive such as, for example, machine learning applications. Machine learning applications may perform various types of data analysis to automate analytical model building. Using algorithms that iteratively learn from data, machine learning applications can enable computers to learn without being explicitly programmed. One particular area of machine learning is referred to as reinforcement learning, which involves taking suitable actions to maximize reward in a particular situation.

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

ASICs), networking components, software components, and others. In addition to GPUs, the storage systems described above may also include neural network processors ('NNPs') for use in various aspects of neural network processing. Such NNPs may be used in place of (or in addition to) GPUs and may also be independently scalable.
(137) As described above, the storage systems described herein may be configured to support artificial intelligence applications, machine learning applications, big data analytics applications, and many other types of applications. The rapid growth in these sort of applications is being driven by three technologies: deep learning (DL), GPU processors, and Big Data. Deep learning is a computing model that makes use of massively parallel neural networks inspired by the human brain. Instead of experts handcrafting software, a deep learning model writes its own software by learning from lots of examples. Such GPUs may include thousands of cores that are well-suited to run algorithms that loosely represent the parallel nature of the human brain.
(138) 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.
(139) 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.
(140) Readers will appreciate that each stage in the AI data pipeline may have varying requirements from the data hub (e.g., the storage system or collection of storage systems). Scale-out storage systems must deliver uncompromising performance for all manner of access types and patterns—from small, metadata-heavy to large files, from random to sequential access patterns, and from low to high concurrency. The storage systems described above may serve as an ideal AI data

hub as the systems may service unstructured workloads. In the first stage, data is ideally ingested and stored on to the same data hub that following stages will use, in order to avoid excess data copying. The next two steps can be done on a standard compute server that optionally includes a GPU, and then in the fourth and last stage, full training production jobs are run on powerful GPU-accelerated servers. Often, there is a production pipeline alongside an experimental pipeline operating on the same dataset. Further, the GPU-accelerated servers can be used independently for different models or joined together to train on one larger model, even spanning multiple systems for distributed training. If the shared storage tier is slow, then data must be copied to local storage for each phase, resulting in wasted time staging data onto different servers. The ideal data hub for the AI training pipeline delivers performance similar to data stored locally on the server node while also having the simplicity and performance to enable all pipeline stages to operate concurrently.
(141) 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.
(142) 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.
(143) 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.
(144) 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 storage systems described herein may be leveraged to support on-chain storage of data as well as off-chain storage of data.
(145) 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).
(146) 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.
(147) 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.
(148) 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 computers in the form of servers and workstations which specialize in using General-purpose computing on graphics processing units ('GPGPU') to accelerate deep learning applications that are interconnected into a computation engine to train parameters for deep neural networks. Each system may have Ethernet external connectivity, InfiniB and 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 computers 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 possible.
(149) Readers will appreciate that the storage systems described above, either alone or in coordination with other computing machinery may be configured to support other AI related tools. For example, the storage systems may make use of tools like ONXX or other open neural network exchange formats that make it easier to transfer models written in different AI frameworks. Likewise, the storage systems may be configured to support tools like Amazon's Gluon that allow developers to prototype, build, and train deep learning models. In fact, the storage systems described above may be part of a larger platform, such as IBM™ Cloud Private for Data, which

includes integrated data science, data engineering and application building services.
(150) Readers will further appreciate that the storage systems described above may also be deployed as an edge solution. Such an edge solution may be in place to optimize cloud computing systems by performing data processing at the edge of the network, near the source of the data. Edge computing can push applications, data and computing power (i.e., services) away from centralized points to the logical extremes of a network. Through the use of edge solutions such as the storage systems described above, computational tasks may be performed using the compute resources provided by such storage systems, data may be storage using the storage resources of the storage system, and cloud-based services may be accessed through the use of various resources of the storage system (including networking resources). By performing computational tasks on the edge solution, storing data on the edge solution, and generally making use of the edge solution, the consumption of expensive cloud-based resources may be avoided, and, in fact, performance improvements may be experienced relative to a heavier reliance on cloud-based resources.
(151) 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.
(152) The storage systems described above may alone, or in combination with other computing resources, serves as a network edge platform that combines compute resources, storage resources, networking resources, cloud technologies and network virtualization technologies, and so on. As part of the network, the edge may take on characteristics similar to other network facilities, from the customer premise and backhaul aggregation facilities to Points of Presence (PoPs) and regional data centers. Readers will appreciate that network workloads, such as Virtual Network Functions (VNFs) and others, will reside on the network edge platform. Enabled by a combination of containers and virtual machines, the network edge platform may rely on controllers and schedulers that are no longer geographically co-located with the data processing resources. The functions, as microservices, may split into control planes, user and data planes, or even state machines, allowing for independent optimization and scaling techniques to be applied. Such user and data planes may be enabled through increased accelerators, both those residing in server platforms, such as FPGAs and Smart NICs, and through SDN-enabled merchant silicon and programmable ASICs.
(153) 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.
(154) 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 of 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.
(155) 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.
(156) 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.
(157) 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.
(158) 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.
(159) 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.
(160) 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.
(161) 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.
(162) 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{circumflex over ( )}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.
(163) 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.
(164) 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.
(165) 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.
(166) 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.
(167) 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.
(168) 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.
(169) 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.
(170) 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.
(171) The storage systems described herein may also be configured implement a recovery point objective ('RPO'), which may be establish 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.
(172) 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.
(173) 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.
(174) 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.
(175) 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.
(176) 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.
(177) 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.
(178) For further explanation, FIG. **3**D 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. **3**D, 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. **3**D, the components illustrated in FIG. **3**D are not intended to be limiting. Additional or alternative components may be used in other embodiments. Components of computing device **350** shown in FIG. **3**D will now be described in additional detail.
(179) 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.
(180) 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**.
(181) 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**.
(182) 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.
(183) 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**.
(184) For further explanation, FIG. **3**E 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**E includes a plurality of storage systems **374***a*, **374***b*, **374***c*, through **374***n* that may each be similar to the storage systems described herein. The storage systems **374***a*, **374***b*, **374***c*, through **374***n* 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 **374***a*, **374***n* depicted in FIG. **3**E are depicted as being cloud-based storage systems, as the resources that collectively form each of the storage systems **374***a*, **374***n* 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**.
(185) The example depicted in FIG. **3**E 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.
(186) The edge management service **366** depicted in FIG. **3**E 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.
(187) 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 **374***a*, **374***b*, **374***c*, through **374***n*. For example, the edge management service **366** may be configured to provide storage services to host devices **378***a*, **378***b*, **378***c*, **378***d*, **378***n* 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 **378***a*, **378***b*, **378***c*, **378***d*, **378***n* and the storage systems **374***a*, **374***b*, **374***c*, through **374***n*, rather than requiring that the host devices **378***a*, **378***b*, **378***c*, **378***d*, **378***n* directly access the storage systems **374***a*, **374***b*, **374***c*, through **374***n*.
(188) The edge management service **366** of FIG. **3**E exposes a storage services module **364** to the host devices **378***a*, **378***b*, **378***c*, **378***d*, **378***n* of FIG. **3**E, 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. **3**E 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.
(189) The edge management service **366** of FIG. **3**E also includes a system management services module **368**. The system management services module **368** of FIG. **3**E includes one or more modules of computer program instructions that, when executed, perform various operations in coordination with the storage systems **374***a*, **374***b*, **374***c*, through **374***n* to provide storage services to the host devices **378***a*, **378***b*, **378***c*, **378***d*, **378***n*. The system management services module **368** may be configured, for example, to perform tasks such as provisioning storage resources from the storage systems **374***a*, **374***b*, **374***c*, through **374***n* via one or more APIs exposed by the storage systems **374***a*, **374***b*, **374***c*, through **374***n*, migrating datasets or workloads amongst the storage systems **374***a*, **374***b*, **374***c*, through **374***n* via one or more APIs exposed by the storage systems **374***a*, **374***b*, **374***c*, through **374***n*, setting one or more tunable parameters (i.e., one or more configurable settings) on the storage systems **374***a*, **374***b*, **374***c*, through **374***n* via one or more APIs exposed by the storage systems **374***a*, **374***b*, **374***c*, through **374***n*, and so on. For example, many of the services described below relate to embodiments where the storage systems **374***a*, **374***b*, **374***c*, through **374***n* 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 **374***a*, **374***b*, **374***c*, through **374***n* to configure the storage systems **374***a*, **374***b*, **374***c*, through **374***n* to operate in the ways described below.
(190) In addition to configuring the storage systems **374***a*, **374***b*, **374***c*, through **374***n*, 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 (TIP) contained in a dataset to be obfuscated when the dataset is accessed. In such an example, the storage systems **374***a*, **374***b*, **374***c*, through **374***n* may be configured to obfuscate PII when servicing read requests directed to the dataset. Alternatively, the storage systems **374***a*, **374***b*, **374***c*, through **374***n* 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 **374***a*, **374***b*, **374***c*, through **374***n* to the host devices **378***a*, **378***b*, **378***c*, **378***d*, **378***n*.
(191) The storage systems **374***a*, **374***b*, **374***c*, through **374***n* depicted in FIG. **3**E may be embodied as one or more of the storage systems described above with reference to FIGS. **1**A-**3**D, including variations thereof. In fact, the storage systems **374***a*, **374***b*, **374***c*, through **374***n* 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 **374***a* may be a cloud-based storage system, another storage system **374***b* may be a storage system that provides block storage, another storage system **374***c* may be a storage system that provides file storage, another storage system **374***d* may be a relatively high-performance storage system while another storage system **374***n* may be a relatively low-performance storage system, and so on. In alternative embodiments, only a single storage system may be present.
(192) The storage systems **374***a*, **374***b*, **374***c*, through **374***n* depicted in FIG. **3**E may also be organized into different failure domains so that the failure of one storage system **374***a* should be totally unrelated to the failure of another storage system **374***b*. 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**.
(193) 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.
(194) 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 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.
(195) 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.
(196) The storage systems **374***a*, **374***b*, **374***c*, through **374***n* 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. **3**E. 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 loading balancing or other performance purposes, and many other tasks. As such, and for many other reasons, the storage systems **374***a*, **374***b*, **374***c*, through **374***n* may be coupled to each other via one or more data communications links in order to exchange data between the storage systems **374***a*, **374***b*, **374***c*, through **374***n*.
(197) 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.
(198) 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.
(199) 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.
(200) 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.
(201) FIG. **3**F 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.
(202) 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.
(203) 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**.
(204) 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**.
(205) 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.
(206) 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.
(207) 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.
(208) 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**.
(209) 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.
(210) 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**.
(211) 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.
(212) 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 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.
(213) 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.
(214) FIG. **4** is a block diagram illustrating an example storage system **400** in accordance with some embodiments of the present disclosure. The storage system **400** may be used to store data and/or provide client/computing devices with access to data, as discussed above. As illustrated in FIG. **4**, the storage system **400** includes storage clusters **407**A, **407**B, and **407**C. A storage cluster may be a group of storage nodes (e.g., one or more chassis, one or more racks, etc.). The storage clusters **407**A, **407**B, and **407**C may be communicatively coupled to each other via network **705**

(e.g., one or more of a LAN, WAN, wireless network, wired network, the Internet, etc.). Although FIG. **4** illustrates the storage clusters **407**A, **407**B, and **407**C as being communicatively via the network **705**, the storage clusters **407**A, **407**B, and **407**C, in other embodiments, some of the storage clusters **407**A, **407**B, and **407**C may be coupled directly to each other via a communications bridge, interconnected, bus, wire, etc. For example, storage cluster **407**A may be directly coupled to storage cluster **407**B. In another example, storage cluster **407**A may be directly coupled to storage cluster **407**B and coupled to storage cluster **407**B via the network **705**.
(215) The storage clusters **407**A-**407**C include different types of storage nodes and/or may include any number of storage nodes. For example, storage cluster **407**A may include a set of primary storage nodes **410** (e.g., one or more primary storage nodes **410**). Storage cluster **407**B may include a set of secondary storage nodes (e.g., one or more secondary storage nodes **420**). Storage cluster **407**C may include both a set of primary storage nodes **410** and a set of secondary storage nodes **420**. A primary storage node **410** may be referred to as a head node, a head storage node, a control node, a control storage node, a primary node, a master storage node, a master node, etc. A second storage node **420** may be referred to as a slave node, a slave storage node, a secondary node, an expansion node, an expansion storage node, etc.
(216) In one embodiment, a primary storage node **410** includes one or more processing devices **411** (e.g., one or more CPUs, ASICs, FPGAs, a multi-core processors, processing cores, etc.). The processing device **411** may be used by the primary storage node **410** to perform various operations, actions, functions, etc., as discussed in more detail below. The processing device **411** may be referred to as a primary processing device. A primary storage node **410** may also include one or more non-volatile memory modules **413**. A non-volatile memory module **413** may be a device that stores data such that the data in the device remains even when power is no longer supplied to the device. Examples of a non-volatile memory module **413** may include, but are not limited to, SSDs, NVME drives, flash drives, etc.
(217) In one embodiment, a secondary storage node **420** includes a processing device **421** (e.g., one or more CPUs, ASICs, FPGAs, multi-core processors, processing cores, etc.). The processing device **421** may be used by the secondary storage node **420** to perform various operations, actions, functions, etc., as discussed in more detail below. The processing device **421** may be referred to as a secondary processing device. A secondary storage node **420** may also include one or more non-volatile memory modules **423** (e.g., SSDs, NVME drives, flash drives, etc.).
(218) In embodiments, the processing devices (e.g., processing devices **411** and **421**) of the storage nodes may directly determine how erase blocks of flash-based storage devices (e.g., non-volatile memory modules **413** and **423**) of the storage nodes are allocated, written to, and when they are erased. In some embodiments, the processing devices may configure write operations to operate during spans of time on a subset of flash die of the flash-based storage devices to preserve availability of the majority of die of the flash-based storage devices for performance of read operations.
(219) In some embodiments, the non-volatile memory modules **413** and/or **423** may be removable from the primary storage nodes **410** and/or the secondary storage nodes **420**. For example, the non-volatile memory modules **413** and/or **423** may be removed, relocated, and/or replaced with other non-volatile memory modules. In another example, the non-volatile memory modules **413** and/or **423** may be moved from one storage node to another (e.g., may be removed from a primary storage node **410** and then inserted into a secondary storage node **420**, from a secondary storage node **420** to a primary storage node **410**, from a first primary storage node **410** to a second primary storage node **410**, from a first secondary storage node **420** to a second secondary storage node **420**, etc.). The non-volatile memory modules **413** and/or **423** may also be removable or insertable while the primary storage nodes **410** and/or the secondary storage nodes **420** are in operation. For example, the non-volatile memory modules **413** and/or **423** may be hot swappable. The non-volatile memory modules **413** and/or **423** may also be heterogenous (e.g., non-uniform). For example, there may be

different makes, models, manufacturers, capacities, supported command sets, memory types (e.g., SLC, MLC, TLC, QLC, PLC, etc.) of non-volatile memory modules **413** and/or **423**.
(220) In one embodiment, the data store on the non-volatile memory modules **413** and/or **423** may be reincorporated, reused, and/or re-accessible when the non-volatile memory modules **413** and/or **423** are removed, relocated, and/or replaced.
(221) In one embodiment, the non-volatile memory modules **423** (e.g., some or all of the non-volatile memory modules **423**) may have lower performance when compared with non-volatile memory modules **413**. For example, the non-volatile memory modules **423** may have higher/longer access times than the non-volatile memory module **413**. In another example, the non-volatile memory modules **423** may have less bandwidth/throughput (e.g., read throughput and/or write throughput) when compared to non-volatile memory module **423**. The network interconnect to the secondary cluster may also itself result in less storage system performance from its non-volatile memory modules than from storage non-volatile memory modules in the primary storage node.
(222) In some embodiments, the primary storage nodes **410** may operate as controllers/managers of the storage system. For example, a primary storage node **410** may determine which storage operations should be performed and may instruct one or more secondary storage nodes to perform the storage operations (e.g., rebuild/reconstruction operations for rebuilding data, compression operations for compressing data, garbage collection operations for freeing data blocks, etc.). In another example, the primary storage node **410** may delegate/offload storage operations to the secondary storage node **420**. For example, an authority **517** may manage/control a portion of data and/or a subset of the non-volatile memory modules **423** of the secondary storage node **420**. The authority **517** may offload/delegate storage operations that are related to or associated with the portion of data and/or the subset of non-volatile memory modules **423**, to the secondary storage node **420**.
(223) In one embodiment, one or more of the primary storage nodes **410** and the secondary storage nodes **420** may use different write paths when writing the memory to the non-volatile memory modules **413** and/or **423**. A write path may refer to the components, memories, circuits, that data may go through when storing/accessing data. For example, a primary storage node **410** may receive data from a client/computing device and may first store the data in an NVRAM, then write the data to the non-volatile memory modules **413**. One example write path may be writing data from NVRAM directly to a MLC flash memory (e.g., TLC memory, QLC memory, penta-level cell (PLC) memory). Another example write path may be writing data from NVRAM, to an SLC flash memory, and then to a MLC flash memory. A further example write path may include writing data directly to MLC flash memory (e.g., bypassing the NVRAM).
(224) In one embodiment, a primary storage node **410** may determine that the programming mode for one or more portions of non-volatile memory modules **413** and/or **423** should be changed. For example, a portion of a non-volatile memory module **413** may use an MLC programming mode (e.g., the non-volatile memory module **413** may include MLC cells that operate in MLC mode). The primary storage node **410** may determine that portion of the non-volatile memory module **413** use a different programming mode (e.g., MLC cells should operate in SLC mode). The primary storage node **410** may vary the programming modes for different portions of the non-volatile memory modules **413** and/or **423** based on various parameters. For example, the primary storage node **410** may change different portions of the non-volatile memory modules **413** and/or **423** to SLC mode and back to QLC mode based on program/erase cycles of the cells in the non-volatile memory modules **413** and/or **423**.
(225) In one embodiment, a processing device **421** may have less computing power (e.g., less processing power) than a processing device **411**. For example, a processing device **421** may have a lower clock frequency than a processing device **411**. In another example, a processing device **421** may have fewer processing cores than a processing device **411**. In a further example, a processing device **421** may have less memory (e.g., less cache, fewer memory registers, etc.) than a processing

device **411**. In another example, the processing device **421** may be able to perform fewer operations/computations during a time period when compared to the processing device **411**.
(226) In some embodiments, the processing device **421** may be different then the processing device **411**. For example, the processing device **421** may be a different make, model, etc., of processor than the processing device **411**. In other embodiments, the processing device **421** may be the same processor (e.g., the same make, model, etc.) as processing device **411**. However, processing device **421** may be operated at lower clock speeds and/or may use less energy/power when compared to processing device **411**.
(227) In some embodiments, the primary storage nodes **410** and/or the secondary storage nodes **420** may include other components, devices, circuits, etc., to perform various other actions, operations, functions, tasks, etc. For example, the primary storage nodes **410** and/or the secondary storage nodes **420** may include graphics processing units (GPUs), an add-on board (e.g., a circuit board, a printed circuit board, etc., that may be used to add additional non-volatile memory modules to a storage node), peripheral components, etc. These other components may be hot swappable, modular, configurable, etc., and may be used to add capabilities to the storage nodes.
(228) In one embodiment, the primary storage node **410** may include one or more authorities (not illustrated in FIG. **4**). An authority may control how and where data is stored in the non-volatile memory modules **413** and **423** of the storage system **400**. For example, an authority may determine how segments, erase blocks, allocation units, etc., are created and used. In another example, an authority may determine how to create stripes (e.g., RAID stripes), parity data, etc. Each primary storage node **410** may include any number of authorities. For example, a primary storage node **410** may include one authority, ten authorities, or any appropriate number of authorities. Each authority may be responsible for managing a set of blocks/segments (e.g., a range of blocks/segments), one or more non-volatile memory modules **413** and/or **423**. The authorities may perform functions, operations, actions, tasks, etc., discussed above in conjunction with FIGS. **2**A-**2**G.
(229) In one embodiment, a first primary storage node **410** may be able to continue or take over the operations, functions actions, tasks, etc., of a second primary storage node **410**, if the second primary storage node **410** fails. For example, if the second primary storage node **410** crashes, resets/reboots, or is otherwise unable to operate, the first primary storage node **410** may be able to continue or take over the operations, functions actions, tasks, etc., of the second primary storage node **410**. In addition, the operations, functions actions, tasks, etc., of the second primary storage node **410** may be distributed to (e.g., allocated to, spread out among) multiple other primary storage nodes **410**. For example, two, ten, or some other appropriate number of primary storage nodes **410** may each take over a portion of the operations, functions actions, tasks, etc., of the second primary storage node **410**.
(230) In one embodiment, an authority may be transferred, moved, reallocated, reassigned, etc., from one primary storage node **410** to another primary storage node **410**. For example, the data structures, metadata, processes, services, etc., for an authority may be copied from one primary storage node to another. In another example, the data structures, metadata, processes, services, etc., for an authority may be copied from one primary storage node **410** to a secondary storage node **420** (e.g., when the secondary storage node **420** is converted to a primary storage node, as discussed above).
(231) In one embodiment, the secondary storage node **420** may initially lack authorities. For example, there may be no authorities on the secondary storage node **420** (e.g., the secondary storage node **420** may not have any authorities) when the secondary storage node **420** initiates operation (e.g., initially boots or starts). The secondary storage node **420** may include one or more authorities after it is converted into a primary storage node **410**.
(232) In one embodiment, a secondary storage node **420** may be converted to a primary storage node (e.g., may operate as a primary storage node). For example, if a primary storage node **410** fails, one or more a secondary storage node **420** may be used to perform the operations, functions

actions, tasks, etc., of failed primary storage node **410**. The conversion from secondary storage node to primary storage node may be temporary. For example, the secondary primary nodes **420** may operate as primary storage nodes for a period of time until the failed primary storage node **410** is restored or until a new primary storage node **410** is added to the storage system.
(233) In one embodiment, one or more device hosts (not illustrated in FIG. **4**) may perform the one or more storage operations. For example, the secondary storage node **420** may include one or more device hosts. A device host may be hardware, software (e.g., a service, a process, a thread, a daemon, a driver, etc.), firmware, or a combination thereof, that may manage, control, etc., one or more non-volatile memory modules **423**. Device hosts may be described in more detail below.
(234) In various embodiments, the storage system **400** allows for dynamic configuration of the primary storage nodes **410** and the secondary storage nodes **420**. For example, primary storage nodes **410** and/or secondary processing storage **420** may be added and/or removed from the storage clusters. In addition, the non-volatile memory modules may also be removable and changeable, further increasing the configurability of the storage system. Because the primary storage nodes **410** may control/manage the operations (e.g., storage operations) of the storage system **400**, more secondary storage nodes **420** may be used in the storage system. The secondary storage nodes **420** may include less processing power and/or lower performance non-volatile memory modules. This may allow the storage system **400** to operate more efficiently (e.g., using less power and with reduced costs) when providing access to data. The secondary storage nodes **420** also allow the storage capacity of the storage system to be increased more efficiently (e.g., using less power and with reduced costs).
(235) 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.
(236) 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.
(237) 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.
(238) 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.
(239) The storage systems described above may, either alone or in combination, by configured to serve as a continuous data protection store. A continuous data protection store is a feature of a storage system that records updates to a dataset in such a way that consistent images of prior contents of the dataset can be accessed with a low time granularity (often on the order of seconds, or even less), and stretching back for a reasonable period of time (often hours or days). These allow access to very recent consistent points in time for the dataset, and also allow access to access to points in time for a dataset that might have just preceded some event that, for example, caused parts of the dataset to be corrupted or otherwise lost, while retaining close to the maximum number of updates that preceded that event. Conceptually, they are like a sequence of snapshots of a dataset taken very frequently and kept for a long period of time, though continuous data protection stores are often implemented quite differently from snapshots. A storage system implementing a data continuous data protection store may further provide a means of accessing these points in time, accessing one or more of these points in time as snapshots or as cloned copies, or reverting the dataset back to one of those recorded points in time.
(240) 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.
(241) 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.
(242) 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.
(243) 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).
(244) Advantages and features of the present disclosure can be further described by the following statements: 1. A storage device configured to: receive a read request with a high priority indication; determine whether an in progress flash programming operation would delay processing the read request for a threshold amount of time; and in response to determining that the in progress flash programming operation delays processing the read request for the threshold amount of time, interrupt the in progress flash programming operation, wherein the storage device is to process the read request upon interrupting the in progress flash programming operation. 2. The storage device of statement 1, wherein the in progress flash programming operation is a write operation to an erase block programmed to store at least three bits per cell. 3. The storage device of statement 1, wherein the in progress flash programming operation is an erase operation. 4. The storage device of statement 1, wherein the in progress flash programming operation is performed internal to the storage device as part of internal maintenance. 5. The storage device of statement 1, wherein in response to determining that the in progress flash programming operation cannot be interrupted, the storage device is further configured to fault the read request to indicate that the read request cannot be processed in the threshold amount of time. 6. The storage device of statement 1, further configured to receive one or more subsequent read requests; and process the read request and the one or more subsequent read requests during the interrupt. 7. The storage device of statement 1, further configured to process the read request; and resume the in progress flash programming operation upon completion of processing the read request. 8. The storage device of statement 1, wherein the storage device is a direct-mapped storage devices. 9. The storage device of statement 1, wherein the storage device supports zoned namespaces. 10. The storage device of statement 1, wherein the storage device supports an abstraction that organizes virtual address space of the storage device such that virtual segments have a size that enables the virtual segments to be written to sequentially within a same erase block of the storage device. 11. A storage device, configured to: receive a first request to program a first erase block in a first programming mode; receive a second request to program a second erase block in a second programming mode, wherein the second programming mode differs from the first programming mode; receive a third request to store first data in the first erase block; receive a fourth request to store second data in the second erase block; store the first data in the first erase block using the first programming mode; and store the second data in the second erase block using the second programming mode. 12. The storage device of statement 11, wherein the storage device is a direct-mapped storage device. 13. The storage device of statement 11, wherein the storage device supports zoned namespaces. 14. The storage device of

statement 11, wherein the storage device supports an abstraction that organizes virtual address space of the storage device such that virtual segments have a size that enables the virtual segments to be written to sequentially within a same erase block of the storage device. 15. A storage device comprising a plurality of erase blocks, the storage device configured to: provide, through an interface, a number of available units, each unit representing a set of the erase blocks that can be programmed in a plurality of modes, each with a set of characteristics that includes differing effective capacity; receive a first request to form a first unit set comprising a first number of the available units and configured to be programmed in a first mode of the plurality of flash modes, resulting in a first virtually addressable capacity corresponding to a first effective capacity for the first number of the available units when programmed in the first mode; receive a second request to form a second unit set of at least a second number of the available units and configured to be programmed in a second mode of the plurality of flash modes, resulting in a second virtually addressable capacity corresponding to a second effective capacity for the second number of the available units when programmed in the second mode; receive a third request to store first data to a first virtual address associated with the first unit set; receive a fourth request to store second data to a second virtual address associated with the second unit set; store the first data into erase blocks of one or more units of the first unit set and programmed in the first mode; and store the second data into erase blocks of one or more units of the second unit set and programmed in the second mode. 16. The storage device of statement 15, wherein each unit of the available units represents a programming mode applied to one or more flash die. 17. The storage device of statement 15, wherein each unit of the available units can be operated in parallel with every other unit, wherein read, erase, write, and maintenance operations of a first unit of the available units do not prevent parallel read, erase, write, or storage device background operations of a second unit of the available units. 18. The storage device of statement 15, wherein a request to the storage device can prevent and subsequently allow performance of storage device background operations of a particular unit of the available units. 19 The storage device of statement 15, wherein a unit can be programmed with additional internal redundancy to increase its internal reliability. 20. The storage device of statement 15, wherein a unit set can be programmed by the storage device controller to store data segments using erasure codes across the flash die in the set. 21. A storage system, comprising: a plurality of storage nodes in a chassis, wherein each of the plurality of storage nodes comprises a processing device, memory, and a plurality of slots for hosting flash-based storage devices, wherein each of the flash-based storage devices provide interfaces configured to allow the plurality of storage nodes to optimize use of flash memory on the flash-based storage devices, wherein the processing device of each of the plurality of storage nodes is configured to; receive read requests and write requests for data of the storage system; direct the read requests to a first set of storage nodes of the plurality of storage nodes that host one or more flash-based storage devices that store data for the read requests; store written data in erasure coded segments distributed in shards across a set of the flash-based storage devices hosted by the plurality of storage nodes, wherein the shards directed to a particular flash-based storage device are delivered to and written by a particular storage node hosting the particular flash-based storage device; and balance write activity to independent performance zones of the flash memory of the flash-based storage devices to ensure the read requests are processed in a threshold amount of time. 22. The storage system of statement 21, wherein the processing device of each plurality of storage nodes are configured to: directly determine how erase blocks of the flash-based storage devices are allocated, written to, and when they are erased. 23. The storage system of statement 22, wherein write operations are configured by the plurality of storage nodes to operate during spans of time on a subset of flash die of the flash-based storage devices to preserve availability of the majority of die of the flash-based storage devices for performance of read operations. 24. The storage system of statement 21, wherein the plurality of storage nodes are configured to program a first number of erase blocks of the flash-based storage devices in single level cell (SLC) mode and a second number of erase blocks of the

flash-based storage devices programmed with at least 3 bits per cell. 25. The storage system of statement 21, wherein the plurality of storage nodes are configured to configure a first subset of units of a configurable flash-based storage device as SLC, and a second subset of units of a configurable flash-based storage device as at least three bits per cell. 26. The storage system of statement 21, wherein write operations are configured by the plurality of storage nodes to operate during spans of time on a subset of the configured units of the flash-based storage. 27. The storage system of statement 21 wherein internal read requests associated with front-end read requests are issued to the flash-based storage devices as high priority reads that can interrupt conflicting QLC modification operations that delay the internal read requests. 28. A method including any of statements 1-27. 29. A non-transitory computer readable storage medium storing instructions that, when executed by a processing device, cause the processing device to perform any of statements 1-27.
(245) 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.
(246) 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.
(247) 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.
(248) In embodiments, a storage system may include a die-aware scheduler (also referred to as "scheduler" hereafter) that sits between an authority (e.g., authority **168**) and one or more storage devices of the storage system. The scheduler may be responsible for managing the flow of input-output (I/O) operations, such as read, write and erase operations, flowing between the authority and the storage device(s). For example, a scheduler may receive I/O operations from the authority and assign the I/O operations to one or more storage devices of a storage system. The scheduler may receive hints about the nature and importance of the operations from the authority and constraints that need to be worked about from the storage device. Different types of operations performed by the storage system may take different amounts of time to complete. For example, a write operation to store data at a storage device may take longer to complete than a read operation to read data that was previously stored at a storage device. Furthermore, different types of operations may have different priority levels. For example, a read operation to read data from a storage device may be a higher priority than a write operation to store data at a storage device.
(249) In a conventional storage system, once a write operation has been initiated to store data at one of the storage devices of the storage system, subsequent operations are delayed until the storage device has completed the write operation. For example, a user may submit a request to read data from a storage device of the storage system. The storage device, however, is currently

performing a write operation and cannot read the data until the write operation is completed. This may result in unwanted delays in providing the data to the user of the storage system.
(250) Turning now to embodiments of a storage system that intelligently schedules operations to perform read operations to read data from a storage device while the storage device is performing a write operation. In some embodiments, the write operation that stores data at a storage device may include multiple stages. For example, a write operation that is programmed in quad-level cell (QLC) mode may include five stages to store data at a storage device. In such an embodiment, one or more read operations may be performed between two of the stages. For example, after the first stage of a multiple stage write operation, a read operation may be performed to read data from the storage device before the second stage of the multiple stage write operation begins.
(251) In embodiments, the scheduler may interrupt (also referred to as "suspend" hereafter) the write operation to enable the storage device to perform one or more read operations to read data from the storage device. For example, upon receiving one or more requests to read data from a storage device, the scheduler may assign read operations to the storage device and transmit an indication that the storage device is to suspend an ongoing write operation, perform the read operations, and subsequently resume the write operation once the read operations have been completed.
(252) Accordingly, embodiments of the disclosure provide for an improved storage system by performing read operations during a write operation. Performing the read operations during the write operation, rather than waiting until the write operation is complete, reduces the latency and improves the performance of the storage system.
(253) It should be noted that the use of read operations being performed during write operations in the examples below are shown for illustrative purposes only. Embodiments of the disclosure may have other types of operations performed during write operations of storage devices of the storage system. Furthermore, in some embodiments, operations may be performed during an erase operation rather than a write operation.
(254) FIG. **5**A is an illustration of an example of a scheduler of a storage system **500** receiving a request to read data from a storage device of the storage system, in accordance with embodiments of the disclosure. In embodiments, storage system **500** may correspond to one of the storage systems previously described in FIGS. **1**A-**4**. Storage system **500** includes scheduler **388** and storage devices **356***a-n*. Storage devices **356***a-n* represent any number of storage devices of storage system **500**. In some embodiments, storage devices **356***a-n* may be storage resources **392** as described in FIG. **3**F. Storage devices **356***a-n* include corresponding data **504***a-n* and queues **506***a-n*. It should be noted that data **504***a-n* and queues **506***a-n* are shown for illustrative purposes only and are not physical components of storage devices **356***a-n.*
(255) In embodiments, storage devices **356***a-n* may be solid-state storage devices that are capable of suspending operations. The storage devices **356***a-n* may provide an interface that the scheduler **388** may use to transmit high priority operations, such as read operations, along with an indication that the storage device is to suspend an ongoing write operation to perform the read operation or perform the read operation in between stages of a multiple stage write operation, as will be described in further detail below.
(256) In embodiments, the interface may permit supplying different priority levels to data that trigger different actions by the storage devices **356***a-n*. For example, a request having an "immediate" priority level may trigger pre-empting or interrupting an in progress write operation. In some embodiments, the immediate priority indication may further mean that a read faults immediately with an indication that the read could not be completed in a timely manner if there is an in progress flash operation that is not completing soon and cannot be interrupted.
(257) In another example, a request having a "medium" priority level may encourage some amount of time waiting in expectation that multiple read requests may be received that can be grouped together in some way that is beneficial to the operation of the storage device, such as, by reducing

the number of interrupted in progress flash operations. In a further example, a request having an even lesser priority (e.g., a “background” priority) may indicate that the operation can be scheduled after any in progress flash operation completes. In some embodiments, the background priority may be further divided into subsets. For example, a “throughput” priority indication may cause the storage device to ensure reasonable progress across a set of “throughput” priority requests (but where individual operation latency can be high if that helps the device) and “idle” priority where operations can be held off until opportune moments without too many concerns that reasonable progress is made across all the “idle” requests.
(258) Storage system **500** may utilize these different priority indications to intelligently schedule operations to be performed by storage devices **356***a-n*. During the operation of storage system **500**, there is an almost constant stream of background read operations, whether it is from data refresh, garbage collection, or space adjustment. In order to decrease the front end read latency, the scheduler **388** of storage system **500** may utilize these priorities to differentiate frontend operations, while ensuring not to starve out the backend operations.
(259) For example, the storage system **500** may utilize pre-empting reads for high priority reads, such as when alternate sources of the content are not available and immediate priority for the most latency sensitive reads, and medium priority for most other reads. In another example, the storage system **500** may utilize throughput priority for latency-insensitive background tasks where forward progress is still needed for the storage system **500** services to continue operating (e.g., garbage collection, particularly if the free blocks are not otherwise accumulating fast enough to keep up with incoming writes), and idle priority for latency-insensitive background tasks where high throughput is less of a consideration (e.g., scrubbing).
(260) In embodiments, immediate priority could also be used for reads that should be performed immediately, if possible, with a fault indicating abort if the read is not possible due to an in progress, uninterruptible flash operation. In some embodiments, the fault may trigger an alternate means of reading the data.
(261) Furthermore, in storage system **500** the throughput of single-level cell (SLC) write operations should be less than or equal to the throughput of quad-level cell (QLC) write operations. This is because the SLC write throughput represents incoming user data, while the QLC write throughput represents writing SLC data onto QLC flash cells as well as maintenance operations (e.g., garbage collection, data refresh, metadata handling, etc.) The scheduler **388** may be configured to keep track of and average out QLC throughput and, once the scheduler **388** has identified that the SLC throughput exceeds the QLC throughput, throttle down the SLC throughput as needed.
(262) Data **504***a-n* may correspond to data stored at the corresponding storage devices **356***a-n*. For example, data **504***a-n* may be stored at one or more data blocks of storage devices **356***a-n*. Queues **506***a-n* may correspond to data structures that include a listing of operations to be performed by the corresponding storage devices **506***a-n*. In some embodiments, queues **506***a-n* may include a listing of read operations (or other types of operations) that are to be performed during write operations at storage devices **356***a-n*, as will be described in additional detail below.
(263) Referring to FIG. 5A, the scheduler **388** may receive a read request **502** to read data (e.g., data **504***a-n*) from one or more of storage devices **356***a-n*. For example, the scheduler **388** may receive a request to read data from a user of the storage system **500**.
(264) FIG. 5B is an illustration of an example of a scheduler of a storage system **550** receiving queue information in accordance with embodiments of the disclosure. Storage system **550** includes similar components as storage system **500** as previously described at FIG. 5A.
(265) Storage devices **356***a-n* may transmit information associated with queues **506***a-n* to the scheduler **388**. The scheduler **388** may receive the queue information and utilize the queue information to determine which of storage devices **356***a-n* the read request should be sent to. In embodiments, the information associated with queues **506***a-n* may be data structures that include a listing of operations to be performed during write operations of storage devices **356***a-n*. For

example, queues **506***a-n* may include one or more read operations that are to be performed intermittently during the performance of write operations.
(266) In some embodiments, the scheduler **388** may transmit a request for the queue information to storage devices **356***a-n*. In embodiments, storage devices **356***a-n* may transmit the queue information to scheduler **388** upon the occurrence of a triggering condition. For example, storage devices **356***a-n* may transmit queue information to scheduler **388** if the queue is empty and does not include any operations to be performed during write operations. In another example, storage devices **356***a-n* may transmit queue information to scheduler **388** if a determined amount of time has elapsed since the previous transmission of queue information to scheduler **388**. In embodiments, other triggering conditions may be used to cause the transmission of queue information from storage devices **356***a-n* to scheduler **388**.
(267) FIG. **6**A is an illustration of an example of a scheduler of a storage system **600** inserting a read operation into a queue of a storage device, in accordance with some embodiments of the disclosure. Storage system **600** includes similar components to the storage systems previously described at FIGS. **5**A and **5**B.
(268) In FIG. **6**A, scheduler **388** has previously received queue information from storage devices **356***a-n*, as previously described at FIG. **5**B. As illustrated in FIG. **6**A, queues **506***a-n* are data structures that include a number of read operations that are to be performed during write operations of the corresponding storage devices **356***a-n*. For example, queue **506***a* includes three read operations (e.g., read operation **1**, read operation **2**, and read operation **3**) that are to be performed by storage device **356***a* during one or more write operations to write data to storage device **356***a*. Similarly, queue **356***n* includes two read operations (e.g., read operation **4** and read operation **5**) that are to be performed by storage device **356***n* during one or more write operations to write data to storage device **356***n*. Meanwhile, queue **506***b* is empty, indicating that there are no operations to be performed by storage device **356***b* during one or more write operations to write data to storage device **356***b*. In some embodiments, upon identifying empty queue **506***b* of storage device **356***b*, the scheduler **388** may transmit a request to storage device **356***b* to perform the one or more read operations **602**. The request may include an indication that storage device **356***b* is to perform the one or more read operations during ongoing write operations to write data to storage device **356***b*. For example, the indication may indicate that the one or more read operations are high priority operations that should be performed during an ongoing write operation by storage device **356***b*.
(269) In some embodiments, the scheduler **388** may pre-emptively transmit an indication to the storage device **356***b* that one or more read requests have been received and are going to be sent to storage device **356***b*. The pre-emptive transmission of the indication may cause the storage device **356***b* to suspend an ongoing write operation (or prevent storage device **356***b* from starting a new write operation) in anticipation of receiving and performing the one or more read operations. The indication may also cause storage device **356***b* to suspend or delay performance of other types of operations that are deemed has having a lower priority than the read requests.
(270) In an embodiment, the scheduler **388** may group multiple read requests together in various ways that is beneficial to the operation of storage device **356***b*. For example, the scheduler **388** may group together read requests to reduce the number of interrupted operations.
(271) In embodiments, the scheduler **388** may pre-emptively transmit the indication in response to an occurrence of an event. For example, if the data of a read request is only available on storage device **356***b*, or if alternate sources of the data are not available, the scheduler **388** may pre-emptively transmit the indication to storage device **356***b*.
(272) Upon receipt of the one or more read operations, the storage device **356***b* may insert the one or more read operations into queue **506***b* for subsequent performance during an ongoing write operation.
(273) FIG. **6**B is an illustration of an example of a scheduler of a storage system **650** transmitting a read operation to a storage device, in accordance with embodiments of the disclosure. Storage

system **650** includes similar components to the storage systems previously described at FIGS. **5**A, **5**B, and **6**A.
(274) In FIG. **6**B, scheduler **388** has previously received queue information from storage devices **356***a-n*, as previously described at FIG. **5**B. As illustrated in FIG. **6**B, queues **506***a-n* are data structures that include a number of read operations (or other types of higher priority operations) that are to be performed during write operations of the corresponding storage devices **356***a-n*. For example, queue **506***a* includes three read operations (e.g., read operation **1**, read operation **2**, and read operation **3**) that are to be performed by storage device **356***a* during one or more write operations to write data to storage device **356***a*. Similarly, queue **356***b* includes two read operations (e.g., read operation **4** and read operation **5**) that are to be performed by storage device **356***b* during one or more write operations to write data to storage device **356***n*. Queue **506***n* includes one read operation (e.g., read operation **6**) to be performed by storage device **356***n* during one or more write operations to write data to storage device **356***n*. In some embodiments, upon identifying queue **506***n* of storage device **356***n* as having the least number of read operations, the scheduler **388** may transmit the one or more read operations **602** into to storage device **356***n*. The storage device **356***n* may insert the read operation into queue **506***n* to perform during ongoing write operations.
(275) FIG. **7** is an illustration of a timeline **700** of a storage device of a storage system performing read operations during a multiple stage write operation, in accordance with embodiments of the disclosure. The timeline **700** includes a vertical arrow (e.g., time **702**) that represents the passage of time, with the top of time **702** representing the start of the timeline **700** and the bottom representing the end of the timeline **700**. FIG. **7** also includes queue **506** that represents one of queues **506***a-n* of storage devices **356***a-n* as previously described in FIGS. **5**A-**6**B.
(276) In some embodiments, a write operation may include multiple stages of programming blocks of a storage device (not shown) before the data is persisted and stored by the storage device. For example, there may be 5 stages for a write operation that writes data using quad-level cell (QLC) mode. Referring to FIG. **7**, a multiple stage write operation **704** is performed over time **702**. The multiple stage write operation includes four stages, stage A **706**, stage B **708**, stage C **710**, and stage D **712**. This allows there to be three opportunities between the four stages to temporarily interrupt the multiple stage write operation **704** to perform the higher priority read operations included in queue **506**.
(277) Referring to FIG. **7**, queue **506** includes three read operations (read operation **1**, read operation **2**, and read operation **3**) that are to be performed during the multiple stage write operation **704**. After stage A **706** has been completed, the multiple stage write operation **704** may be interrupted to perform read operation **1** to read data from the storage device. Once the data for read operation **1** has been read, stage B **708** of the multiple stage write operation **704** may be performed. After stage B **708** has been completed, the multiple stage write operation **704** may be interrupted to perform read operation **2** to read data from the storage device. Once the data for read operation **2** has been read, stage C **710** of the multiple stage write operation **704** may be performed. After stage C **710** has been completed, the multiple stage write operation **704** may be interrupted to perform read operation **3** to read data from the storage device. Once the data for read operation **3** has been read, stage D **712** of the multiple stage write operation **704** may be performed and the multiple stage write operation **704** is completed.
(278) It should be noted that the number of stages of the multiple stage write operation **704** as well as the number of read operations performed in between the stages are shown for illustrative purposes only. Embodiments of the disclosure may be applied to multiple stage write operations having any number of stages. Embodiments of the disclosure may perform any number of operations, such as read operations, in between the stages of the multiple stage write operation **704**.
(279) FIG. **8** is an illustration of a timeline **800** of a storage device of a storage system suspending a write operation to perform a read operation, in accordance with embodiments of the disclosure. The timeline **800** includes a vertical arrow (e.g., time **802**) that represents the passage of time, with

the top of time **802** representing the start of the timeline **800** and the bottom representing the end of the timeline **800**. FIG. **8** also includes queue **506** that represents one of queues **506***a-n* of storage devices **356***a-n* as previously described in FIGS. **5**A-**6**B.
(280) In some embodiments, rather than performing higher priority operations in between the stages of a multiple stage write operation, a scheduler (not shown) of a storage system may suspend the write operation **804** at any point during the write operation **804**. This may provide an additional improvement in the latency of the storage system, as the storage device (not shown) no longer has to wait until the end of a stage to perform a higher priority operation. In embodiments, the scheduler may suspend the write operation **804** by transmitting an indication to the storage device that the write operation **804** is to be suspended to perform the read operation in queue **506**. Upon receiving the indication, the storage device may suspend the write operation **806**. Upon suspending the write operation **806**, the storage device may perform the read operation **808** from queue **506** to read data from the storage device. Once the read operation is complete, the storage device may resume the write operation **810**.
(281) It should be noted that the number of suspensions of the write operation **804** as well as the number of read operations performed during the suspension are shown for illustrative purposes only. Embodiments of the disclosure may include any number of suspensions during write operation **804**. Embodiments of the disclosure may perform any number of operations, such as read operations, during the suspension of write operation **804**.
(282) In some embodiments, a background read operation may be performed either in between one or more stages of a multiple stage write, or during a suspension of a write operation, as previously described at FIGS. **6** and **8**. For example, a background read operation may be performed if the increase in latency resulting from the background read does not cause the latency to go above a threshold amount.
(283) FIG. **9** is an example method **900** to intelligently process read requests in accordance with embodiments of the disclosure. In general, the method **900** may be performed by processing logic that may include hardware (e.g., processing device, circuitry, dedicated logic, programmable logic, microcode, hardware of a device, integrated circuit, etc.), software (e.g., instructions run or executed on a processing device), or a combination thereof. In some embodiments, processing logic of a storage device, such as the storage devices described in FIGS. **1**A-**8**, may perform the method **900**.
(284) In some embodiments, a storage device of a storage system may include an interface that allows for the application of deadlines for read requests. In embodiments, the deadline may be received from a storage controller or scheduler of a storage system. In some embodiments, the deadline may cause storage device to abort a read request if the read request is not completed by the deadline. For example, if the deadline is 50 milliseconds (ms) to read the data from storage device, and 51 ms has elapsed and the data has not been read, then the storage device aborts the read request. In embodiments, the storage device may immediately abort the read request if the storage device determines that the read request will not be completed by the deadline.
(285) In some embodiments, the deadlines or other general scheduling hints may be used to group together read requests that may be accomplished together in some way. For example, multiple read operations could be grouped together to be performed during the same interrupted operation.
(286) Method **900** may begin at block **902**, where the processing logic receives a read request that includes a high priority indication to read data from the storage device.
(287) At block **904**, the processing logic determines if a flash programming operation being performed by the processing logic would delay the processing of the read request by at least a threshold amount of time if the read were to wait for the flash programming operation to complete (as is often the case if a read is directed to the same flash die of a flash package). The performance of the flash programming operation may lock out read requests or cause significant delays in the processing of the read request. In some embodiments, the threshold amount of time may be

received from a user of the storage device. In embodiments, the threshold may be received from a storage controller of a storage system. In an embodiment, the flash programming operation may have a lower priority than the read request.
(288) In embodiments, the flash programming operation may be a write operation to store data at the storage device. In some embodiments, the write operation may be a high bit per cell write operation. A high bit per cell write operation may be a write operation where at least three bits are stored per cell of the flash memory. For example, the write operation may be performed using a triple level cell (TLC), quad-level cell (QLC), penta-level cell (PLC), or higher bit per cell programming mode. The write operation may store pages of data, where a page is one layer of these bits, and sequential pages are generally of different ranges of cells and not different bits of the same cells. In an embodiment, the write operation may be the result of one or more requests to store data in the flash memory of the storage device.
(289) In some embodiments, the flash programming operation may be an erase operation. In an embodiment, the erase operation may be the result of one or more requests to erase data stored in the flash memory of the storage device. In embodiments, the flash programming operation may be performed internal to the storage device as part of internal maintenance.
(290) At block **906**, the processing logic completes the flash programming operation if the processing logic determines that the flash programming operation does not delay the read request by the threshold amount of time, and then processes the read request.
(291) At block **908**, the processing logic determines if the flash programming operation can be interrupted upon determining that the flash programming operation delays the read request by the threshold amount of time.
(292) A flash programming operation can be interrupted depending on particular characteristics of the particular flash chip's implementation, where implementations have different restrictions on how many times or for what durations of time, a flash programming operation can be interrupted. At block **910**, the processing logic aborts the read request if the processing logic determines that the ongoing operation cannot be interrupted. In embodiments, upon aborting the read request, the processing logic may generate an indication that the read request has been aborted. For example, the indication may indicate that the storage device is busy and cannot process the read request. The indication may then be provided to a user of the storage device, a storage controller of a storage system, or any other recipient. In some embodiments, the processing logic may fault the read request to indicate that the read request cannot be processed in the threshold amount of time.
(293) In some embodiments, the indication may further indicate that the read request was aborted because the read request could not be performed in a timely manner. This may be an optional sub-behavior of the high priority indication of block **902**.
(294) At block **912**, the processing logic interrupts the flash programming operation upon determining that the flash programming operation can be interrupted.
(295) At block **914**, the processing logic processes the read request. In some embodiments, the processing logic may receive one or more subsequent read requests to read data from the storage device. The processing logic may decide to process these subsequent read requests during the interruption of the flash programming operation along with the read request received at block **902**.
(296) At block **916**, the processing logic resumes the flash programming operation upon completing the processing of the read request.
(297) FIG. **10** is an illustration of an example of reducing latency of a storage system **1000** using a die-aware scheduler, in accordance with embodiments of the disclosure. As previously described, a die-aware scheduler (also referred to as “scheduler” hereafter) sits between an authority (e.g., authority **168**) and one or more storage devices of the storage system. The scheduler may be responsible for managing the flow of input-output (I/O) operations, such as read, write, and erase operations, flowing between the authority and the storage device(s). For example, a scheduler may receive different types of I/O operations from the authority and assign the I/O operations to one or

more storage devices of a storage system. The scheduler may receive hints about the nature and importance of the operations from the authority and constraints that need to be worked about from the storage device. In embodiments, a die-aware scheduler may be able to reduce the latency of a storage system. There are multiple constraints from storage devices of a storage system. In general, only one operation can be executed per die at a time. There is also a limit of 100 operations per drive due to energy constraints (e.g., power token limits associated with an available amount of stored energy in the form of a battery or a super capacitor that can be used to complete partial operations on a power failure to ensure that the flash is in a usable state the device is powered up in the future). Furthermore, there are slower write speeds for certain programming modes of the storage device. For example, the write speed of a write operation using a quad level cell (QLC) mode is typically slower than the write speed of a write operation using a single level cell (SLC) mode, often quite substantially slower.
(298) Additionally, there are multiple constraints from an authority of the storage system. For user-visible read operations to read data stored at the storage system, if a read is queued up behind a 60 millisecond (ms) write operation, the latency may become unacceptable for a user of the storage system.
(299) Also, in the storage system write operations for data over a certain size are written directly to flash memory (also referred to as a “bypass write” hereafter). If these write operations are stuck behind a 60 ms write operation the latency for the end-user may become unacceptable (even if a write is staged, that flash memory will often be written eventually, but it may then become a “background” operation which can occur at an unpredictable future time from the point of view of the user of the storage device). For example, a write operation having 128K of data may be written directly to flash memory of a storage device. In some embodiments, the data may be written to flash using a SLC programming mode. In other embodiments, the data may be written to flash using a QLC programming mode at a die that is not currently performing a previous 128K write. In embodiments, the write operation having 128K of data may be performed if the number of idle dies of the storage device is above a threshold amount and available operations given power token limits of the storage device.
(300) As previously described at FIG. **5**A, in embodiments of a storage system SLC write operation throughput should be less than or equal to QLC write operation throughput to ensure sustained write operation performance. In addition, the scheduler **388** may also ensure that reads of SLC data have a high enough priority so that while managing the traffic of SLC writes, the SLC reads can be processed unencumbered. In an embodiment, the scheduler **388** may utilize separate queues (not shown) for read and write operations for SLC-only dies to ensure that even as write operations are queuing up, they do not hinder the progress of flushing out QLC writes.
(301) In embodiments, the queues may be configured to operate against one or more characteristics of the flash memory or of the behavior of the storage device. In some embodiments, the storage device **356** may provide feedback to scheduler **388** on characteristics of storage device **356**. For example, storage device **356** may provide a number of independent channels of storage device **356** that can benefit from the use of multiple queues.
(302) Referring to FIG. **10**, the storage system **1000** may correspond and include components from any of the storage systems described in FIGS. **1**A-**4**. The storage system **1000** includes a storage device **356** having dies **1002***a-h* for the storage of data and a die-aware scheduler **388**, as previously described. To improve the latency of the storage system **1000**, the scheduler **388** may split up dies **1002***a-h* as having SLC only dies and QLC only dies. For example, in FIG. **10**, dies **1002***a-b* are designated as SLC only dies, meaning that the writes of data to dies **1002***a-b* are performed using a SLC programming mode. Similarly, dies **1002***c-h* are designated as QLC only dies, meaning that writes of data to dies **1002***c-h* are performed using a QLC programming mode.
(303) The scheduler **388** may create one or more “dedicated lanes” for the SLC only dies **1002***a-b*, meaning that while the QLC only operations need to fight for power tokens, the SLC only

operations have a dedicated power token per die and thus there is always an SLC operation outstanding. This allows to reduce the latency for bypass writes and user-visible reads reading from data on the SLC dies **1002***a-b*.
(304) It should be noted that the programming modes (e.g., SLC and QLC), number of dies of the storage device, and the proportion of SLC only dies to QLC only dies as described in FIG. **10** are shown for illustrative purposes only. In embodiments, different programming modes may be assigned to dies of the storage device, and the characteristics and capabilities of actual flash implementations can vary considerably. For example, a first programming mode having a lower latency may be assigned to a first set of dies, and a second programming mode having a higher latency may be assigned to a second set of dies. Furthermore, the storage device may have any number of dies and differing proportions of dies designated for particular programming modes. In some embodiments, some dies of a storage device may not be designated as using one programming mode to store data and any programming mode may be used.
(305) FIG. **11** is an example method **1100** to improve storage system latency using differing programming modes in accordance with embodiments of the disclosure. In general, the method **1100** may be performed by processing logic that may include hardware (e.g., processing device, circuitry, dedicated logic, programmable logic, microcode, hardware of a device, integrated circuit, etc.), software (e.g., instructions run or executed on a processing device), or a combination thereof. In some embodiments, processing logic of a storage device of a storage system, as previously described at FIGS. **1**A-**4**, may perform the method **1100**.
(306) Method **1100** may begin at block **1102**, where the processing logic receives a first request to program a first erase block in a first programming mode. In embodiments, the first programming mode to be used is identified as part of the first request.
(307) At block **1104**, the processing logic receives a second request to program a second erase block in a second programming mode. In embodiments, the second programming mode to be used is identified as part of the second request.
(308) At block **1106**, the processing logic receives a third request to store first data in the first erase block.
(309) At block **1108**, the processing logic receives a fourth request to store second data at the second erase block.
(310) At block **1110**, the processing logic stores the first data in the first erase block using the first programming mode.
(311) At block **1112**, the processing logic stores the second data in the second erase block using the second programming mode.
(312) FIG. **12** is an example method **1200** to optimize data storage using different flash programming modes, in accordance with embodiments of the disclosure. In general, the method **1200** may be performed by processing logic that may include hardware (e.g., processing device, circuitry, dedicated logic, programmable logic, microcode, hardware of a device, integrated circuit, etc.), software (e.g., instructions run or executed on a processing device), or a combination thereof. In some embodiments, processing logic of a storage device of a storage system, as previously described at FIGS. **1**A-**4**, may perform the method **1200**.
(313) Method **1200** may begin at block **1202**, where the processing logic provides, through an interface, a number of available units of the storage device. Each of the units may represent a set of erase blocks of the storage device that can be programmed in different flash modes (e.g., SLC, MLC, TLC, QLC, etc.) The different flash modes may each have different effective capacities, performance characteristics, and susceptibility to failure from repeated erase/program cycles resulting, for example, in a differing number of full unit writes per day for an expected lifespan of five years (a variation of a common means of describing lifespan for flash-based storage devices). In some embodiments, the processing logic may provide programming modes that do internal RAID or some other type of erasure coding across the available units, if a sufficient number of

units are grouped together, to increase the reliability of the data stored at the units. It should be noted that embodiments of the disclosure are not limited to flash programming modes, but may utilize other types of programming modes, such as added internal redundancy modes.
(314) In some embodiments, a unit may simply represent some number of erase blocks (for example, a group of 100 erase blocks). In some embodiments, each of the units may represent one or more flash die of the storage device. In embodiments, each unit of the available units can be operated in parallel with every other unit. For example, read, erase, write, and maintenance operations of a first unit of the available units do not block parallel read, erase, write, or storage device background operations of a second unit of the available units. In embodiments, a request received by the storage device may be able to prevent and subsequently allow background operations of a particular unit of the storage device to be performed (when prevented, users of the storage device can be assured that read and write operations, for example, will not be slowed down by background operations of the storage device).
(315) At block **1204**, the processing logic receives a first request to form a first unit set that includes a first number of the available units that is configured to be programmed in a first flash programming mode. Forming the first unit set may result in a virtually addressable capacity that corresponds to the effective capacity for the first number of available units when programmed in the first mode.
(316) At block **1206**, the processing logic receives a second request to form a second unit set that includes a second number of the available units that is configured to be programmed in a second flash programming mode that is different than the first programming mode. Forming the second unit set may result in a virtually addressable capacity that corresponds to the effective capacity for the second number of available units when programmed in the second mode.
(317) At block **1208**, the processing logic receives a third request to store first data to a first virtual address associated with the first unit set.
(318) At block **1210**, the processing logic receives a fourth request to store second data to a second virtual address associated with the second unit set.
(319) At block **1212**, the processing logic stores the first data into erase blocks of one or more units of the first unit set and programmed in the first mode.
(320) At block **1214**, the processing logic stores the second data into erase blocks of one or more units of the second unit set and programmed in the second mode.
(321) FIG. **13** is an example method **1300** to intelligently direct flash operations of a modular storage system, in accordance with embodiments of the disclosure. In general, the method **1300** may be performed by processing logic that may include hardware (e.g., processing device, circuitry, dedicated logic, programmable logic, microcode, hardware of a device, integrated circuit, etc.), software (e.g., instructions run or executed on a processing device), or a combination thereof. In some embodiments, processing logic of a storage node of a storage system, as previously described at FIGS. **1**A-**4**, may perform the method **1300**.
(322) Method **1300** may begin at block **1302**, where the processing logic receives read requests and write requests for data of a storage system.
(323) At block **1304**, the processing logic directs the read requests to a first set of storage nodes that host flash-based storage devices that store data for the read requests.
(324) At block **1306**, the processing logic stores written data in erasure coded segments distributed in shards across a set of storage devices hosted by a second set of storage nodes. The shards may be directed to a particular flash-based storage device and are delivered to and written by a particular storage node hosting the particular flash-based storage device.
(325) In embodiments, the write operations may be configured by multiple storage nodes of the storage system to operate during spans of time on a subset of flash die of the flash-based storage devices to preserve the availability of the majority of die of the flash-based storage devices for read operations to read data from the flash-based storage devices.

(326) At block **1308**, the processing logic balances write activity to independent performance zones of the flash memory of the flash-based storage devices to ensure that most read requests are processed in a threshold amount of time without requiring more interrupts of write or erase operations than are allowed by the flash implementation, as previously described.
(327) While this specification contains many specific implementation details, these should not be construed as limitations on the scope of any inventions or of what may be claimed, but rather as descriptions of features specific to particular embodiments of particular inventions. Certain features that are described in this specification in the context of separate embodiments can also be implemented in combination in a single embodiment. Conversely, various features that are described in the context of a single embodiment can also be implemented in multiple embodiments separately or in any suitable subcombination. Moreover, although features may be described above as acting in certain combinations and even initially claimed as such, one or more features from a claimed combination can in some cases be excised from the combination, and the claimed combination may be directed to a subcombination or variation of a subcombination.
(328) Similarly, while operations are depicted in the drawings in a particular order, this should not be understood as requiring that such operations be performed in the particular order shown or in sequential order, or that all illustrated operations be performed, to achieve desirable results. In certain circumstances, multitasking and parallel processing may be advantageous. Moreover, the separation of various system components in the embodiments described above should not be understood as requiring such separation in all embodiments, and it should be understood that the described program components and systems can generally be integrated together in a single software product or packaged into multiple software products.
(329) Thus, particular embodiments of the subject matter have been described. Other embodiments are within the scope of the following claims. In some cases, the actions recited in the claims can be performed in a different order and still achieve desirable results. In addition, the processes depicted in the accompanying figures do not necessarily require the particular order shown, or sequential order, to achieve desirable results. In certain implementations, multitasking and parallel processing may be advantageous.

Claims

1. A storage device configured to: receive, from a storage system controller that is external to the storage device, a read request with a high priority indication; receive, from the storage system controller, a threshold amount of time for the storage device to process the read request; determine whether an in progress flash programming operation would delay processing the read request for the threshold amount of time; and in response to determining that the in progress flash programming operation delays processing the read request for the threshold amount of time, interrupt the in progress flash programming operation, wherein the storage device is to process the read request upon interrupting the in progress flash programming operation and wherein the storage device offloads management responsibilities to the storage system controller that is external to the storage device.
2. The storage device of claim 1, wherein the in progress flash programming operation is a write operation to an erase block programmed to store at least three bits per cell.
3. The storage device of claim 1, wherein the in progress flash programming operation is an erase operation.
4. The storage device of claim 1, wherein the in progress flash programming operation is performed internal to the storage device as part of internal maintenance.
5. The storage device of claim 1, wherein in response to determining that the in progress flash programming operation cannot be interrupted, the storage device is further configured to: fault the read request to indicate that the read request cannot be processed in the threshold amount of time.

6. The storage device of claim 1, further configured to: receive one or more subsequent read requests; and process the read request and the one or more subsequent read requests during the interrupt.
7. The storage device of claim 1, further configured to: process the read request; and resume the in progress flash programming operation upon completion of processing the read request.
8. The storage device of claim 1, wherein the storage device supports zoned namespaces.
9. The storage device of claim 1, wherein the storage device supports an abstraction that organizes virtual address space of the storage device such that virtual segments have a size that enables the virtual segments to be written to sequentially within a same erase block of the storage device.
10. The storage device of claim 1, wherein the storage device and storage system controller are integrated in a flashsystem.
11. The storage device of claim 1, wherein the storage device is further configured to: group the read request with one or more other read requests using one or more scheduling hints.
12. A method comprising: receiving, from a storage system controller that is external to a storage device, a read request with a high priority indication; receiving, from the storage system controller, a threshold amount of time for the storage device to process the read request; determining, by a processing device of the storage device, whether an in progress flash programming operation would delay processing the read request for the threshold amount of time; and in response to determining that the in progress flash programming operation delays processing the read request for the threshold amount of time, interrupt the in progress flash programming operation, wherein the storage device is to process the read request upon interrupting the in progress flash programming operation and wherein the storage device offloads management responsibilities to the storage system controller that is external to the storage device.
13. The method of claim 12, wherein the in progress flash programming operation is a write operation to an erase block programmed to store at least three bits per cell.
14. The method of claim 12, wherein the in progress flash programming operation is an erase operation.
15. The method of claim 12, wherein the in progress flash programming operation is performed internal to the storage device as part of internal maintenance.
16. The method of claim 12, wherein in response to determining that the in progress flash programming operation cannot be interrupted, the method further comprising: faulting the read request to indicate that the read request cannot be processed in the threshold amount of time.
17. The method of claim 12, further comprising: receiving one or more subsequent read requests; and processing the read request and the one or more subsequent read requests during the interrupt.
18. The method of claim 12, further comprising: processing the read request; and resuming the in progress flash programming operation upon completion of processing the read request.
19. A non-transitory computer readable storage medium storing instructions, which when executed, cause a processing device of a storage device to: receive, from a storage system controller that is external to the storage device, a read request with a high priority indication; receive, from the storage system controller, a threshold amount of time for the storage device to process the read request; determine whether an in progress flash programming operation would delay processing the read request for the threshold amount of time; and in response to determining that the in progress flash programming operation delays processing the read request for the threshold amount of time, interrupt the in progress flash programming operation, wherein the storage device is to process the read request upon interrupting the in progress flash programming operation and wherein the storage device offloads management responsibilities to the storage system controller that is external to the storage device.